INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

ALIGNING PROCESS, PRODUCT, AND ORGANIZATIONAL ARCHITECTURES IN SOFTWARE DEVELOPMENT

Manuel E. Sosa
Assistant Professor of Technology and Operations Management. INSEAD, France.

ABSTRACT

One of the most difficult challenges when managing product development is to identify the individuals or design teams that need to coordinate closely their interdependencies during the design process. To address this challenge, the literature in product development has studied the task structure of product development processes, the architecture of complex products, and the communication patterns of development organizations. Yet, we still lack an integrated view of how process, product, and organizational structures align themselves when developing new products. In this paper, I introduce the notion of the *affiliation matrix* to map the product architecture onto the organizational structure and estimate the potential organizational communication patterns which, in turn, drive the design iterations identified in the development process. This approach has implications for process improvement, product and organization design, and project management. I illustrate the implementation of this approach in the context of a software development project.

Keywords: product architecture, software development, organization design, project management.

INTRODUCTION

One of the most important challenges in product development is to manage design iterations and change propagations [1][2]. Ultimately, this can be done effectively if engineering managers are able to identify the individual actors involved in such iterations and the product interfaces such actors need to attend to. The literature in product development has studied the task structure of product development processes, the architecture of complex products, and the communication patterns of development organizations [1][3][4]. Recent simulation-based studies have investigated the interplay of organizational and problem structure dynamics in complex engineering projects [5][6]. Yet, we still lack an integrated view of how process, product, and organizational structures align themselves when developing new products. Previous work shows that identifying and attending the interfaces between product components that require special attention to coordinate during the development process is a challenging task, even when the product architecture maps directly onto the organizational structure [3][4]. The managerial challenge becomes even harder when such mapping is not direct, which occurs when the design of product components is assigned to various teams or individuals within the development organization [7]. This is common in software development projects in which many individual actors typically contribute to the design and integration of software components in a flexible development process [8]. To address this challenge, I develop a structured approach to integrate process and organizational structures using the architecture of the product under development. I introduce the notion of the *affiliation matrix* to map the product architecture onto the organizational structure and estimate potential organizational communication patterns. By documenting both the architecture of the product and the contribution of development actors to the design of each product component, one can identify potential technical interactions that would need to take place to coordinate the interfaces between product components. Although the approach presented in this paper is general, and therefore applicable to any product development effort, I illustrate its implementation in a software development firm, which offers additional insights particularly relevant for software development.

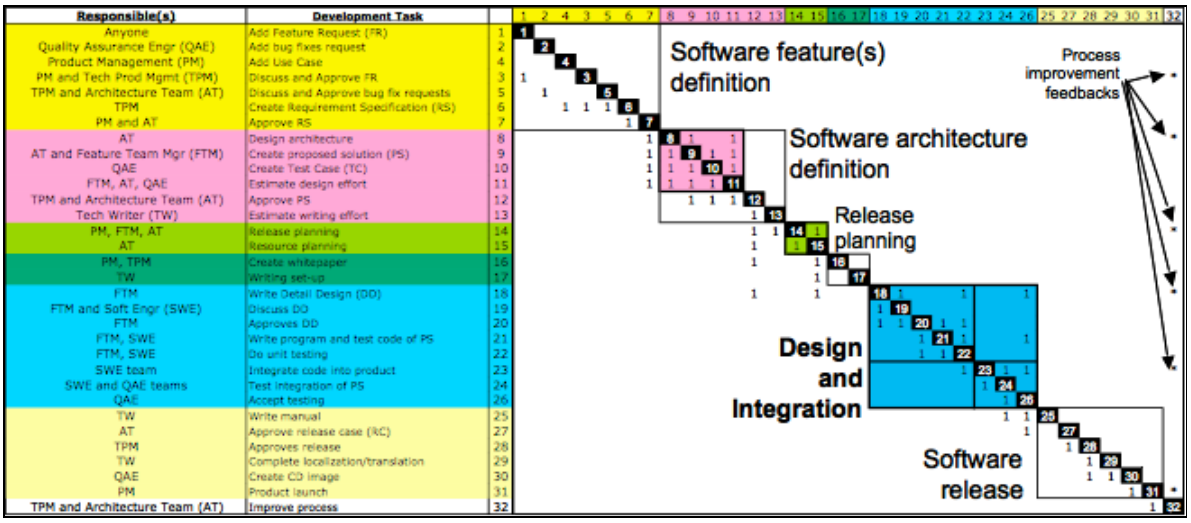
Previous work has studied the mapping of various dimensions of product development systems. Morelli *et al.* [7] map the process and organizational structure to predict task related interactions. They found that task interdependency is a better predictor of technical communication than distance-based models [9]. Sosa *et al.* [4] have studied the direct mapping of product and organizational structures in complex products. They found that several organizational and product-related factors significantly influence the misalignment of design interfaces and team interactions. Eppinger and Salminen [10] proposed a framework to study complexity in product development by examining the overlap between product, process, and organizational structures. This paper contributes to this stream of literature in three ways. First, this paper shows that to effectively manage intended (or planned) design iterations, managers need to instantiate their development process with a specific product architecture which allows them to predict the organizational communication patterns that need managerial attention. Second, I operationalize a general structured approach to align product and organizational structures with the use of the *affiliation matrix*. Third, because of the predictive power of the approach presented in this paper, I present a project management framework to predict the actors and their organizational interactions that may need special attention when developing a *subset* of product components, which is the case in flexible software development.

RESEARCH MOTIVATION AND FRAMEWORK

Improving product development efforts typically starts by documenting design tasks and their information requirements [1][2][11]. By examining the task structure of the process, managers can uncover the interdependent activities that are more likely to generate design iterations. The design structure matrix (DSM) is a matrix-based analytical tool introduced by Steward [12] and used by Eppinger and his colleagues to represent and organize design tasks in complex product development projects [1][13]. In the product domain, a matrix representation has also been used to represent products as networks of interconnected components [14][15][16]. Finally, in the organizational domain, development organizations have been considered as social networks of interacting actors that integrate their efforts to develop new products and services [4][6][7][9]. Therefore, product development systems can be considered as a network of design tasks (process architecture) put in place by a social network of developers (organizational architecture) to develop products composed of interdependent components (product architecture). These three dimensions significantly influence one another, and understanding their relationships is crucial to improve product development systems [4][10]. Moreover, I argue that to manage design iterations effectively, one must examine how interdependent design tasks and interdependent product components ultimately determine the communication patterns of the organization. Next, I examine the process structure in the software organization I studied and illustrate the need to instantiate such a development process with the architecture of a particular product under development which, in turn, determines the potential communication patterns of the organization during the completion of the iterative development tasks.

Examining the process architecture

The task structure of the development process used by the software firm I studied is represented in the design structure matrix shown in Figure 1. This DSM representation captures their development process internally documented in a multi-page process flow diagram. The matrix shown in Figure 1 is a square matrix whose rows and columns are identically labelled with the development tasks, and an off-diagonal mark, (*i,j*), indicates that to complete task *i* (labelling row *i*) needs information from task in column *j*. The blocks along the diagonal of such a DSM highlight the groups of tasks that are executed together (in parallel, sequentially, and/or iteratively) within each phase. As evident from Figure 1, an important contribution of a DSM representation is the simple and explicit depiction of complex and iterative processes where sets of iterative activities (i.e., design iterations) can be highlighted. Figure 1 shows three sets of highly interdependent tasks: 1) Software architecture definition; 2) Software release planning; and 3) designing and integrating software features. Note that Figure 1 does not show *unintended* interdependencies that occur across phases (e.g. from the "design and integration" phase to the "define software architecture" phase). Instead, this DSM only shows the intended iterations that are planned to occur.

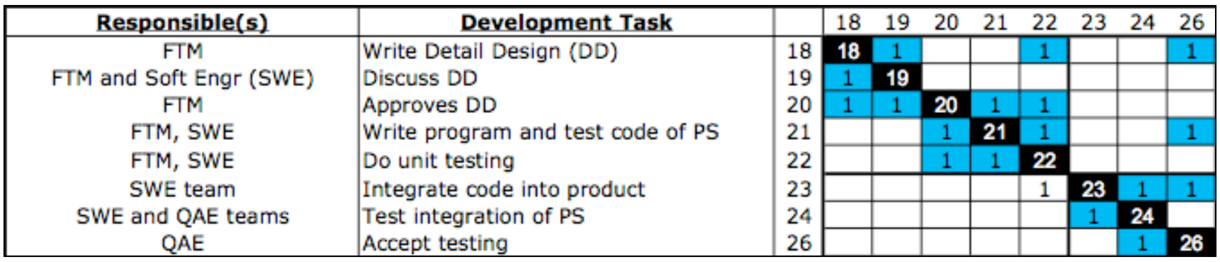


Responsible(s)	Development Task	
Anyone	Add Feature Request (FR)	1
Quality Assurance Engr (QAE)	Add bug fixes request	2
Product Management (PM)	Add Use Case	4
PM and Tech Prod Mgmt (TPM)	Discuss and Approve FR	3
TPM and Architecture Team (AT)	Discuss and Approve bug fix requests	5
TPM	Create Requirement Specification (RS)	6
PM and AT	Approve RS	7
AT	Design architecture	8
AT and Feature Team Mgr (FTM)	Create proposed solution (PS)	9
QAE	Create Test Case (TC)	10
FTM, AT, QAE	Estimate design effort	11
TPM and Architecture Team (AT)	Approve PS	12
Tech Writer (TW)	Estimate writing effort	13
PM, FTM, AT	Release planning	14
AT	Resource planning	15
PM, TPM	Create whitepaper	16
TW	Writing set-up	17
FTM	Write Detail Design (DD)	18
FTM and Soft Engr (SWE)	Discuss DD	19
FTM	Approves DD	20
FTM, SWE	Write program and test code of PS	21
FTM, SWE	Do unit testing	22
SWE team	Integrate code into product	23
SWE and QAE teams	Test integration of PS	24
QAE	Accept testing	26
TW	Write manual	25
AT	Approve release case (RC)	27
TPM	Approves release	28
TW	Complete localization/translation	29
QAE	Create CD image	30
PM	Product launch	31
TPM and Architecture Team (AT)	Improve process	32

Software feature(s) definition
Software architecture definition
Release planning
Design and Integration
Software release
Process improvement feedbacks

Figure 1. Software development process at the firm studied

Figure 2 shows a closer look at the most iterative set of development tasks in the process documented in Figure 1. Even though managers could differentiate interdependencies by highlighting which task interfaces involve people from the *same* group, as opposed to people from *different* groups, the managerial challenge remains: they need to predict "who should talk to whom?" and "what should they talk about?" in order to facilitate an efficient completion of this group of *planned*, highly iterative activities. This is particularly challenging in software development because of the additive way in which software products are developed. In other words, in software development, product features are developed, integrated, and tested on the main product in an additive manner while they get external feedback about the evolving product [8]. Hence, for managers being able to facilitate the completion of the iterative set of activities shown in Figure 2, they need to understand how the software components that instantiate these design tasks link to one another (i.e., the product architecture) as well as the people responsible for contributing to the design of such software components (i.e., the affiliation of people to components' design).



Responsible(s)	Development Task		18	19	20	21	22	23	24	26
FTM	Write Detail Design (DD)	18	18	1			1			1
FTM and Soft Engr (SWE)	Discuss DD	19	1	19						
FTM	Approves DD	20	1	1	20	1	1			
FTM, SWE	Write program and test code of PS	21			1	21	1			1
FTM, SWE	Do unit testing	22			1	1	22			
SWE team	Integrate code into product	23					1	23	1	1
SWE and QAE teams	Test integration of PS	24						1	24	
QAE	Accept testing	26							1	26

Figure 2. "Design and Integration" iterative activities of the software development process studied

As shown in Figures 1, having an aggregated view of how information flows among development tasks helps identify the tasks involved in design iterations, both *within* and *across* phases of the development process [1]. Yet, managing design iterations poses different challenges depending on whether they occur within or across phases. Managing interfaces *across* different phases of the development process can be facilitated by a process view alone (as the ones shown in Figure 1), and typically require preventive actions from managers to avoid major rework due to going back to previous phases of the development process [13]. Managing interfaces *within* the same phase, as the one shown in Figure 2, poses a significantly different managerial challenge because these are design iterations that are (or should be) planned and facilitated by managers. Yet, these development tasks typically refer to specific components of the product under development. As a result, the organizational interdependencies that generate such planned (within the phase) design iterations are determined not only by the architecture of the product itself but also by the contribution of people to the design of such product components. In some cases, this challenge can be facilitated by a direct mapping from the process structure to the product and organizational structures. For example, in a previous study, Sosa *et al.* [4] shows how the detailed design tasks carried out to develop a commercial aircraft engine mapped directly onto the architecture of the engine (i.e., design task x was defined as: "*complete the detailed design of engine component x"*), and how the engine architecture mapped directly onto the organization responsible for designing the 54 main components of the engine. However, in many other cases, such as software development projects, the mapping is far from being one-to-one. As a result, we need to provide a general approach to map product architectures to organizational structures to effectively manage planned design iterations. Next, I describe a research approach to address this challenge.

RESEARCH APPROACH

In order to help managers to handle the planned design iterations that typically occur in the design phase of product development processes, I suggest a structured approach that maps the architecture of the product that instantiate the development process to the organization that designs it. Note that this approach generalizes the research approach introduced by Sosa *et al.* [3][4] in which the product and organizational structures map one-to-one.[1] As seen below, I introduce the use of the *affiliation matrix* to align product and organizational structures that do not map directly. I structure the research approach in five steps (see Figure 3):

1. Capture the product architecture. By interviewing systems architects, I identify the n components that form the product and the interfaces among them. Then, I document the product data into a *product architecture matrix* (***P***). $P_{n,n}$ is a square matrix whose rows and columns are identically labeled with the n components of the product. A non-zero, off-diagonal cell, p_{ij}, in this matrix indicates that component i imposes design constraints to component j. Such convention is consistent with previous work in software architecture [17].
2. Capture actual organizational structure. By surveying the m development actors involved in the development of the product, I document their actual product-related interactions (or the actual intentions to interact) onto a square (person to person) *organizational communication matrix* ($C_{m,m}$). To be consistent with the convention used in step 1, the rows of the matrix are labeled with the "providers" of product-related information while the columns are labeled with the "recipients" of information. Hence, cell c_{ij} indicates that actor j reports actual interactions with actor i (i.e., actor j "goes to" actor i to request product-related information).
3. Capture task assignment. I document the task assignment of the organization by asking the m development actors about their level of involvement in the design of each of the n product components. I document this information in an *affiliation matrix* (***A***). $A_{m,n}$ is a rectangular matrix whose rows are labeled with the m development actors and its columns are labeled the n product components. Hence, cell a_{ij} indicates the degree of involvement of actor i in the design of component j.
4. Determine potential interactions. I introduce the following model to determine the potential interactions among development actors, given the product architecture matrix and the affiliation matrix. I define $T_{m,m}$ as the *potential interaction matrix* whose non-zero, off-diagonal cell, t_{ij}, indicates that development actor i provides information to development actor j because they are involved in product components that share design interfaces. Hence,

   $$T_{m,m} = A_{m,n} \cdot P_{n,n} \cdot A^T_{n,m} \tag{1}$$

   Note that if ***P*** and ***A*** are binary matrices, then t_{ij} captures the number of components designed by actor j that depend on components designed by actor i. Since the affiliation matrix captures various levels of task involvement, I determine potential interactions for two extreme cases: 1) strongest involvement only, and 2) all levels of involvement. (For both cases, I assume ***P*** to be a binary matrix). The results of this model are captured in *potential interaction matrices*, which correspond to the two levels of design involvement.
5. Compare potential and actual interactions. By overlaying the *potential interaction matrices* and the actual *organizational communication matrix*, I determine *mismatches* of actual and potential interactions associated with the development of such a product. Since I consider two potential

[1] A one-to-one mapping of product and organizational structures is characterized by the mutually exclusive assignment of the design of each component of the product to one individual actor in the organization.

interaction matrices associated with extreme levels of design involvement, I obtain two *preliminary comparison matrices*. First, by considering the strongest-only level of involvement of people in their component design, I determine the potential interactions that are *unattended* by actual interactions. This result comes about because by considering strong-only design involvement, we are effectively predicting the set of most likely potential interactions among the people who are strongly involved in the design of product components. Next, by considering all levels of task involvement, I determine the actual interactions that are *unpredicted* by potential interactions. With this second approach, one can predict potential interactions that have the lowest probability of occurring due to a low level of involvement in design tasks. Hence, if an actual interaction falls outside such a wide set of potential interactions, it is considered a truly unpredicted interaction. I document the results of these two preliminary comparison matrices in the *final comparison matrix*.

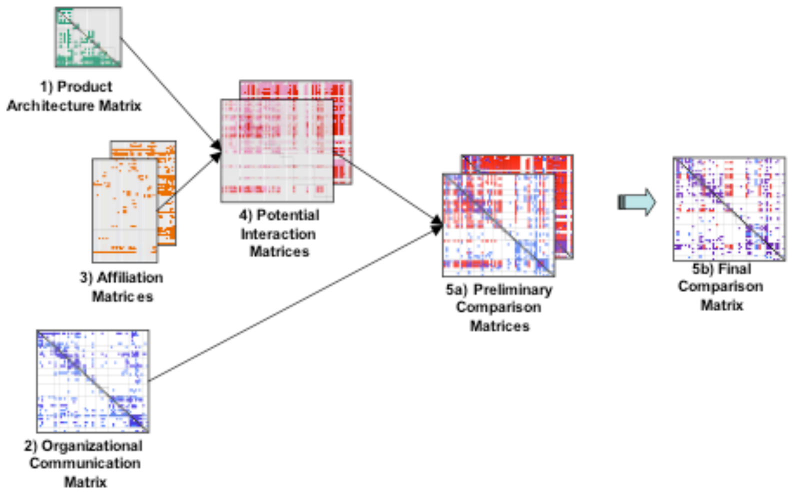

1) Product Architecture Matrix
3) Affiliation Matrices
4) Potential Interaction Matrices
5a) Preliminary Comparison Matrices
5b) Final Comparison Matrix
2) Organizational Communication Matrix

Figure 3. Five-step Research Approach

A SOFTWARE DEVELOPMENT EXAMPLE

I implemented my research approach in a software development firm. The firm is a mature, public, European company and one of the leaders in the market for a specific type of application for business customers. By the time of the data collection, the firm was allocating over 60% of their development resources to the development of one radically new product whose development effort had started within the previous 12 months. The product comprised 34 interdependent modules and the development organization was formed by 66 people, most of whom contributed to the conception and implementation of the 34 modules of the product. Two important factors facilitated the selection of the project studied. First, the firm was interested in examining their process, product, and organizational structures to accelerate the development of the product studied. Second, the architecture of the product studied and the development organizational structure did not map directly to each other. This provided an ideal opportunity to implement the five-step approach outlined above.

Step 1: Capturing the software architecture. After a long concept development phase, in which the firm assessed their market needs and technological opportunities, they established the architecture of the product to be analysed in this study. The product comprised 34 *modules*, whose detailed design would address all the functional requirements of the product. System architects had also identified how each of these modules would depend on the others. With this information, I built a 34x34 product architecture binary matrix, whose off-diagonal marks, (i,j), indicate that to design the module in column j, designers "need to know about" the module in row i. Such a convention facilitates the mapping of a matrix representation to a block diagram representation commonly used in software development. Note that because of the highly asymmetric nature of interdependences in software products, I used a partitioning algorithm (instead of a clustering algorithm) to identify the highly interdependent modules in the product [11][17]. In sum, I built a partitioned product architecture matrix to capture the dependency structure of the 34 modules that formed the software product studied. As shown in Figure 4, the 34 modules of the product are organized into 6 groups of components. We identified 250 critical design interfaces among the 34 modules. Although all the interfaces needed careful attention to ensure that the modules integrated well and the entire software application fulfilled its functional requirements, some interfaces would pose significant managerial challenges due to the iterative constraints they would impose on some of the components. Such interfaces are highlighted in "blue" in Figure 4. Figure 4 shows both a matrix representation and a block diagram of the product studied.

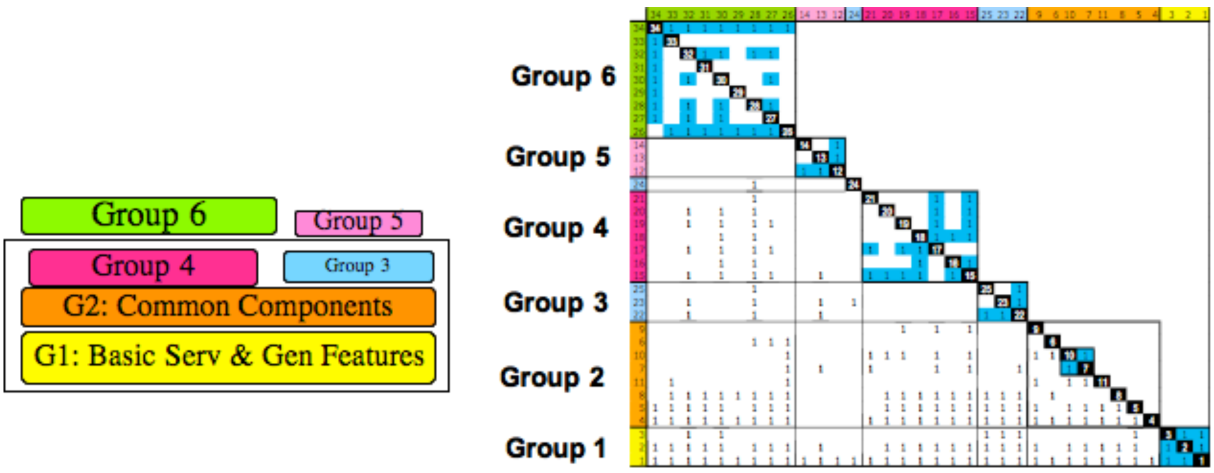


Figure 4. Software Product Architecture

Step 2: Capturing the formal and informal development organizational structure. The development organization studied was formed by 66 people organized onto 11 groups distributed in three different sites in Europe. Eight groups were dedicated to software development (i.e., programming), six of which were responsible for the design of the 34 modules of the product studied. The other three groups provided support to the rest of the organization in areas such as quality assurance, system architecture design, and technical documentation (see Figure 5). I distributed a comprehensive web-survey among all the 66 people involved in the development organization to capture their product-related interactions. I documented these data into an organizational communication matrix whose off-diagonal marks (*i,j*) indicate how often the person in column *j* went (or intended to go) to person in row *i* to request product-related information during the last year. Note that I sequence this matrix to capture the structure of the organization into its 11 functional groups; hence, I cluster together people who belong to the same organizational group. I got 59 complete communication surveys for an overall response rate of 89%. Moreover, the response rate among the developers and testers was over 95%. Figure 5 shows the actual technical communication patterns associated with the development of the product studied in a 59x59 organizational communication matrix. Respondents reported 511 product-related interactions in which actor *j* "went to (or intended to go to)" actor *i* for product-related information. This results in a communication network density of 15%.

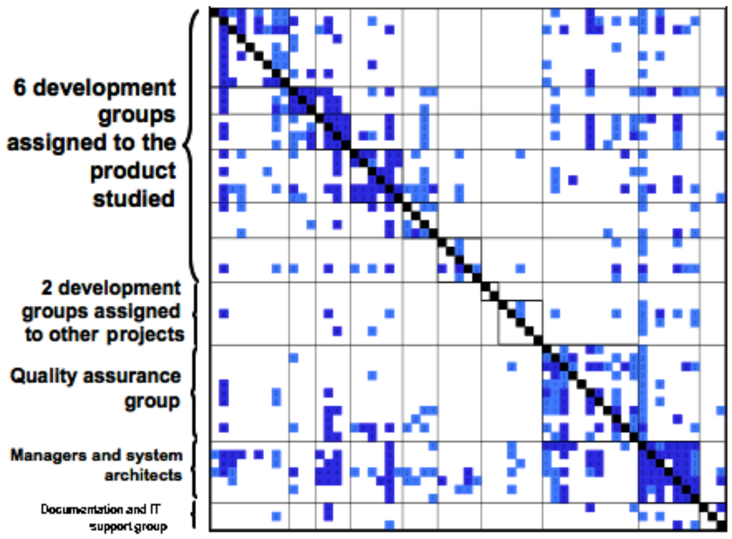


Figure 5. Actual Organizational Interaction Matrix

Step 3: Capturing the involvement of people in the design of product modules. As part of the web-survey, I asked respondents to rate their level of involvement in the conception and implementation of each of the 34 product modules. The six-point scale used to capture their level of involvement included the following values: "Not involved", "Barely involved", "Somewhat involved", "Involved", "Very involved", and "Strongly involved". I documented this data in a valued *affiliation matrix* [18]. The rows of the affiliation matrix are labelled identically to the rows in the organizational interaction matrix (step 2), while its columns are labelled identically to the columns in the product architecture matrix (step 1). Hence, cell (*i,j*) in this matrix indicates the level of involvement of the person in row *i* in the conception and implementation of the software module in column *j*. Finally, I built binary affiliation matrices for the following two cases: 1) Design involvement rated as "strongly involved" only; 2) design involvement rated at least as "Barely involved". Figure 6 shows the two binary affiliation matrices for these two cases, respectively.

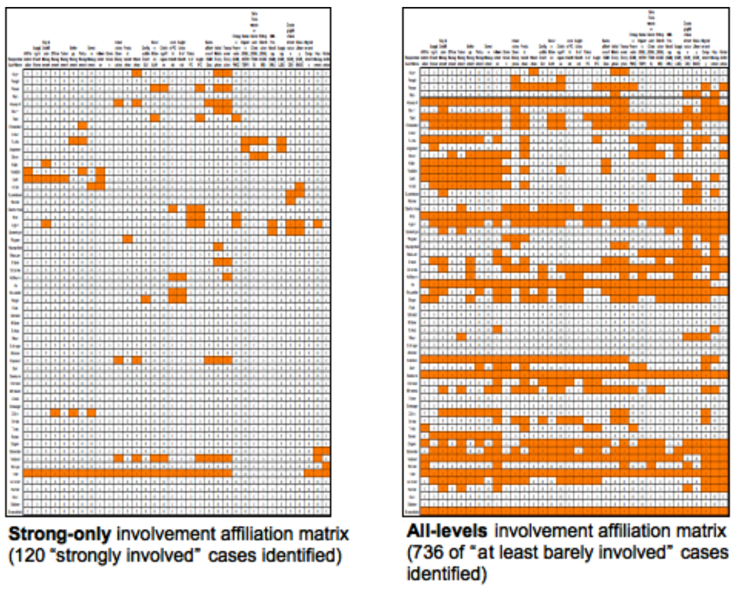


Figure 6. Potential affiliation matrices

Step 4: Determining potential organizational interactions. By combining the product architecture matrix and the affiliation matrix in the algebraic model introduced in Equation 1, one can determine the total number of interfaces between product modules that any pair of developers need to potentially coordinate on. I document the results of this model in two *potential interaction matrices*, whose rows and columns are labelled identically to the rows and columns in the organizational interaction matrix captured in step 2. These matrices correspond to the two levels of task involvement reported in the affiliation matrices documented in step 3. Hence, for the case of *strong-only* design involvement, the potential interaction matrix captures 594 potential interactions. Such a matrix has a density of 17%. For the case of *all-levels* design involvement, the potential interaction matrix shows 2,306 potential interactions, which results in a communication network density of 67% (see Figure 7).

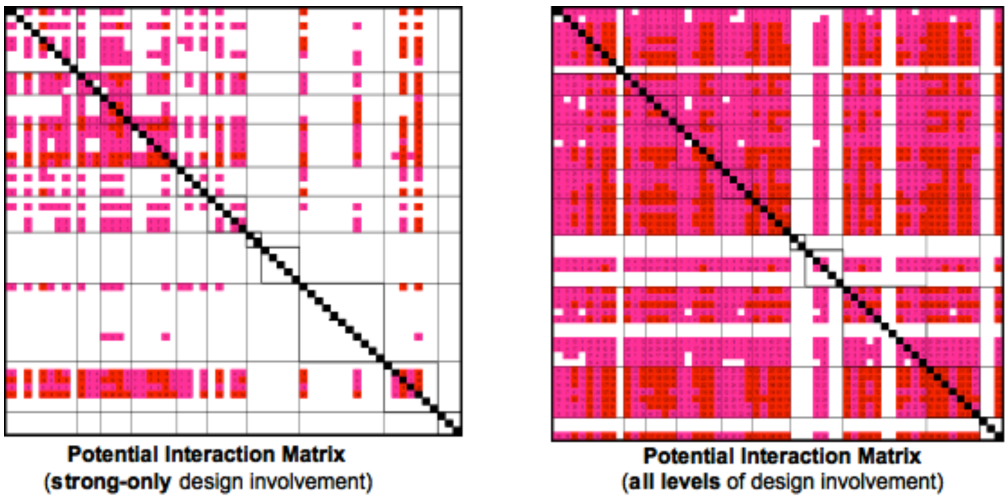


Figure 7. Potential Communication Matrices

Step 5: Comparing actual and potential interactions. I carried out this step by completing two separate comparisons. I do this because I use the two potential interaction matrices determined in step 4 to compare them with the actual organizational communication matrix documented in step 2. The first comparison is focused on identifying the *potential unattended interactions*, whereas the second comparison is used to uncover *unpredicted organizational interactions*. *Potential unattended interactions* are those that correspond to pairs of developers who are expected to interact because the components they design are interdependent, and no one else in the organization addresses such interdependences. On the other hand, *unpredicted interactions* are those which take place between development actors, even though they are not involved in the design of components that share interfaces with the other actors' components.

- First, to identify the potential *unattended* interactions, I compare the actual interaction matrix with the "strong-only" potential interaction matrix. In order to obtain the final set of *truly* potential unattended interactions, I filter out potentially "redundant interactions" associated with each product interface identified. I define "redundant interactions" as those associated with product interfaces that were matched by actual interactions between other potential "providers" and "receivers". Hence, to remove redundant interactions from the initial count of unattended interactions, one must identify the product interfaces whose potential interactions are completely unmatched by actual interactions (i.e., interfaces whose "providers" and "receivers" do not report actual interactions). This can be done systematically by examining, one by one, whether the potential interactions associated with each interface are matched by actual interactions. Those product interfaces, whose totality of potential interactions are not matched by actual interactions, are defined as *unmatched product interfaces,* and those potential (unmatched) interactions are the *truly potential unattended interactions*. Originally, I identified 367 potential unattended interactions which were not matched by actual interactions. Yet, after removing redundant interactions, only 116 *truly potential unattended interactions* were identified. Note that these truly unattended potential interactions corresponded to 36 unmatched product interfaces. Again, from the original set of potential unattended interactions, 251 of them were not considered truly potential unattended interactions because other pairs of developers, also involved in those product interfaces, reported actual interactions which could be associated with such product interfaces.
- Second, to uncover *unpredicted interactions*, I compare actual organizational interactions with potential interactions for the case of "at least barely involved" design involvement. In this case, the potential interaction matrix shows a high communication density because any two people

who are “at least barely involved” in the design of any of the 34 modules, would need to interact with other actors if their components share interfaces. Even after controlling for such a possibility, I still found 72 *truly unpredicted interactions* (i.e., 14% of the actual interactions were unpredicted). These interactions took place between people who interacted (or planned to interact), even though all the components they designed (or contributed to) did not share technical interfaces.

The aggregated results of the two comparisons are documented in a *final comparison matrix* shown in Figure 8. The red cells indicate *truly unattended potential interactions*, the blue cells show the *truly unpredicted interactions,* and the purple cells mark *matched interactions*.

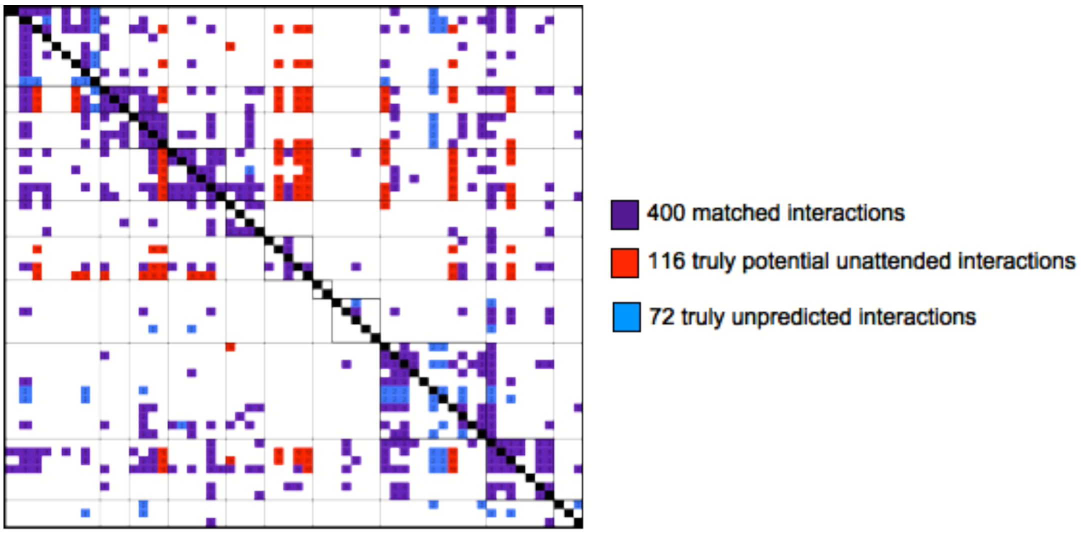


Figure 8. Final Comparison Matrix

ANALYSIS AND DISCUSSION OF RESULTS

An important benefit of implementing the structured approach described in this paper is that it provides a systematic way to identify *mismatches* between product and organizational architectures for cases where their structures do not map one-to-one. Identifying these mismatches in a systematic way can help managers steer their attention to areas, both within the product and the organization, that may require special managerial action. Two types of mismatches between potential and actual interactions can occur. First, *truly potential unattended interactions* take place between development actors who have not coordinated the interfaces (or do not intend to do so) of some of the software modules whose design they strongly contribute to. Second, *truly unpredicted interactions* occur between development actors who interact even though they are not involved in the development of interdependent modules. More specifically, I found that 32% of the 367 potential unattended interactions identified were truly unattended, while only 14% of the 511 actual product-related interactions were truly unpredicted. Note that in order to determine truly potential unattended interactions, I also identify the product interfaces that are not associated with actual organizational interactions. In this case, over 14% of the 250 product interfaces identified by system architects had not been matched by actual interactions of people strongly involved in the design of such interdependent software modules.

Factors associated with unattended and unpredicted interactions

Analyzing the final comparison matrix further allows us to test whether unattended and unpredicted interactions are concentrated in few actors or are sparsely distributed throughout the development network. In this case, I found that truly potential unattended interactions were significantly concentrated in a small group of actors. In other words, 90% of the 116 truly unattended interactions were associated with nine actors who reported significantly fewer product-related interactions with others. That is good news for managers because they can focus their attention on a small set of actors to minimize the risk of overlooking critical product interfaces. Yet, an overwhelming 98% of the unattended interactions occurred across group boundaries, which confirms the importance of carefully identifying and managing cross-boundary interfaces. Moreover, 87% of the truly potential unattended interactions occurred between, or with, development actors of one of the six development groups

responsible for the design of the product studied, which suggests that such unattended interactions could have a detrimental impact on product performance (if they remained unattended). An important benefit of identifying truly potential unattended interactions is that to do so one must identify the product interfaces that are not matched by actual interactions. In this case, I found that 36 product interfaces (out of 250) were not attended by actual interactions. This provides clear guidance for managers on the aspects of the product architecture that the development organization is likely to "miss" if no special attention is paid to them.
As for the people involved in truly unpredicted interactions, it is important to highlight that a significantly large proportion of them (65%) were interactions initiated by development actors outside the development groups. This suggests that using the architecture of the product to predict technical interactions with groups of the organization that perform development tasks different from "design tasks" has certain limitations.

MANAGERIAL AND ACADEMIC IMPLICATIONS

This research has important implications for both managers and academics. Research in engineering design has suggested that identifying design iterations is essential to manage them effectively. There are two types of design iterations: 1) intended, or planned, design iterations that typically occur within the same product development phase; and 2) unintended, or unplanned, design iterations which typically occur across product development phases. In this paper, I focus on managing planned design iterations within the same product development phase. This is particularly relevant in software development in which design and integration activities take place concurrently as the products are built. To manage these types of design iterations, it is essential for managers to identify the set of actors that need to interact and the interfaces they need to interact about. This paper presents a general and structured approach to tackling this challenge. Moreover, the systematic implementation of this approach to small "portions" of the product can help managers to manage design iterations at a more granular level because they can identify systematically the potential interactions that need to take place to address a subset of product interfaces. Because "potential interactions" represent the set of interactions that could potentially coordinate a set of product interfaces, managers must select and facilitate the subset of those potential interactions that would address such a subset of interfaces effectively. Figure 9 illustrates how, by bringing together the process, product, and organizational views, managers can effectively tackle the management of iterative design tasks for a subset of software modules. Such a project management framework is particularly relevant in software development, where products are developed in a flexible and additive fashion.

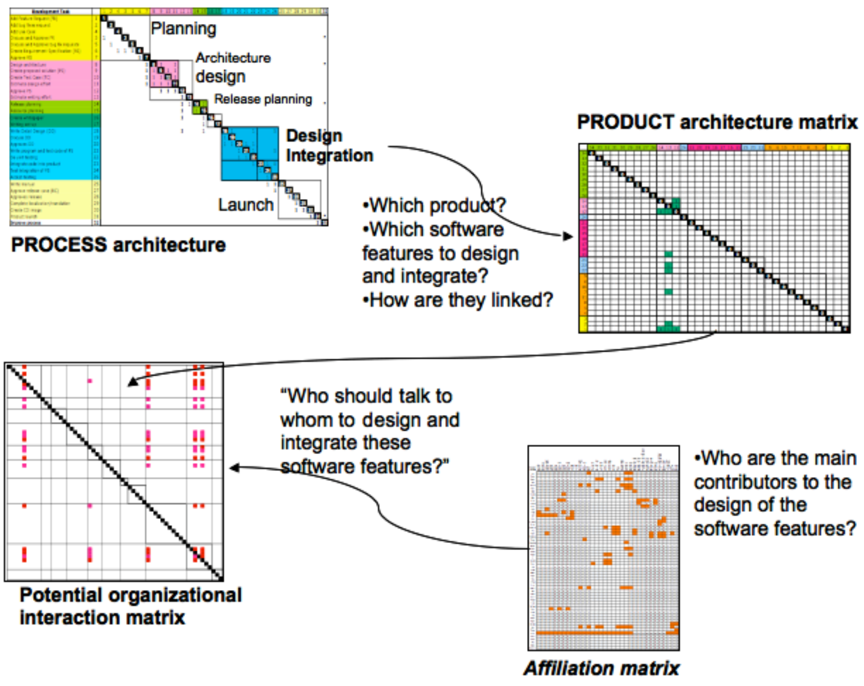

Planning
Architecture design
Release planning
Design Integration
Launch
PROCESS architecture
PRODUCT architecture matrix
•Which product?
•Which software features to design and integrate?
•How are they linked?
"Who should talk to whom to design and integrate these software features?"
•Who are the main contributors to the design of the software features?
Potential organizational interaction matrix
Affiliation matrix

Figure 9. Aligning process, product, and organizational views

Figure 9 starts with the process architecture in which the main phases of the process are highlighted, including the “design integration” phase in which most of the planned, yet longer, iterations take place. Then, to effectively manage those iterations, managers capture the architecture of the specific product whose software components are to be (re)designed and integrated. The architecture view can show the entire product architecture or focus on a subset of them. Figure 9 shows the interfaces associated with one group of software features only. Then, by combining the affiliation matrix and the product architecture matrix, managers can predict the set of potential interactions that would need to take place between people who strongly contribute to the design of the software features included in the product architecture matrix. The potential communication matrix provides managers with the set of people and organizational interactions to choose from to effectively coordinate the product interfaces in question. Note that for managers being able to choose the appropriate set of organizational interactions to facilitate, they must distinguish which potential interactions are “non-redundant. Those are the interactions that must take place because the development actors involved are the only pair of actors who are strongly involved in the design of a particular pair of interdependent components. For the example shown in Figure 9, only two potential interactions were “non-redundant”.

More generally, the approach presented in this paper allows managers to identify mismatches between actual and potential interactions. Once mismatches are identified, managerial actions can take place in the product and process domain by updating the product and planning information when new interfaces are uncovered, or in the organization domain by reorganizing development actors to attend interfaces that may have otherwise been overlooked. From a theoretical viewpoint, the implications of this approach rest on the analytical usage of the potential interaction matrices to make predictions about structural properties of the actors involved in the development of a new product.

REFERENCES

[1] Eppinger, S.D., Whitney, D.E., Smith, R.P., and Gebala, D.A. A Model-Based Method for Organizing Tasks in Product Development, *Res. in Eng'g. Des.* 1994, 6(1), 1-13.

[2] Clarkson, P.J., Simons, C.S., and Eckert, C.M. Predicting Change Propagation in Complex Design,” *ASME Journal of Mechanical Design*, 2004, 126(5), 765-797.

[3] Sosa, M.E., Eppinger, S.D., and Rowles, C.M. Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions, *ASME Journal of Mechanical Design,* 2003, 125(2), 240-252.

[4] Sosa, M.E., Eppinger, S.D., and Rowles, C.M. The misalignment of product architecture and organizational structure in complex product development, *Management Science*, 2004, 50(12), 1674-1689.

[5] Mihm, J., Loch, C., Huchzermeier, A. Problem-solving oscillations in complex engineering projects. *Management Sciente*, 2003, 46(6), 733-750.

[6] Olson, J., Cagan, J., Kotovsky, K. Unlocking organizational potential: A computational platform for investigating structural interdependence in design, *Proceedings of ASME Conference on Design Theory and Methodology,* 2006.

[7] Morelli, M.D., S.D. Eppinger, R.K. Gulati. Predicting technical communication in product development organizations. *IEEE Trans. Eng'g. Management,* 1995, 42(3), 215-222.

[8] MacCormack, A., Verganti, R., and M. Iansiti. Developing products on Internet time: The anatomy of a flexible product development process, *Management Science*, 2001, 47(1),133-150.

[9] Allen, T.J. *Managing the Flow of Technology*. 1977. (Cambridge, Mass.: MIT Press).

[10] Eppinger S.D. and Salminen V.K. Patterns of product development interactions. In *International Conference on Engineering Design, ICED ’01, Vol. 1,* Glasgow, August 2001, pp. 283-290 (Professional Engineering Publishing, Bury St Edmunds).

[11] Browning, T. R. Applying the Design Structure Matrix to System Decomposition and Integration Problems: A review and New Directions, *IEEE Transactions on Engineering Management,* 2001, 48(3), 292-306.

[12] Steward, D. The Design Structure Matrix: A Method for Managing the Design of Complex Systems, *IEEE Transactions on Engineering Management,* 1981, EM-28(3), 71-74.

[13] Eppinger, S.D. Innovation at the speed of information. *Harvard Business Review*, 2001, pp. 149-158.

[14] Pimmler, T.U., S.D. Eppinger. Integration analysis of product decompositions. *Proceedings of*

ASME Conference on Design Theory and Methodology. 1994, pp. 343-351.

[15] MacCormack, A., J. Rusnack, and C. Baldwin. Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code, *Management Science*, 2006, 52(7), 1015-1030.

[16] Sosa, M., Eppinger, S., and Rowles. A network approach to define modularity of components in complex products, ASME Journal of Mechanical Design, 2007 (forthcoming).

[17] Sangal, N., E. Jordan, V. Sinha, D. Jackson. Using Dependency Models to Manage Complex Software Architecture, 2005, *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications*. San Diego, CA, USA.

[18] Wasserman, S. and Faust, K. *Social Network Analysis*, 1994, (Cambridge University Press, New York).

Contact: Manuel E. Sosa
INSEAD
Technology and Operations Management Area
Boulevard de Constance
77305, Fontainebleau
France
Phone: +33 (0)1 60 72 45 36
Fax: +33 (0)1 60 74 61 99
e-mail: manuel.sosa@insead.edu
URL: http://www.insead.edu/facultyresearch/faculty/profiles/msosa/