Integrating Systems and Software Engineering: What Can Large Organizations Learn From Small Start-Ups?

Paul E. McMahon
PEM Systems

In an effort to integrate more effective systems and software engineering, many companies today are examining their internal processes. Recent research conducted on distributed development efforts may provide insight that could aid today's systems and software integration initiatives. Drawing material from his book, "Virtual Project Management: Software Solutions for Today and the Future [1]*," the author explores variations in large and small engineering organizations and presents an alternative view of large projects that may aid companies in their quest for more effective systems and software integration.*

Today, many companies are examining their internal processes in an effort to integrate more effective systems and software engineering. Many of the challenges faced in integrating systems and software engineering exhibit similarities to challenges observed on large distributed efforts.

In this article, engineering organizational variations are first explored, not to judge but to recognize the existence of variation and to note a common characteristic observed in all successful organizations regardless of size or structure. The article then discusses how the identified characteristic is achieved in different organizations.

This preliminary investigation sets the stage for a closer examination of what systems and software integration means in practice. An alternative view of a successful large project is presented that may challenge current published literature. The information presented in this article is the result of research initially conducted on large distributed projects [1].

Organizational Variation

If you were to ask a manager or senior engineer working today in a large advanced technology software-intensive organization to examine the chart in Figure 1, it is likely he (or she) would nod his (or her) head up and down, reflecting familiarity with the functional organizational structure and terminology employed on the chart.

Each rectangle on Figure 1 underneath the engineering manager is implemented through a department each with its own manager and pool of skilled engineers. At this level, similarity across diverse organizations is evident. Nevertheless, while many organizations have a similar top level structure, we have seen that inside these organizations implementation can vary greatly.

For example, in some organizations the Systems Engineering department is totally responsible for producing the software requirements specification (SRS). In other organizations, the Systems and Software Engineering departments collaborate on the production of the SRS with each producing specific *piece-parts* of the final SRS. We have also witnessed a third organizational variation where the Software Engineering department produces the complete SRS, while the systems group provides a review and approval role.

"... it is important to note that what we are asking engineers to do in departments by the same name, but in different organizations, can differ greatly."

Note that in all three cases described, the organizational chart referenced in Figure 1 could be used to describe the organizational structure. At the same time, it is important to note that what we are asking engineers to do in departments by the same name, but in different organizations, can differ greatly.

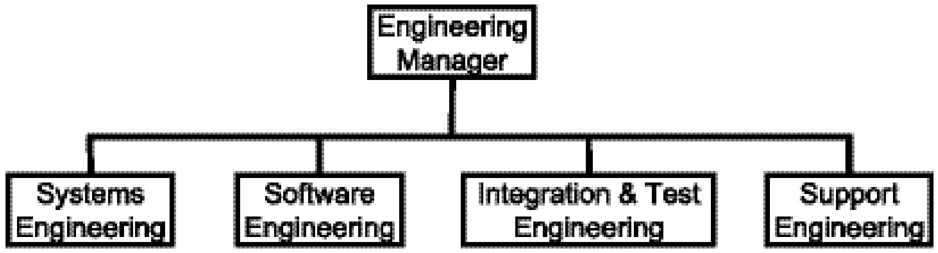
Common Successful Key Characteristics

It is not the intent here to judge the merits of particular organizational approaches, but rather to acknowledge their existence and to point out a key characteristic we have observed common to all successful organizations. In each case, when an organization functions successfully to produce an end product, individuals within that organization *understand and accept their specific role*. That is, they understand the organization's expectations of them.

This mutual understanding of roles, responsibilities, and expectations leads to operational efficiency with minimal duplication of effort. The successful organization appears from the outside to function as a single unit. Its piece-parts may vary on the inside, but in each case they come together without major surprises into a final integrated product.

It is worth noting here that in our experience we have found in many successful organizations that the *definition of* and *responsibility for* each piece-part is oftentimes not written down or described formally. It has been our experience working with large software intensive organizations with long histories of development and evolution that this knowledge may have been written down at some point in time but due to organizational evolution, its current state is most often passed on through less formal means.

Figure 1: *Traditional Functional Engineering Organization*



Communicating Expectations in Large Organizations

In large organizations we tend to see expectations communicated through structure and what we refer to as a process focus. While many large firms have over the past few years undergone organizational streamlining, our experience indicates that sizable written command media with phase-related exit and entry criteria continues to be relied upon.

The process employed inside many of these large organizations can be referred to as predictive, or repeatable. While written command media aids repeatability and communication, we have found that new engineers in many large organizations cannot rely totally on this written word to fully comprehend organizational expectations. Frequently, local cultures are also relied upon to aid communication of expectations.

Communicating Expectations in Small Organizations

Unlike many large organizations, small organizations most often see little structure, little process, and little established culture. The focus of most small organizations is on surviving. We find many small organizations to be heavily reliant on specific individuals. Given this situation, on the surface it would appear there is little a large organization could learn from a small one. However, let us take a closer look at the small organization.

The Small Organization Super Programmer Model

The communication of expectations in many small organizations is simple to describe. We refer to it as the *super programmer* model that implies a *do-it-all* expectation. The problem with this model is that, while expectations are clear, those expectations often lead to over-reliance on, and burnout of, individuals. We frequently find organizations that live by the super programmer model also live by the *code-and-fix* methodology.

During the past few years, we have known colleagues who have given up the relative security of the large, established organization in favor of the increased opportunity afforded by small start-ups. Unfortunately, many have also found that the demands of the small start-up require great personal sacrifice. While some have returned to the more *predictable* large corporate environment, others have found increased job satisfaction through an alternative small-company model that is rapidly gaining in popularity: the small team model.

The Small Team Model

Today, many small organizations are moving away from the super programmer model of operation in favor of a *small team* development approach. This change also often reflects a move away from a code-and-fix methodology, or no process, to what is referred to as an adaptive or lightweight process or method [2]. When we use the term *adaptive method* in this article we mean a method that supports rapid change initiated through small teams.

Lightweight processes can be thought of as *just enough* process, or process without a process-focus. Examples include eXtreme Programming (XP), which have been described by Kent Beck [3] and Jim Highsmith [4].

Characteristics of many lightweight processes include the following:

- Code and test focus.
- Continual iterative design.
- Pair programming.
- Continual planning and integration.

Upon first learning about the characteristics of XP, our reaction was, "This is fluff camouflaging a traditional code-and -fix methodology." However, after observing and interacting with a number of small teams embracing this approach, we have reached a different conclusion.

XP Fundamentals in Practice

While it is true that XP focuses on today, we have found that organizations that take this methodology seriously do not do so at the expense of planning. In fact, teams that follow this process often find themselves planning continuously. Planning in an XP environment is different from traditional planning conducted on many large projects. Plans are short in length, contain specific attainable goals, and often focus on periods of only a few weeks in duration.

This notion might not even sound like planning to those familiar with the process as currently implemented in many large organizations. It also might sound short-sighted and non-optimizing (not looking to the future for improvement), but the immediate feedback provided through short repeated cycles of planning and execution is proving to be effective from the perspective of the engineer in the trench.

This is the first fundamental difference we noticed when dealing with a small company employing an adaptive methodology. The second came when talking to software engineers working on an XP team: We found a surprising level of awareness and ownership of schedule and budget. The software engineers were aware of the schedule and budget and felt ownership of it because they had participated in its development. On the other hand, in many large organizations, we have found that it is not uncommon for engineers to have little insight into the project schedule and budget.

Upon first learning about XP, we found its code-focus to be difficult to accept. However, after observing an XP team in action, it left us with a different impression.

The notion that programmers using XP do not design is a misunderstanding. One member of a small XP team explained it to us this way: "Just because we focus on the code doesn't mean we don't give each task considerable forethought. We just don't write down the result formally, and we don't use a formal design tool." Then after he hesitated, he added, "But we might draw a sequence diagram or two, if we think it might help." Another member of the same team said, "We keep our designs as simple as possible, and we try not to spend any unnecessary time in design."

We had an opportunity to witness the design process in action with a small team, and it reminded us of an informal brainstorming session you might see in any organization. The meeting had not been scheduled ahead of time. It just happened because one member of the team wanted some help. Within a few minutes, three team members had gathered around a white board, and 25 minutes later there was a design solution sketched out on the board. We suggested that someone capture the diagram more formally.

Another interesting aspect of XP is pair programming. When we first heard about pair programming, we expressed agreement with the concept to a young client. Our thinking was that this technique would provide a backup in case one team member was pulled off the project or got sick. But the client was quick to explain that pair programming had nothing to do with having a backup.

We have since learned that some pair programming team members are adamant about having both team members present side by side during 100 percent of the programming activity. That is right! Not only does it take two people to complete one program, but also if one of the two is missing, in some cases, the other does not want to move on. Doesn't that sound incredibly inefficient?

But then the client went on to explain: "It's the dialogue that I don't want to miss. By having my teammate right next to me, it forces me to verbalize my thought process at the moment I type the code in. Often through this process, errors are detected at the same moment when they are about to be created."

As I listened to this process being described, a light bulb was flicking on in my head – this process is what peer reviews and early error detection was always meant to be!

Tailoring XP

When reading about a new methodology, you often envision something different from the way organizations actually apply it. XP, according to the book, includes 12 key practices. However, not all companies that claim to employ XP follow all the practices exactly as outlined by Beck [3]. For example, while many small organizations have recognized the value of pair programming, others have also recognized that what their engineers actually do extends beyond programming itself.

Often we find in practice that the pair recognizes that one of the members has more of a *systems* inclination (i.e., customer interface, requirements management), while the other prefers the traditional programmer role. These recognitions are usually based on individual strengths and desires. As a result, we tend to see a *systems focus* and a *software focus* being supported within the small team and small company environment, but without the formal system and software department boundaries that are prevalent in large organizations.

Effective Systems and Software Integration in Practice

We have witnessed large organizations communicating expectations through defined organizational structures, supported by *heavyweight* command media (policies, practices, and procedures). We have also seen the key role of culture in communicating expectations in large organizations.

In small organizations applying lightweight methodologies, on the other hand, communication occurs through short-range planning that leverages individual teammate strengths. Given what we have witnessed inside both large and small organizations, we are led to a question: "What does effective systems and software integration mean in practice?"

We believe that the answer to this question should be independent of the size and structure of the organization. We propose the following definition: Effective systems and software integration means that the right interactions are occurring at the right time, the right questions are being asked at the right time, and the right factors are being considered and acted upon at the right time.

Systems Engineering Inside Large Organizations

We have, on a number of occasions, taken the opportunity to ask an engineer in a large organization, "What is systems engineering?" Often the answer received has equated to, "whatever the systems engineering group does," in that particular organization.

Unfortunately, this answer can be problematic for two reasons. First, from an educational viewpoint, how are we to prepare systems engineers in our universities when the expectations of a systems engineer can vary dramatically from one organization to another?

Second, when task expectations are too tightly coupled to an organizational structure or department charter, the increased likelihood for tasks to fall through cracks exists. This is because no organization is perfect. We have also found that a tight coupling of task responsibilities to organizational partitioning tends to give rise to the "it's not my job" syndrome in large organizations.

Systems Engineering Inside Small Organizations

On the other hand, when we have asked, "What is systems engineering?" to an engineer who has experienced only the small organization environment, the most common response has been a puzzled look on his/her face. This is because in most small organizations, engineers do not think in terms of distinct systems and software tasks; rather, they think in terms of getting the job done.

One reason small organizations do not tend to exhibit the task-related difficulties we have observed in large organizations is because they have not artificially partitioned detailed tasking responsibility based on organizational boundaries. Stated differently, in small organizations that use adaptive methods, the team works out the task responsibilities knowing that together the team is responsible for everything.

Applying a Lightweight Approach to a Large Project

Published literature available today on lightweight methodologies [3, 4] indicates these approaches may not be scalable to large projects. While we do not recommend XP practices be applied in full on large projects, a number of the most successful large projects we have witnessed tend to already embrace many of these same practices. Although it is unwritten, i.e., not found in any formal corporate command media, a number of the most successful large projects we have observed also tend to exhibit an *unspoken adaptive subculture*.

Characteristics of these projects include incremental development, plans that focus on today, and a code focus. A code focus may seem unusual to a large project especially in large disciplined organizations, but in practice it has proven to be particularly effective when heavy reuse is involved. Testing candidate reuse code early, peer reviewing proposed changes early and often, integrating early, and staying integrated through a series of incremental builds have proven to be effective techniques on projects of all sizes.

A Key to Success

This article recommends managing a large project as a collection of small adaptive projects. This recommendation is not meant to imply that the adoption of such methods alone is sufficient to ensure large project success. On the contrary, if small teams within a large team were allowed to continually adapt their plan independently, then chaos would certainly result.

On one large project that we worked as a team member, the software work was partitioned across the country at three distinct sites. But before the work partitioning took place, a small group of senior project personnel established overall system architecture with very specific constraints, including computer platforms, compilers, tools, and interfacing requirements.

As the project evolved, there also evolved a number of small teams each with approximately seven to 10 engineers. Each small team had its own unique development issues. The project leader empowered these lower level informal small teams to make key decisions constrained only by the project requirements and the defined project architecture.

Many of the characteristics we have seen in successful small companies employing adaptive methods are also evident within small informal teams inside large projects. You will not necessarily find the small teams we are referring to on a *formal* organizational chart.

In-the-large XP practices can work, but only if implemented within the context of a higher level framework.

In short, we want our small teams inside large teams to take on increased responsibility, but the key to success on large projects is ensuring small team adap-

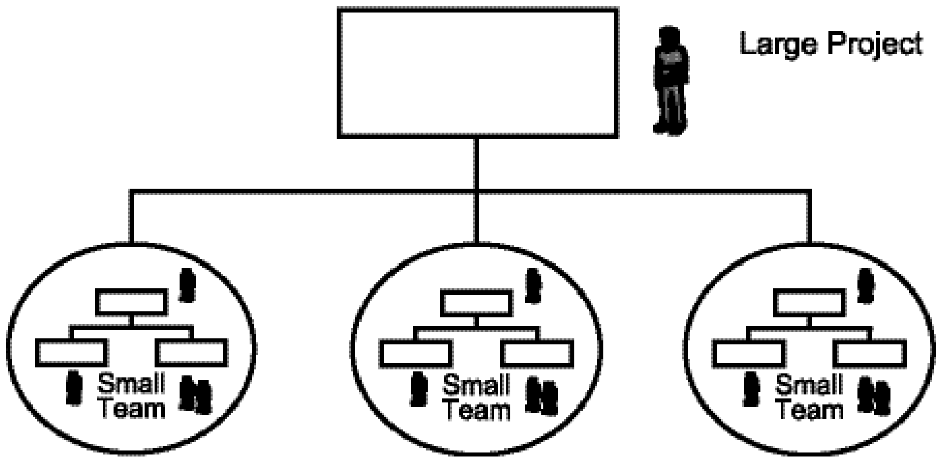


Figure 2: *An Alternative View of a Successful Large Project*

tations are consistent with well-defined, well-communicated system architecture.

An Alternative View of a Large Project

Through the use of a small architecture group and other related techniques to communicate architectural decisions and responsibilities [1], large projects can effectively be managed as a collection of small adaptive projects (Figure 2). While many large companies may not formally describe their operating structure in these terms, many successful large projects operate in this manner today.

It is also worth noting that we do not recommend that the notions of small teams and pair programming be overly formalized in large organizations. This would, in fact, undo the value we seek. Informality is an essential ingredient to the success of the small team in any organization.

Pair programming can be effective in an organization of any size, but it is also important to realize that some people just do not team well, and we do not believe that forcing small teams or pair programming makes sense in any organization.

Different companies have different cultures and differing past experiences and beliefs surrounding their workspace. Some believe in open workspaces, while others pride themselves in the private offices they provide their professional personnel. Nevertheless, we are witnessing a definite movement within the software community toward more group activities in support of increased productivity.

One new engineer in a large company told us that he sat in his cubicle for the first three months of his new job continually wanting to ask questions, but not wanting to be perceived as a nuisance. Other new engineers in the same company, we found out later, felt the same way.

Soon thereafter, an engineering lab environment was set up. It was not long before all the new software engineers were spending more and more time together in the lab. One engineer said the lab environment made it much easier to ask questions and to listen to answers to questions asked by others. Progress on the project increased at lightning speed shortly thereafter.

Conclusions

A few months after providing assistance to a small company utilizing XP, we asked one of their engineers a simple question: "Had anything changed over the past few months?" The engineer responded: "If I had to point to one thing, I've noticed that I'm spending less time dealing with urgent and unimportant matters, and more time on the things that count."

Adaptive methods not only can work in large organizations, we have found they are often key to large project success. We recommend large organizations that are not operating as effectively as desired consider the selective adoption of adaptive techniques.

What really first caught our attention with adaptive methods was the focus on the engineer in the trench. Too often, well-intentioned process improvement initiatives never seem to reach the real workers.

In reality, the short cycles of planning are not shortsighted, rather they are based on the length of time into the future where we have control over where we are going.

It is also important to note that it is not that the process we see today in large companies is not working, but it works at its own pace. Adaptive techniques and small informal teams inside large organizations can complement a formal organizational process focus, and can also be an effective method to facilitate change in an organization that is not evolving at the pace needed to remain competitive in today's world.

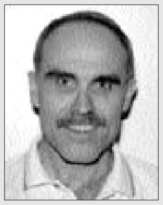
Write plans that work today, and do not discourage your team from continually updating their plans to reflect increased knowledge tomorrow. Encourage small teams to leverage your organization's strengths regardless of where those strengths lie inside your organization.

Small teams and adaptive methods can not only help your systems and software integration efforts, but they can also prepare your organization to be more effective in tomorrow's collaborative world.◆

References

1. McMahon, Paul E. Virtual Project Management: Software Solutions for Today and the Future. Boca Raton: St. Lucie Press, An Imprint of CRC Press LLC, 2001.
2. Fowler, Martin. "Put Your Process on a Diet." Software Development Magazine Dec. 2000: 32-36.
3. Beck, Kent. eXtreme Programming Explained: Embrace Change. Boston: Addison-Wesley, 1999.
4. Highsmith, James A. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York: Dorset House Publishing, 1999.

About the Author



Paul E. McMahon is an independent contractor providing technical and management leadership services to large and small engineering organizations. Before initiating independent work as PEM Systems in 1997, McMahon held senior technical and management positions at Hughes and Lockheed Martin. Today he employs his 28 years of experience to help organizations deploy high quality software processes integrated with systems engineering and project management. He has taught software engineering at Binghamton University in New York, conducted software process and management workshops, and has published more than 20 articles and a book on virtual project management.

118 Matthews St.
Binghamton, NY 13905
Phone: (607) 798-7740
E-mail: pemcmahon@acm.org