

## EXPLORING INTEGRATIVE MECHANISMS WITH A VIEW TOWARDS DESIGN FOR INTEGRATION

**Tyson R. Browning**

Technology, Management, and Policy Program  
Massachusetts Institute of Technology  
Cambridge, Massachusetts

### ABSTRACT

The integrated product development (IPD) paradigm has gained recognition as a preferred approach to product development. In complex system development programs, the concurrent engineering aspect of IPD is often approached through the use of *integrated product teams* (IPTs), each assigned to develop various components of the overall system. Many have struggled to determine the characteristics of highly effective IPTs and the circumstances in which particular perspectives should be incorporated within an IPT. However, much less research has addressed the nature and management of the relationships between IPTs—the integration of IPTs within a program. While many have lamented that coordination problems have played a large part in diminishing the performance of their overall programs, a systematic approach for considering these issues *a priori* in program design is lacking. This paper presupposes a familiarity with interteam integration issues and (1) describes a framework for thinking about organization integration within a program, (2) presents *integrative mechanisms* (IMs) useful for managing IPT interfaces, and (3) begins to develop a systematic approach for designing programs that explicitly considers integration needs, *design for integration* (DFI).

### INTRODUCTION

For several years now, the *integrated product development* (IPD) paradigm has been gaining recognition as a preferred approach to complex system product development. When developing a complex system, no single person or small group is able to provide all the necessary perspectives. Furthermore, competition and shrinking budgets are forcing product development schedule compression, leading to task overlapping, or concurrent engineering. This aspect of IPD has been approached through the use of integrated product and process development teams—*integrated product teams* (IPTs)

for short—each consisting of a cross-functional, upstream/downstream representation and assigned to various subsystems or components of the overall system. Many have struggled to determine the characteristics of highly effective IPTs and the circumstances in which particular perspectives should be incorporated within an IPT. However, much less research has addressed the relationships between IPTs. While integration problems continue to play a large part in diminishing program performance, a systematic approach for considering these complications *a priori* in program design is lacking.

Program integration challenges stem from systems architecting and engineering issues. Complex systems derive their value from the relationships among their parts. These interactions make a system much greater than its parts. Therefore, as Rechtin (1991) notes, “The greatest leverage in system architecting is at the interfaces” (p. 29). While true for a system architecture, it also applies to the IPTs that develop a system’s constituent components. A well integrated development organization can provide a significant source of competitive advantage. Furthermore, organization and system architecture are linked. Recognizing this connection enables the discerning organization to exploit architectural decisions to enhance interteam integration. Where possible, the organization structure should mirror the system product architecture (Grady, 1994). Thus, the efficiency of the organization will to an extent depend on the nature of the interfaces within the system architecture.

Program design becomes increasingly difficult as system complexity increases. (Complexity here implies numerous, highly-coupled components.) The IPTs working together to develop such a system face a daunting task. Team A needs to know what values team B has set for parameters  $x$  and  $y$ ; team B needs to know what values team C is using for parameters  $w$  and  $z$ ; but team C needs to know the result of team A’s activities to determine  $w$  and  $z$ . Coupled “chicken and egg” problems may

imply a slow, iterative development process. Without cautious system partitioning, the number of required interfaces and thus the number of iterations required to converge to an acceptable design can increase exponentially with system complexity. As the amount of task and IPT interdependence increases, the traffic on inter-IPT communication channels increases. A study by Tausworthe adds the following caution: “A team producing at the fastest rate humanly possible spends half its time coordinating and interfacing” (Qtd. in Rechlin, 1991, p. 284).

The necessity of so much coordination points to the management facets of the program integration issue. The design of complex system products involves hundreds or thousands of individuals making millions of design decisions over months or years. Couplings within the system architecture imply these activities require personal interaction among the designers: design choices involve tradeoffs which affect many other design, process, cost, risk, manufacturing, and operational parameters. Managers’ “primary development challenge is to integrate the many sub-problem solutions into a well-designed system. ... The trouble is that such interactions are often poorly understood and are rarely known in advance” (Eppinger *et al.*, 1994). Teams are seen as a means of achieving cross-functional integration, but, as Cole (1995) notes, “Integration of teams’ activities can ... be a nightmare. Like functionally oriented organizations, teams can get tunnel vision and forget that there are other missions and reasons for an organization’s existence.” In managing the interactions, it helps to have a design process and an organization in place that has been designed to make this effort as straightforward as possible. In any case, better understanding the issues underlying IPT interfaces would seem to hold great potential for improving the product development process.

The keys to managing IPT interfaces effectively lie in ensuring the proper interactions between well-partitioned IPTs and functional groups followed by facilitating the smooth transfer of information across these interfaces. First, the right teams must be formed. Second, the right interfaces must be arranged between them. Then, the right coordinating mechanisms must streamline the necessary transfers across those interfaces.

A few authors have described how designing the design process on the basis of intelligent task partitioning and information flow can improve product development effectiveness and competitiveness (von Hippel, 1990; Whitney, 1990). Following this lead, this paper presupposes a familiarity with multi-team coordination issues and (1) describes a framework for thinking about integration within a program, (2) presents *integrative mechanisms* (IMs) useful for managing IPT interfaces, and (3) outlines a systematic approach for designing programs that explicitly considers integration needs, *design for integration* (DFI).

While organization design in a program and the application of IMs may be handled implicitly, an explicit, informed, systematic approach has several advantages. Systematic methods which serve as a baseline for—not as a substitute for—intuitive judgments are prized because they lend themselves to comparison and benchmarking. Change

becomes measurable along dimensions of interest. Furthermore, systematic methods provide an orientation framework that the varied perspectives addressing the issue can agree on and work from.

Throughout, this paper draws from research conducted largely in the American defense aircraft industry.<sup>1</sup> While many aspects of this industry make it unique, the framework presented here applies to complex system product development projects in other contexts (e.g., commercial aircraft, spacecraft, ocean vessels, etc.). In all such programs, the necessity of integrating the perspectives of the groups developing the various subsystems has been noted as acutely problematic.

## A FRAMEWORK FOR PROGRAM INTEGRATION

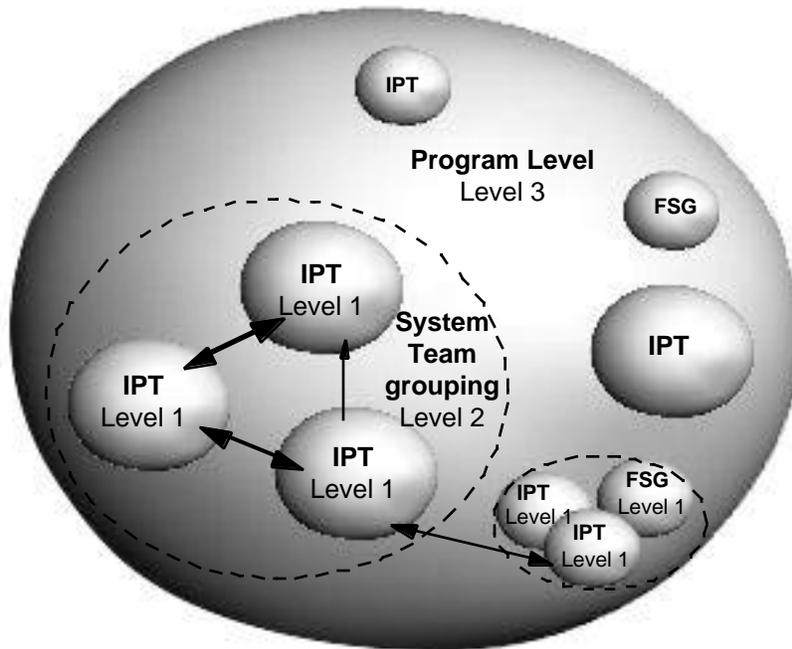
This section presents a hierarchical framework for program integration. The hierarchy reflects the fact that not only must product teams be cross-functionally integrated as IPTs, but also the IPTs themselves must be integrated on a higher level. Hence, there exists at least a second level of integration within each program. Furthermore, large, complex system development programs experience great difficulties integrating all functions into small IPTs. Often, this is not practical or even feasible. Various *functional support groups* (FSGs) such as test labs remain outside IPTs’ jurisdictions. Such situations challenge an IPT approach to IPD. Yet certain IMs can successfully integrate these groups at a higher level in the program. For programs with many IPTs and FSGs, integration-oriented organization structures are useful ways to partition the organization and include disparate groups.

Thus, it is convenient to think of integration as necessary at at least three levels within a program. McCord and Eppinger (1993) distinguish IPTs and “system teams”—sets of IPTs and FSGs pulled together because of mutual, high traffic interfaces.<sup>2</sup> IPTs represent a first level of integration; system teams exemplify a second. Considering the overall program as yet another level, an organization integration framework must consist of at least three levels. Figure 1 illustrates a three level framework: IPT level (first level), system team level (second level), and program level (third level). Individual IPTs (and FSGs, if applicable) can be aggregated in system teams, which, along with other IPTs (and FSGs) and system teams, comprise a program. Other IPTs and FSGs, not fitting within system teams, are tied into the organization at higher levels (here, the program level). Arrows in Figure 1 represent the most significant information exchange needs. (Less intense information dependencies are not shown.) Dashed ovals distinguish system team groupings.

---

<sup>1</sup> This paper draws from material in (Browning, 1996a).

<sup>2</sup> Technically, it is probably more correct to refer to these groupings as “subsystem teams,” since the organizational entity formed often consists of a group of IPTs working on the respective components of a particular subsystem—one of many in the overall system which the program as a whole is developing. Alas, the system team designation is already prevalent in industry.

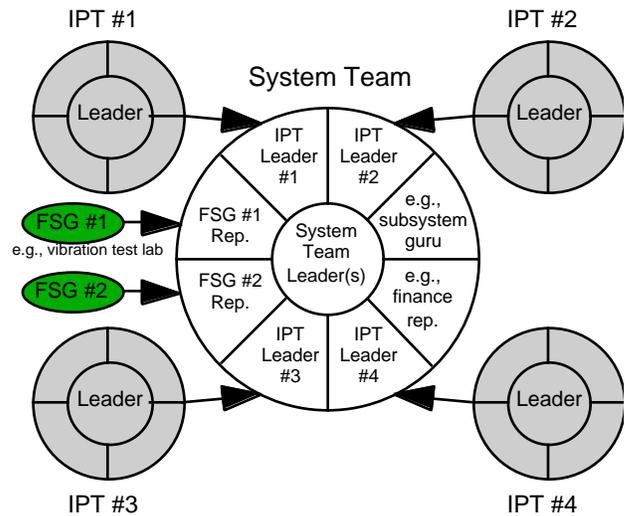


**FIGURE 1: THREE LEVELS OF INTEGRATION**

Note that the three level structure provides only a minimum amount of hierarchical distinction. Complex system development programs often require additional levels between one and three—i.e., different levels of system teams. For example, the F/A-18E/F program expands the hierarchy to four or five levels to capture finer distinctions between system team sizes and/or priorities. Note also that the hierarchical nature of the proposed framework is intended to mirror the hierarchy of the product architecture, not the power structure of the company or program organization—although these may correlate to an extent.

Figure 2 shows an example system team composition. This system team serves to integrate four IPTs and two FSGs as well as a couple of functional perspectives not inserted at the IPT level. This system team is composed of representatives from the IPTs (here, the team leaders, although this need not be the case) and FSGs, additional functional representatives, and one or more system team leaders (who also likely represent the system team at the program or higher system team level).

Integrating at the various levels reveals tradeoffs. For example, should a given functional representative be made available to and integrated within a particular IPT? or will some resources have to be shared by many IPTs and instead integrated at level two or three? The addressing of these tradeoffs and the appropriate application of IMs to facilitate inter-IPT and system team integration are the subject of the proposed DFI process.



**FIGURE 2: EXAMPLE SYSTEM TEAM COMPOSITION<sup>3</sup>**

**INTEGRATIVE MECHANISMS FACILITATING COORDINATION AT LEVELS 2 AND 3**

*Integrative mechanisms* (IMs) are strategies and tools for effectively coordinating actions between multiple IPTs and

<sup>3</sup> This diagram is adapted from PRTM’s illustration of what they call Interlocking Development Teams. While the illustration is useful here, the related methodology is different.

FSGs. IMs are catalysts, facilitating information flow across organizational, locational, cultural, traditional, and other barriers. They must regulate information flow such that it is available but not overwhelming. The categories of IMs listed below represent the tools in an integrator’s “tool kit.” Elsewhere (Browning, 1996a; Browning, 1996b; Browning, 1997; McCord and Eppinger, 1993; Susman, 1992, pp. 140-156), the IMs in Table 1 have been discussed and explored in varied contexts. For the purposes of DFI, IMs are presented as the type of coordinating mechanisms available for integrating IPTs, FSGs, and system teams in programs.

**TABLE 1: INTEGRATIVE MECHANISMS**

1. Systems engineering and interface optimization—designing the organization to mirror the system product architecture. An intelligently decomposed and partitioned architecture (such that interfaces between subsystems are minimized) will facilitate interteam interface management.
2. Improved information and communication technologies—linked CAD/CAM/CAE systems, e-mail, tele- and videoconferencing, common databases (easily accessed and shared), common nomenclature, etc.
3. Training—especially in team-building (and “system team-building” and “program-building”); raising awareness about integration needs and roles
4. Co-location—physical adjacency of IPT, FSG, system team, and/or program members
5. Traditional meetings—face-to-face gatherings for information sharing and/or decision making
6. “Town meetings”—not to share technical information, but to boost camaraderie and to increase awareness of program-wide issues
7. Manager mediation—“up-over-down” (hierarchical) issue mediation schemes; heavyweight product managers (HPMs) or integrators
8. Participant mediation—liaisons, engineering liaisons, conflict resolution engineers
9. Interface “management” groups—integration teams tasked with ensuring ongoing or incident-specific mediation of interface issues
10. Interface contracts and scorecards—explicit delineation of interface characteristics and metrics for evaluating interface effectiveness

**TOWARDS A SYSTEMATIC APPROACH TO DESIGNING PROGRAMS FOR INTEGRATION**

In the efforts to create effective interfaces between teams and insure the proper flow of information, a systematic

approach aids in coordinating the endeavors and helps guarantee the inclusion of important considerations. Mohrman *et al.* (1995), having looked at team-based organizations extensively, note that “the most effective teams we saw used systematic planning processes for determining responsibilities and for scheduling and integrating their work” (p. 178). Systematic approaches introduce rigor and structure to the decision-making process. Planning and forethought regarding IPT interfaces likewise facilitates effective program execution. This section outlines the key steps in a proposed approach to *design for integration* (DFI) that explicitly considers interteam integration issues. DFI applies to the design of the design process itself. While additional details, unique to a given program, will provide constraints and guidance in this process, the general steps shown in Figure 3 apply to the majority of situations. Note, then, that the process proposed below is not *the* DFI process, but *a* DFI process. The strawman process herein is intended to illustrate the link between considerations important to incorporate into one’s DFI efforts. Hopefully, one will also discern from the process the necessity of planning a development organization while explicitly and systematically contemplating integration issues.

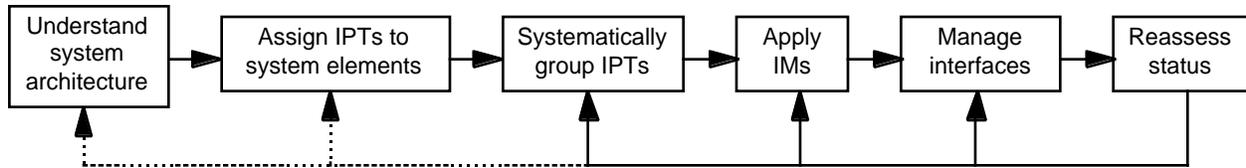
As the control loop implies, a DFI process is iterative. The dashed portion of the feedback loop indicates changes that will most likely take effect on a subsequent project, whereas the solid line portion represents easier opportunities to modify projects already underway. Each of these steps is elaborated below, along with suggestions for implementation.

**1) Know System Architecture**

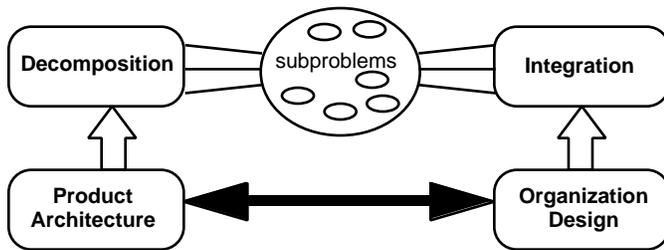
Gulati and Eppinger (1996) explore the coupling of product architecture to development program organization structure, noting that decisions in one realm affect and even constrain opportunities in the other. Thus, as one considers organization integration issues, one must first look at the system architecture that organization will be charged to develop. Figure 4 shows their proposed relationship: architecture and organization are linked through the process of problem decomposition and system integration.

Decomposition and integration are generalized inverse problems. Therefore, the first step in the DFI process is to understand as completely as possible (or practical) the nature of the system architecture, especially its decomposition and internal interfaces, for these will directly affect the organization and the ease of integrating the teams working on the various subproblems.

How, specifically, does a product architecture and its associated task set affect the organization and its communication patterns? Morelli *et al.* (1995) investigate the extent to which coordination-type communication between project groups is predictable given a known task set. With the proposed tasks before them, project participants were able to predict 81% of the communication that in fact took place. This result signifies the possibility of designing an organization on the basis of a proposed system architecture and its associated task set.



**FIGURE 3: THE DESIGN FOR INTEGRATION PROCESS**



**FIGURE 4: ARCHITECTURE TIED TO ORGANIZATION THROUGH DECOMPOSITION/INTEGRATION PROBLEM (GULATI AND EPPINGER, 1996)**

Given an unprecedented or revolutionary system, however, possessing a thorough understanding of the architecture and the tasks to develop it is understandably difficult. The need to better understand system specifications has thus brought the recent focus on requirements management and on systems engineering as a discipline. Knowledge of tasks and their duration is essential to the creation of the *statement of work (SOW)*, *work breakdown structure (WBS)*, and *integrated master schedule (IMS)*. This knowledge is likewise crucial for interface planning and management. For upgrades and other largely preceded systems, it is much easier (although not necessarily easy) to outline tasks and their information requirements. Where the architecture and/or tasks are yet to be determined, organization designers must build in flexibility so the organization can adjust once the characteristics of the required IPT and system team interfaces settle out. Baseline organization designs for programs developing unprecedented systems will have to be the most flexible of all. (This requires an incentive system that encourages organization flexibility.)

The issues of systems architecting are crucial as inputs to any design process. The importance of intelligently grouping functional requirements and decomposing architectural elements to meet those requirements cannot be overstated. It is here that DFI begins, for it is here that much of the difficulty of the remaining integration tasks is determined. However, this paper will not address the tenets of intelligent decomposition or systems architecting, which several authors have discussed. (See, for example, Alexander, 1964; Altus *et al.*, 1995; Kusiak and Wang, 1993; Michelena and Papalambros, 1995; Pimmler and Eppinger, 1994; Rehtin, 1991,.) Leaving aside the notion that product architectures can be designed with a

consideration of organization integration concerns, the remainder of this paper will focus on how, given an architecture, one might proceed to design an organization to develop it.

## **2) Assign IPTs to system components**

Once architectural subsystems and components have been decomposed and defined to the extent possible, an IPT is assigned to the development of each component. Ideally, each IPT would include all of the cross-functional and other resources necessary over the life of its component development project. However, resource constraints limit this possibility and will create the need for additional external interfaces. Also, depending on the size of the program, and realizing that IPTs should be kept to a small size, one must determine what scale of component the IPT will develop. These issues and others combine to make the IPT assignment problem nontrivial.

IPT size is an important assignment consideration, one that directly constrains integration efforts. Since IPTs have a cross-functional, upstream/downstream (time phase), customer, and supplier representation, they require more people than an equivalent, non-cross-functional team. Klein and Susman (1995) have found the average size of an IPT in the defense aircraft industry to be 26 full-time members (40 including part-timers). However, Katzenbach and Smith (1993) set the size of an ideal team at close to ten people, within a range from two to 25 people (p. 45). Peters (1995) notes how the reorganized Space Station program uses IPTs of 8-12 members. Sheard and Margolis (1995) cite D. Quinn Mills as putting the "ideal" team size at 5-7 people. These numbers represent the optimal number of people who can usually work together as an effective team "should." Larger groups can still be called teams, of course, but they will find it difficult to achieve the same level of intimate teamwork espoused by the proponents of "teaming." As Cole (1995) understands, "Group dynamics being what they are, groups of say, 30 or more, tend to break themselves down into smaller groups anyway." Certainly, teams with 70 or 100 members will not be able to perform as IPTs should, and should not be termed as such. Therefore, given the size limits for effective IPTs, one is automatically constrained to assign them to a task of approximately subassembly level development scope.

Table 2 shows in its first column the architectural hierarchy nomenclature used in space systems development (Rehtin, 1991; Shishko *et al.*, 1995). The second column exhibits an example mapping to organization structures.

Large programs will require all of the architectural hierarchy listed, while smaller projects will not. Note that there is quite a bit of flexibility as to the level correspondence of IPTs and system teams. Larger programs will likely require multiple levels of system teams. In this case, the distinction between system teams and IPTs begins to blur in many programs. Since many resources and perspectives can only be integrated at higher levels, sometimes system teams get called IPTs. While the distinction does not have to be made, one should keep in mind the spirit of the IPT is to integrate cross-functionally and upstream/downstream at the lowest level possible. Also, the step of assigning IPTs to appropriate level tasks precedes the step of grouping those IPTs into system team organization structures. As DFI Step 3 describes, system team structures should stem from known IPT task assignments and the interfaces that can then be predicted between them.

**TABLE 2: ARCHITECTURE TO ORGANIZATION HIERARCHY MAPPING (EXAMPLE)**

Architecture	Organization
System	Program
Segment	
Element	
Subsystem	System Team
Assembly	
Subassembly	IPT
Part	

Additional insight into IPT assignment can be gained by directly examining the *tasks* required to develop particular components. The information flow required between tasks (a time-based consideration) can also inform the composition of IPTs and the level of resource or perspective integration vis-a-vis the levels proposed above. Note that different levels of task complexity between interdependent teams can lead to interface difficulties. For example, one team might have a “big picture” perspective while its sibling IPT dwells in the details. In some cases, it may be more appropriate to assign some tasks to FSGs.

Much more could be said about the chartering of IPTs. Again, this paper does not focus on the creation and management of IPTs themselves, although this is a salient issue and the subject of much other research. Instead, the point here is to show how this important step fits into a DFI process.

### 3) Systematically group IPTs

After IPTs and necessary FSGs have been assigned to develop low level components of the system architecture, these teams and groups should be aggregated into system teams. System teams will tend to form around major subsystems, especially if the architecture is wisely partitioned. However, constraints affecting architectural decomposition are likely to differ in some ways from those in the organization. Hence, it is best to conduct an additional analysis to make the system team grouping determination from the “bottom-up.”

This type of analysis is described using a *design structure matrix* and clustering algorithms by McCord and Eppinger (1993).

This is also the stage in a DFI process where resources that are not available to each individual team are included at a system team or program level. Referring again to Figure 1, some IPTs, FSGs, and/or individual contributors cannot or need not be integrated at the lowest levels. This is often the case for tasks requiring minimal interfaces. It is also true in the opposite case, when groups have highly integrative tasks, such as process coordination or the design of an extremely interactive component (e.g., a data transmission subsystem). In these cases, it is best to integrate the highly interactive group at a higher level, such as within a system team or at the program level.

Grouping IPTs and FSGs into system teams is a key step in the DFI process, for it determines which interfaces will be crucial in the program. Table 3 discusses the desirable characteristics of information transfer interfaces.

The goal of this DFI step is to come up with the best system team breakouts and the beginnings of a viable *scheme for interface mediation* (SIM) for the program. A SIM document explicitly outlines the expected interfaces, their desired characteristics, and the means of monitoring them. It is not given a formal outline here, although it should contain the details of the trade-offs involved in the design of the integration scheme as well as the implementation plan. As a formal document, however, it should not be an end in itself. The SIM document will serve the program’s organizers as an invaluable interface management tool. Results of SIM development are included as part of the program’s Systems Engineering Management Plan (SEMP)<sup>4</sup>, Integrated Master Plan (IMP), and/or Program Execution Plan (PEP).<sup>5</sup> How well the system team and integration level determinations are made will directly affect a program’s success.

### 4) Apply Integrative Mechanisms (IMs)

Once IPTs, FSGs, and system teams have been determined and interface characteristics have been established, the appropriate IMs from the list in Table 1 must be implemented to facilitate the interactions—based on the unique physical, political, architectural, and other attributes of the program. The deliberations in this step—especially the reasons for choosing particular IMs—should be recorded in the SIM document so they can be communicated to the entire program and revisited later (Step 6).

<sup>4</sup> For more details on a SEMP, see, for example, (Blanchard, 1990; DoD, 1994; Kockler, 1990; Shishko, 1995).

<sup>5</sup> As described by Peters (, p. 3), a PEP was used for the organizational transition from a functional organization to IPTs in the Space Station program. “The PEP represents customer needs in terms of program strategies and issues, unique contract requirements, program requirements, program schedules and budget, unique and standard processes as well as IPT responsibilities and structure. ... i.e. their operating node, schedule development, resource allocation, organizational structure, process, and overall team authority and responsibilities.”

**TABLE 3: DESIRABLE INTERFACE CHARACTERISTICS**

1.	<b>Defined</b> , in terms of what information needs to flow, where, when, and how (i.e., “the right information at the right place at the right time”).
2.	<b>Tight-fitting</b> , in terms of task assignment. Tasks should not overlap or “underlap.”
3.	<b>Permeable</b> , in terms of permitting and regulating information flow. Information should arrive “just-in-time”—not too early or too late. It must be the correct amount of information—not more and not less. It must flow readily and smoothly, yet not inundate its recipients. The interface must allow just the right amount of the right information to flow.
4.	<b>Mutable</b> , in terms of altering information flow. There must be a means of adjusting what information gets transferred, when, and how.
5.	<b>Efficient</b> , in terms of time lag from provider to recipient. This path should be free of undue bureaucracy or other delays.
6.	<b>Documented</b> , in terms of keeping a record of information flow. Information useful once may be useful again. To avoid “reinventing the wheel,” one needs a record of the flow. Documentation facilitates learning and accumulation of a knowledge base.
7.	<b>Measurable</b> , in terms of allowing analysis of success and flow rate. Success should be based on objective criteria where possible. Metrics to evaluate information flow and interteam interactions are crucial to provide appropriate incentives and further process improvement.
8.	<b>Adapted</b> , in terms of the program’s task, size, and stage. One size does not fit all. Each important interface deserves explicit, personalized, unique attention and optimization.

### **5) Manage interfaces**

After designing the best possible organization structure and choosing appropriate IMs *a priori*, program management must keep information within the program flowing smoothly. This includes mediating technical issues via suitable IMs and monitoring the effectiveness of communication. Evaluating the effectiveness of the SIM is difficult on a real-time basis. Often only “lagging indicators” such as adherence to budget and schedule are available as macro level proxy metrics. This is one reason why successful integration is notoriously difficult. One proposed metric includes counting the number of change notices involving interfaces (which is most of them). Their source and scope must be noted. However, change notice data are difficult to obtain and analyze. Integrated prototype testing and design reviews also provide opportunities to formally evaluate organization integration effectiveness. Are

the issues that result from these tests and reviews the result of a lack of information? miscommunication? unresolved technical issues? Checklists and scorecards can assist reviewers in asking the right questions about integration. Tracking the time spent (perhaps through charge numbers) by members of integration teams and other individuals on IM-related tasks might also yield interesting data on which mechanisms are being utilized and where the issues reside.

### **6) Reassess status**

As the program evolves, the SIM will need to be reevaluated. Some IMs will no longer be appropriate. Some interfaces will become more important, others less. New interfaces will form. Some will disappear altogether. For example, IPTs will tend to utilize design support FSGs early in the design process; later, they will spend more time with production process FSGs. As the organization changes, so must the IMs that connect it. It should be the periodic, if not ongoing, task of a group of organization designers to reevaluate the SIM and effect the necessary changes. Making sure this process is well-documented will greatly improve its effectiveness in the long run, especially for long-term development projects where turnover among the membership of the SIM group is likely.

### **CONCLUSION**

The proposed DFI process is aimed at providing a systematic approach to organization design that explicitly accounts for interteam integration issues. Such issues have been the root cause of numerous complications and setbacks in complex system development projects. Despite the attractiveness of incorporating these considerations earlier and explicitly in program design decisions, barriers exist to doing so. First, one must establish that such issues indeed account for substantial performance gaps. This paper addressed this issue in a cursory fashion but largely assumes that the immediate CERA audience is aware of such matters. Second, some organization changes deemed advantageous by the process analysis may run counter to tradition. New programs are the best opportunities to make necessary changes. This also argues for organization design foresight early in the project. Third, a suitable group of individuals to conduct the DFI process may be hard to find and assemble. Obviously, they must have a systems perspective and the authority or support of authority necessary to enact their decisions. As a minimum, it is expected that key program managers and potential system team managers will be present on this committee (keeping in mind that it must be kept to a reasonable size).

Processes resembling aspects of the six step DFI framework are developing in several programs I have visited and are suspected in many others that also appreciate integration issues and their potential to influence program success. It is hoped that further elaboration and refinement will provide a DFI template suitable for a wide range of

programs, yet powerful enough to provide explicit directions towards the anticipation and resolution of integration issues.

Companies can pursue launching and managing complex programs as a core competency or organizational capability. In the development of large, complex systems, a systematic approach is required to ensure an effective product and process. Companies that learn and begin improving upon a systematic process for process development will be a step ahead of most competitors.

## ACKNOWLEDGMENT

Direct and indirect funding for this work has been provided through the Massachusetts Institute of Technology's Lean Aircraft Initiative, sponsored jointly by the US Air Force and a consortium of aerospace companies. Additional support has come from MIT and the National Science Foundation. I am grateful for helpful comments provided by Stanley Weiss, Steven Eppinger, and Charlie Boppe.

## REFERENCES

- Alexander, C., 1964, *Notes on the Synthesis of Form* Cambridge, MA: Harvard University Press.
- Altus, S. S., Kroo, I. M. and Gage, P. J., 1995, "A Genetic Algorithm for Scheduling and Decomposition of Multidisciplinary Design Problems" *Proceedings of the 21st ASME Design Automation Conference*, Boston, Sept.
- Blanchard, B. S. and Fabrycky, W. J., 1990, *Systems Engineering and Analysis*. Second Edition. Englewood Cliffs, NJ: Prentice Hall.
- Browning, T. R., 1996a, *Systematic IPT Integration in Lean Development Programs* Master's Thesis (Aero./TPP), M.I.T., Cambridge, MA.
- Browning, T. R., 1996b, "A Systems Engineering Approach to Multi-Team Integration: Interdependence and Integrative Mechanisms" *Proceedings of the Sixth Annual International Symposium of INCOSE*, St. Louis, July.
- Browning, T. R., 1997, "Mechanisms for Interteam Integration: Findings from Five Case Studies" *Proceedings of the Seventh Annual International Symposium of INCOSE*, Los Angeles, Aug. 3-7.
- Cole, W. E., 1995, "Cooking Up a Batch of Team Synergy: Ingredients for Setting Up Successful Teams" *Program Manager* No. Sept.-Oct., pp. 28-33.
- DoD, 1994, *MIL-STD-499B Systems Engineering (Draft)* DoD, Military Standard Specifications MIL-STD-499B.
- Eppinger, S. D., Whitney, D. E., Smith, R. P. and Gebala, D. A., 1994, "A Model-Based Method for Organizing Tasks in Product Development" *Research in Engineering Design* Vol. 6, , pp. 1-13.
- Grady, J. O., 1994, *System Integration* Boca Raton, FL: CRC Press.
- Gulati, R. K. and Eppinger, S. D., 1996, "The Coupling of Product Architecture and Organizational Structure Decisions" M.I.T. International Center for Research on the Management of Technology, Working Paper no.151.
- Katzenbach, J. R. and Smith, D. K., 1993, *The Wisdom of Teams: Creating the High-Performance Organization* Boston: Harvard Business School Press.
- Klein, J. A. and Susman, G. I., 1995, "Lean Aircraft Initiative Organization & Human Resources (O&HR) Survey Feedback-Integrated Product Teams (IPTs)" M.I.T. Lean Aircraft Initiative, White Paper LEAN 95-03.
- Kockler, F. R. et al., 1990, *Systems Engineering Management Guide* Defense Systems Management College.
- Kusiak, A. and Wang, J., 1993, "Decomposition of the Design Process" *Journal of Mechanical Design* Vol. 115, No. December, pp. 687-695.
- McCord, K. R. and Eppinger, S. D., 1993, "Managing the Integration Problem in Concurrent Engineering" M.I.T. Sloan School of Management, Working Paper no.3594.
- Michelena, N. F. and Papalambros, P. Y., 1995, "Optimal Model-based Decomposition of Powertrain System Design" *Proceedings of the ASME Advances in Design Automation*.
- Mohrman, S. A., Cohen, S. G., and Mohrman, Jr., A. M., 1995, *Designing Team-Based Organizations* San Francisco: Jossey-Bass.
- Morelli, M. D., Eppinger, S. D. and Gulati, R. K., 1995, "Predicting Technical Communication in Product Development Organizations" *IEEE Transactions on Engineering Management* Vol. 42, No. 3, pp. 215-222.
- Peters, J. F., 1995, "The Transition of Functional Organizations to Integrated Product Teams on the Space Station Program" *Proceedings of the Fifth Annual International Symposium of INCOSE*, St. Louis, July 22-26.
- Pimmler, T. U. and Eppinger, S. D., 1994, "Integration Analysis of Product Decompositions" *Proceedings of the ASME Design Theory and Methodology Conference*, Minneapolis, MN, Sept.
- Rechtin, E., 1991, *Systems Architecting: Creating & Building Complex Systems* Englewood Cliffs, NJ: P T R Prentice Hall.
- Sheard, S. A. and Margolis, M. E., 1995, "Team Structures for Systems Engineering in an IPT Environment" *Proceedings of the Fifth Annual International Symposium of INCOSE*, St. Louis, July 22-26.
- Shishko, R. et al., 1995, *NASA Systems Engineering Handbook* NASA.
- Susman, G. I., Ed. 1992, *Integrating Design and Manufacturing for Competitive Advantage*. New York: Oxford University Press.
- von Hippel, E., 1990, "Task Partitioning: An Innovation Process Variable" *Research Policy* Vol. 19, , pp. 407-418.
- Whitney, D. E., 1990, "Designing the Design Process" *Research in Engineering Design* Vol. 2, , pp. 3-13.