Res Eng Design (2008) 19:47–70
DOI 10.1007/s00163-007-0039-5

ORIGINAL PAPER

A structured approach to predicting and managing technical interactions in software development

Manuel E. Sosa

Received: 6 July 2007 / Accepted: 22 October 2007 / Published online: 11 January 2008

Abstract One of the most difficult challenges of managing product development is identifying the individuals who need to coordinate closely their interdependencies during the design process. "Who should talk to whom?" and "Which interfaces should they talk about?" are key questions that engineering managers must address when planning and executing product development efforts. In this paper, I introduce the notion of the *affiliation matrix* to map the product architecture onto the organizational structure and predict potential technical communication patterns. By comparing potential interactions with actual communications, engineering managers can uncover product interfaces and organizational interactions that may require special managerial action during the design phase of development processes. This provides an integrated view of how process, product, and organizational structures align themselves when developing new products. I illustrate the implementation of this approach in a software development organization, which offers relevant insights about the challenges associated with managing new software development.

Keywords Software development · Product architecture · Design iterations · Project management

1 Introduction

One of the most important challenges in product development is to manage design iterations and change propagations not only when designing new products but also when redesigning existing ones (Eppinger et al. 1994; Clarkson et al. 2004; Eckert et al. 2004; MacCormack et al. 2001, 2006; Cataldo et al. 2006; Chen et al. 2007). Ultimately, this can be done effectively if engineering managers can identify the individual actors associated with design iterations and the crucial product interfaces involved in them. In simpler terms, managers need to be able to answer two critical questions when planning and executing development efforts: "Who should talk to whom?" and "Which interfaces should they talk about?" To address this challenge, this paper provides a structured and general approach to predicting and managing potential technical interactions in product development organizations.[1]

The basic premise of this paper is that technical organizational interactions take place to coordinate the critical interfaces that connect product components (Henderson and Clark 1990).[2] However, identifying and attending the interfaces between product components that require special attention to coordinate is a challenging task, even when the product architecture maps directly onto the organizational structure (Sosa et al. 2004). (A direct, or 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 or team in the organization.) The managerial challenge becomes even harder when this mapping is not

M. E. Sosa (✉)
Department of Technology and Operations Management,
INSEAD, Boulevard de Constance, 77300 Fontainebleau, France
e-mail: manuel.sosa@insead.edu

[1] I refer to product development in a broad sense to include the development of hardware or software products, or both. However, when referring to software development exclusively, I will make the distinction explicit.

[2] I reserve the use of the word "interfaces" to refer to linkages between product components while "interactions" refer to actual or potential communications between development actors.

direct, as has been observed in product development projects in the electronics industry (Morelli et al. 1995). This is also 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 (MacCormack et al. 2001; Cataldo et al. 2006; Sosa et al. 2007a). To tackle this challenge, this paper suggests a structured way to predict communication patterns based on the architecture of the product and the assignment of design tasks to people in the development organization. The approach introduced in this paper is general not only because it applies to the development of either hardware or software products but also, and more importantly, because it is applicable in cases where the mapping between the product and organizational structures is not one-to-one. More specifically, I introduce the notion of the *affiliation matrix* to capture the involvement of organizational actors in the design of the various components of the product under development ("Who does what?"). With the affiliation matrix, engineering managers can systematically map the product architecture onto the organizational structure and estimate potential technical communication patterns that would need to take place to coordinate critical interfaces between product components.

Improving product development efforts typically starts by documenting design tasks and their information requirements (Eppinger et al. 1994; Browning 2001). By examining the task structure of the process, managers can uncover the interdependent activities that are more likely to generate design iterations. To that end, the design structure matrix (DSM) is a matrix-based analytical tool introduced by Steward (1981) and used by Eppinger and his colleagues to represent and organize design tasks in complex product development projects (Eppinger et al. 1994). In the product domain, a matrix representation has also been used to represent hardware and software products as networks of interconnected components (Pimmler and Eppinger 1994; Sosa et al. 2003, 2007a, b; Sharman and Yassine 2004; MacCormack et al. 2006; Lai and Gershenson 2006). 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 (Allen 1977; Morelli et al. 1995; Sosa et al. 2004; Cataldo et al. 2006; Olson et al. 2006). Therefore, product development systems can be considered as a network of design tasks (process architecture) carried out by a social network of developers (organizational architecture) to develop products comprised of interdependent components (product architecture). These three dimensions influence one another significantly, and understanding the way they interrelate is crucial to improving product development systems (Eppinger and Salminen 2001; Sosa et al. 2004). Moreover, to manage design iterations effectively, it is crucial to understand how interdependent design tasks and interdependent product components ultimately determine the technical communication patterns of the organization, which is what this paper aims to do.

Research in engineering design has also investigated the drivers of design change propagation. This stream of research has analyzed the architecture of complex products to predict how the change in one part of the product may result in changes in other parts (Clarkson et al. 2004; Jarratt et al. 2005). This paper also complements this line of research by emphasizing that to manage change propagation effectively it is necessary not only to understand "the state of the design and the connectivity between the parts of the design" (Eckert et al. 2004, p. 20) but also how design changes could propagate into the organizational structure and impact the technical communication patterns among the development actors involved.

The organizational literature recognizes the challenge faced by organizations when attempting to coordinate the links between the components of the system they develop (Allen 1977; Henderson and Clark 1990; Mihm et al. 2003) and has proposed some strategies to improve the coordination associated with developing interdependent components (Sanchez and Mahoney 1996; Baldwin and Clark 2000; Terwiesch et al. 2002). However, this stream of work provides little specific guidance to predicting technical communication patterns based on the architecture of the systems under development. An important exception to this stream of research is presented by Sosa et al. (2004), which studied the misalignment of the product and organizational structures associated with the development of a large commercial aircraft engine. This paper extends their work by providing a general and structured approach to study product and organizational architectures that do not map directly to each other as observed in software development efforts. Another exception to this stream of work comes from engineering design and it is provided by Cataldo et al. (2006) which suggest an approach (similar to the one presented here) to computing coordination requirements and comparing them with actual coordination mechanisms. This paper, however, differs from Cataldo et al. (2006) in four important aspects: (1) This paper provides detailed mathematical justification, based on matrix algebra, for the expressions that allow us to determine systematically potential communication patterns based on the product architecture and the organizational affiliation of design engineers; (2) This paper acknowledges that the comparison of potential and actual organizational interactions is approximate and therefore corrects for any systematic redundancies built in our approach; (3) The analyses in this paper focuses on identifying mismatches of potential and actual communication patterns which are potentially indicative of coordination

issues that managers might want to attend to. In contrast, Cataldo et al. (2006) focus their analysis on testing the impact that aligning coordination requirements with actual coordination mechanisms have on design task performance; (4) This paper uses data collection methods that differ significantly from the empirical methods used by Cataldo et al. (2006), which relies primarily on analyzing "modification requests" data in a distributed software development organization. In that sense, both papers complement each other by providing alternative data collection mechanisms which managers could consider when implementing their efforts to improve coordination in software development organizations.

From a methodological viewpoint, this work builds on research in social networks that uses the notion of affiliation networks to study the relationship between individuals when they are affiliated with certain groups or events (Wasserman and Faust 1994). The membership of individuals to events, groups or other collectivities has been important in organizational research because these affiliations significantly influence the social identity of the individuals involved (Simmel 1955). Past research has also recognized that various types of networks exist within organizations due to interactions of systems, knowledge, tasks, and organizational units, and therefore properties can be measured in terms of any of these networks or combination of them (Carley 2002, p.10). The notion of affiliation networks formed by both social actors and social events has been represented in alternative ways, including affiliation matrices (also called *incidence matrices*), bipartite graphs, and hypergraphs (Seidman 1981). These representations have been used in engineering design to search for optimal ways to explore alternative strategies for decomposing complex design problems into more manageable sub-problems, which can ultimately lead to improved management of design iterations (Michelena et al. 1995, 1997; Chen et al. 2005, 2007), yet these methods do not evaluate the organizational implications of various problem decomposition alternatives.

This paper makes three important contributions to the engineering design literature. First, it operationalizes a general and structured approach to align product and organizational architectures with the use of the affiliation matrix. Because the affiliation matrix captures the level of involvement of organizational actors in the design of product components, it can be used systematically to estimate technical communication patterns during the planning and execution of product development efforts. Second, this paper illustrates and validates this structured approach by implementing it in a real software development setting. Doing this resulted in the identification of product interfaces and organizational interactions that required special managerial action in the organization studied. Third, this paper examines the interplay of process, product, and organizational structures in the same development organization. This provides us with an integrated view of how these separate but related perspectives align themselves when developing new products.

The structure of this paper is as follows: in the subsequent section, I present the research motivation by examining the task structure of the development organization studied. Then, I introduce the research approach and illustrate it with a simple numerical example. An industry example from software development is detailed in the subsequent section. Finally, after discussing the empirical results, I conclude the paper with a project management framework that aligns process, product, and organizational structures for better management of design iterations.

2 Research motivation: examining the process structure

Although there are substantial benefits associated with documenting and analyzing the structure of the processes organizations carry out when developing new products (Browning and Ramasesh 2007), it is important to realize that engineering managers need to go beyond the process domain into the product and organizational views in order to manage design iterations effectively. Next, I examine the information requirements of the development activities carried out in the software organization studied in this paper and illustrate the need to instantiate such a development process with the architecture of a particular product under development. This, in turn, determines the potential communication patterns of the organization during the completion of the most iterative set of development tasks.

The task structure of the development process used by the software firm I studied is represented in the design structure matrix shown in Fig. 1. This DSM representation captures their development process, internally documented in a multi-page process flow diagram. The matrix shown in Fig. 1 is a square matrix, the rows and columns of which are identically labelled with the development tasks, and an off-diagonal mark, (i,j), indicates that to complete task i (labelling row i) information from the task in column j is needed. 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 Fig. 1, an important contribution of a DSM representation is the simple and explicit depiction of complex processes where sets of iterative activities (i.e., design iterations) can be highlighted. The figure shows three sets of planned interdependent tasks: (1) software architecture definition; (2) software release planning; and

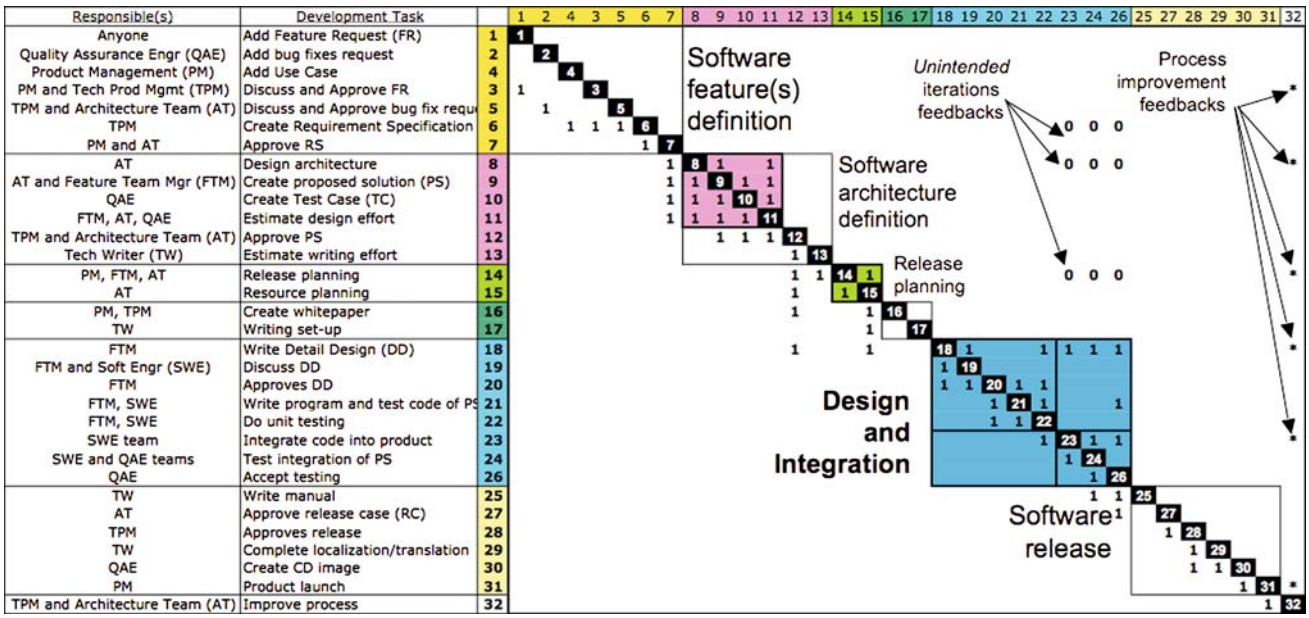

Fig. 1 Software development process at the organization studied

(3) design and integration of software features.[3] Yet, to effectively manage these planned design iterations one must examine how this process view is implemented for a given product with a specific organizational structure.

Figure 2 takes a closer look at the most iterative set of development tasks in the process documented in Fig. 1. The efficient completion of this group of *planned* highly iterative set of activities depends on both the specific product and organizational structures involved in the process. For example, the actual communication patterns associated with the tasks "Do unit testing", "Integrate code into product", and "Test integration of PS" (tasks 22, 23 and 24, respectively) were significantly different depending on the type of product under development:[4] For some legacy products the design and integration of proposed solutions (i.e., product components) would involve a small set of developers while designing and integrating components for a novel product such as the one studied in this paper involved more than half of the developers available in the organization. In addition, the organizational interactions associated with these design tasks largely depended not only on the inherent characteristics of the components to be designed, tested, and integrated, but also on the connectivity of those components with other components in the product. Hence, if managers are to be able to facilitate the completion of iterative set of design activities (as the ones shown in Fig. 2), they need to understand how the product components that instantiate these design tasks link to one another (the product architecture) as well as who the people responsible for contributing to the design of those product components are (the affiliation of people to the components' design). Next, I introduce a structured approach to address this challenge.

3 Predicting and managing technical interactions

In order to improve the management of planned design iterations that typically occur in the design phase of product development processes, this paper introduces a five-step approach structured in two phases (see Fig. 3). The first phase (steps 1, 2, and 3) focuses on predicting potential technical interactions based on the architecture of the product and the affiliation (or involvement) of developers in the design tasks of each product component. The second phase (steps 4 and 5) focuses on validating the potential interactions identified in the first phase by comparing them against actual interactions, which, in turn, provides important insights to improve the management of technical interactions. Fundamental to this approach is the introduction of the affiliation matrix (in step 2) to capture the design task involvement of the organization, which permits the alignment of product and organizational structures that do not map one-to-one.

[3] Note that Fig. 1 distinguishes *unintended* feedback interdependencies that could occur from the "design and integration" phase to either "release planning", "software architecture definition", or "software feature definition" phases. Because these interdependencies are unintended (or unplanned) they are not considered when identifying *planned* design iterations. This DSM also distinguishes the feedback interdependencies associated with process improvement because they are not part of planned design iterations either.

[4] The process illustrated in Figs. 1 and 2 was used for developing both legacy and novel software applications.

Fig. 2 "Design and Integration" tasks of the software development process studied

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

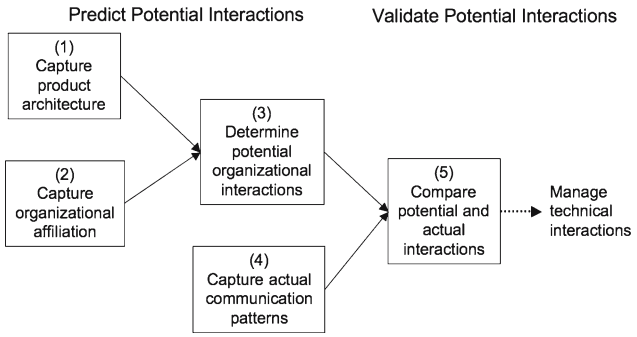

Fig. 3 A structured approach to predict and validate technical interactions

3.1 Predicting potential technical interactions

The basic assumption behind this first phase of the approach is that design interfaces between product components generate coordination requirements among the people involved in their design. This phase focuses on predicting the set of interactions that could potentially take place to coordinate the design interfaces of the product being developed. These potential interactions are likely to differ somewhat from the actual technical interactions that occur in the organization, which is why the results of this phase are validated by comparing them against the actual interaction patterns in the second phase of the approach. Predicting potential technical interactions can be done in three steps.

3.1.1 Capture the product architecture

The n components that form the product and the interfaces among them are identified by interviewing systems architects. The product data are then documented into a *product architecture matrix* ($\boldsymbol{P}$). $\boldsymbol{P}_{n,n}$ is a square matrix, the rows and columns of which 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 on component j. Note that this convention to represent design interfaces between components is opposite to the convention used in previous related work in which the components that impose constraints on other components are used to label the columns of the product architecture matrix (Sosa et al. 2003, 2007b, MacCormack et al. 2006). In this paper, I use the opposite convention because it facilitates the mapping of a matrix representation to a block diagram representation commonly used in software development (Sangal et al. 2005, Sosa et al. 2007a).[5] In addition, I assume that the directionality of the communication patterns follow the same direction as the directionality of design constraints (Sosa et al. 2004). Even though organizational communications are likely to be symmetric (i.e., in a dyad an actor seeks information while the other one provides information), *information-seeking* behavior (which is the type of organizational relationship we are aiming to predict) is typically determined by the directionality of design constraints (Eppinger et al. 1994; Sosa et al. 2007c). Nonetheless, for cases in which design constraints are used to predict symmetric organizational relationships, then the product architecture matrix can be symmetrized. Regardless of the convention used, the key point at this step is to capture the dependency structure of the product so that the corresponding communication patterns can be determined.

3.1.2 Capture the affiliation network: the affiliation matrix (A)

In order to capture the involvement of people in the design of product components systematically, it is important to recognize that development actors and product components form an affiliation network because developers are affiliated with (i.e., involved in the design of) product components (Wasserman and Faust 1994). This type of affiliation network is captured by asking the m development actors about their level of involvement in the design of each of the n product components (Alternatively, one could also ask engineering managers about the level of involvement of each of the m available developers in the design of the n product components). This information is documented in the *affiliation matrix* ($\boldsymbol{A}$). $\boldsymbol{A}_{m,n}$ is a rectangular matrix in which m rows are labeled with the development actors and n columns are labeled with the product components. Cell a_{ij} indicates the degree of

[5] When using block diagrams to represent the structure of software products, components that serve others as platforms to build upon are typically placed at the bottom of the diagram.

involvement of actor i in the design of component j. Note that for consistency the columns of $\boldsymbol{A}$ are sequenced following the same order as in the product architecture matrix ($\boldsymbol{P}$). As for the order of the rows, one can sequence people following the formal organizational structure in which developers are organized into groups so that group members are sequenced together.

Before describing how to use the affiliation and the product architecture matrices to determine potential organizational interactions, I will first examine the properties of the affiliation matrix, $\boldsymbol{A}$. To do this, let us consider a binary affiliation matrix in which $a_{ij} = 1$ if developer i is involved in the design of component j, otherwise $a_{ij} = 0$. In this case, the row marginal totals of $\boldsymbol{A}$, $a_{i+} = \sum_j a_{ij}$, are equal to the number of components to which developer i contributes to the design of. (As a result, a row where the total marginal is equal to zero indicates that such a developer does not contribute to the design of any product component.) Similarly, the column marginal totals of $\boldsymbol{A}$, $a_{+j=}\sum_i a_{ij}$, are equal to the number of people that contribute to the design of component j and therefore a column whose marginal total is equal to zero has no developers contributing to the design of such a component.

Affiliation networks are considered two-mode networks because they consist of a set of actors and a set of events (or, in our case, a set of components) instead of a set of elements (of the same kind) with links between them (Wassserman and Faust 1994). However, we are typically interested in the one-mode networks embedded in the affiliation networks, that is, the communication network of people in the development organization and/or the design dependency structure of the product under development. Fortunately, the information contained in the affiliation matrix itself sheds some light on these one-mode network structures. With the affiliation matrix we can determine "component-related ties" between people based on their involvement in the design of product components, and similarly, we can determine "organizational links" between components based on the people involved in their design.

First, let us compare the columns of the affiliation matrix to determine the number of developers that any pair of product components has in common. To that end, two components that share the same developers will have 1's in the same rows. That is, $a_{ik} = a_{il} = 1$ so that developer i contributes to the design of both components k and l. Counting the number of times that such an equality occurs for all the developers ($i = 1, \ldots, m$) results in the number of developers involved in the design of both components k and l (p_{kl}). Hence, $p_{kl} = \sum_{i=1}^{m} a_{ik} \cdot a_{il}$.

Clearly, if components k and l do not share any developers then $p_{kl} = 0$ (which is the minimum possible value) and, if all the developers contribute to the design of these two components, then $p_{kl} = m$ (which is the maximum possible value). Now, we can define the common-contributor product matrix ($\boldsymbol{P}_{\text{common-contributor}}$) as a function of the affiliation matrix ($\boldsymbol{A}$) as follows

$$P_{\boldsymbol{common-contributor}} = A^T A. \tag{1}$$

This is a square, valued, symmetric matrix of size n in which non-zero off-diagonal cells indicate the number of developers that any pair of components shares. The diagonal cells indicate the number of developers who contribute to the design of each component. Also, because the columns of the affiliation matrix are originally sequenced in the same order as the product architecture matrix ($\boldsymbol{P}$), then the $\boldsymbol{P}_{\text{common-contributor}}$ preserves the same label sequencing of $\boldsymbol{P}$.

In a similar fashion, we can use the affiliation matrix ($\boldsymbol{A}$) to determine the number of common components to whose design any pair of developers contributes. In such a case, we are interested in comparing the rows of the affiliation matrix so that $a_{ik} = a_{jk} = 1$ if both developers i and j contribute to the design of component k. Hence, we can define the common-component potential interaction matrix ($\boldsymbol{T}_{\text{common-component}}$) as follows

$$T_{\boldsymbol{common-component}} = AA^T. \tag{2}$$

This is a square, valued, symmetric matrix of size m, in which non-zero off-diagonal cells indicate the number of components to which a pair of developers contributes. The diagonal of such a matrix captures the number of components to which each developer makes a design contribution. Because people who contribute to the design of the same components are likely to exchange technical information related to the intrinsic design of such components, I call this matrix the common-component potential interaction matrix ($\boldsymbol{T}_{\text{common-component}}$). Note that this matrix does not capture the potential interactions that would need to take place to coordinate the actual product interfaces documented in the product interface matrix ($\boldsymbol{P}$) captured in the previous step. I tackle this challenge in the next step.

For illustration purposes, let us consider a simple organization with six developers developing a four-component product. Figure 4 shows the hypothetical affiliation matrix that captures how the six developers are affiliated with the design of each of the four product components. Figure 4 also shows how the affiliation matrix determines the potential interactions that could occur between any pair of developers due to the contribution they make to common components. In this example, *person 2* could potentially interact with *person 5* and *person 6* because they all contribute to the design of *component C*.

It is important to emphasize that $\boldsymbol{A}$ uniquely determines both $\boldsymbol{P}_{\text{common-contributor}}$ and $\boldsymbol{T}_{\text{common-component}}$, but the

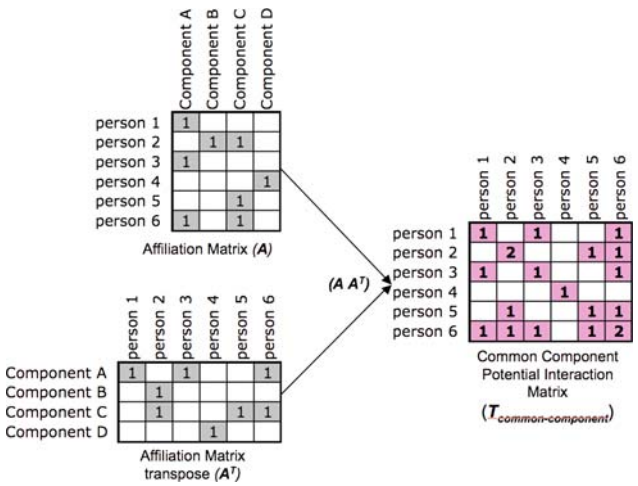

Fig. 4 Predicting common-component potential interactions

reverse is not true. Generally, the two latter matrices can be generated by a number of different affiliation matrices (Breiger 1991), so it is not possible to reconstruct the original affiliation matrix from either $\boldsymbol{P}_{\text{common-contributor}}$ or $\boldsymbol{T}_{\text{common-component}}$. This is because when considering these single-mode matrices one loses information about the affiliation network. That is, in the $\boldsymbol{P}_{\text{common-contributor}}$ matrix one loses identity of the people that contribute to the linked components, and in the $\boldsymbol{T}_{\text{common-component}}$ matrix one loses information about the components co-designed by any pair of actors. In addition, because the information captured in the $\boldsymbol{P}_{\text{common-contributor}}$ and $\boldsymbol{T}_{\text{common-component}}$ corresponds to pairs of components (which share the same developers) or pairs of developers (who contribute to the same components) respectively, one cannot infer any properties of subgroups larger than pairs from these one-mode network matrices (Breiger 1991; Wasserman and Faust 1994). For example, by examining $\boldsymbol{T}_{\text{common-component}}$ one can say that a pair of developers contributes to the design of certain number of components; however we cannot say that three or more developers contribute to the same set of components. To do so, we would need to examine the affiliation matrix ($\boldsymbol{A}$).

3.1.3 *Determine potential organizational architectural interactions*

I define architectural interactions as those that need to take place to coordinate identified product interfaces. To determine the set of potential architectural interactions between developers i and j we need to examine the entries of both the affiliation matrix ($\boldsymbol{A}$) and the product architecture matrix ($\boldsymbol{P}$). More specifically, developer i would look for technical information from developer j ($t_{ij} > 0$) if component k designed by developer i ($a_{ik} > 0$) depends on component l ($p_{kl} > 0$) which is designed by developer j ($a_{jl} > 0$). Hence, ($t_{ij} > 0$), if ($a_{ik} > 0$) and ($p_{kl} > 0$) and ($a_{jl} > 0$). Moreover, if we consider $\boldsymbol{A}$ and $\boldsymbol{P}$ to be binary matrices, then we are interested in the number of times that developers i and j need to coordinate product interfaces between components to which they contribute. That is, we are interested in adding the number of times that $a_{ik} = p_{kl} = a_{jl} = 1$ for developers i and j. Formally,

$$t_{ij} = \sum_{k=1}^{n} \sum_{l=1}^{n} a_{ik} p_{kl} a_{lj}. \tag{3}$$

Now, I can formally define the *potential architectural interaction matrix* ($\boldsymbol{T}_{\text{architectural}}$) to record the number of design interfaces in which each pair of developers would potentially need to interact. This matrix is, as might be expected, a function of both the affiliation matrix ($\boldsymbol{A}$) and the product architecture matrix ($\boldsymbol{P}$). Hence,

$$T_{\boldsymbol{architectural}} = APA^T. \tag{4}$$

This matrix is a square, valued matrix of size m. Note that this matrix is not symmetric if $\boldsymbol{P}$ is not symmetric, which is typically the case in software products. A non-zero cell, t_{ij}, indicates that developer i could potentially provide information to developer j because they are involved in the design of product components that share design interfaces. As for the diagonal elements of this matrix, they indicate the number of interdependent components with which a developer is involved.

Note that if one substitutes $\boldsymbol{P}$ by the identity matrix, $\boldsymbol{I}$, in Eq. (4), then one obtains the common-component interaction matrix, $\boldsymbol{T}_{\text{common-component}}$, which captures the potential organizational interactions that could take place among developers contributing to the same set of components without considering the interdependencies among components. As a result, to obtain the total set of potential technical interactions we simply add $\boldsymbol{T}_{\text{architectural}}$ and $\boldsymbol{T}_{\text{common-component}}$.

To illustrate the rationale behind Eq. (4), let us extend the simple numerical example introduced in Fig. 4. Figure 5 shows the product architecture matrix ($\boldsymbol{P}$) of the four-component product with six design interfaces, designed by an organization with six developers whose involvement in the design of each component is captured by the affiliation matrix ($\boldsymbol{A}$). The product $\boldsymbol{A}$ $\boldsymbol{P}$ produces a rectangular matrix in which non-zero cells capture the number of components with which developer i is involved, imposing design constraints on component j. For example, *person 2* is involved in the design of two components that impose design constraints on *component A* (such components are *components B* and *C*). Then, to obtain the potential architectural interaction matrix one must multiply

this matrix by A^T. Again, because we are using binary matrices, the cells in the resulting matrix capture the number of design interfaces that two developers potentially need to be able to coordinate. For example, *person 2* has three design interfaces that he or she would potentially need to coordinate with *person 6*. More specifically, *person 2* might need to provide design information to *person 6* about three product interfaces. Those interfaces are the two design interfaces from *components B* and *C* to *component A* (because *person 2* is involved in the design of *components B* and *C* and *person 6* is involved in design of *component A*), and the interface from *component B* to *component C* (because *person 2* is involved in the design of *component B* and *person 6* is involved in the design of *component C*). Of course, some of these potential interactions might not take place either because the same person is involved in the design of the two interdependent components (e.g., *person 2* is involved in the design of both *components B* and *C*, which share design interfaces in both directions) or because there is another pair of actors involved in those interfaces who are indeed coordinating such an interface. That is, in the same way that there is a potential interaction from *person 2* to *person 6* to deal with the interface from *component B* to *component A*, there are two other potential interactions that could take place to deal with such an interface. Those are the potential interactions between *person 2* and *person 1*, and between *person 2* and *person 3*.

Note that Eq. (4) allows us to determine "who should potentially seek technical information from whom?" to address *direct* dependencies between product components. However, such an expression could be slightly modified to predict the set of interdependent actors associated with indirect design interfaces. Because design changes tend to propagate beyond adjacent components (Clarkson et al. 2004; MacCormack et al. 2006; Sosa et al. 2007b), managers may be interested in determining who should seek information from whom to handle indirect interfaces between product components. For example, the system in Fig. 5 contains an indirect interface from *component D* to *component B* through *component C*. To determine the pairs of actors that could potentially handle such an indirect interface we can simply substitute P for P^2 in Eq. (4) because the non-zero cells of the square of a binary product architecture matrix documents the pairs of components that are linked through at least one intermediary component (i.e., product interface chains of length 2). Doing so, we find that *person 4* could seek information from *person 2* to handle the indirect interface from *components D* to *B*. Similarly, we could use P^3 to predict potential interactions associated with product interface chains of length 3 and so on. Nonetheless, the intention at this phase of the approach is simply to identify all possible potential interactions that could take place to address the identified direct product interfaces, given certain design task involvement in the organization captured in the affiliation matrix. Next, I compare potential interactions against actual interactions to test the validity of this approach.

Validating potential technical interactions In order to test the predictive power of the approach described in the three

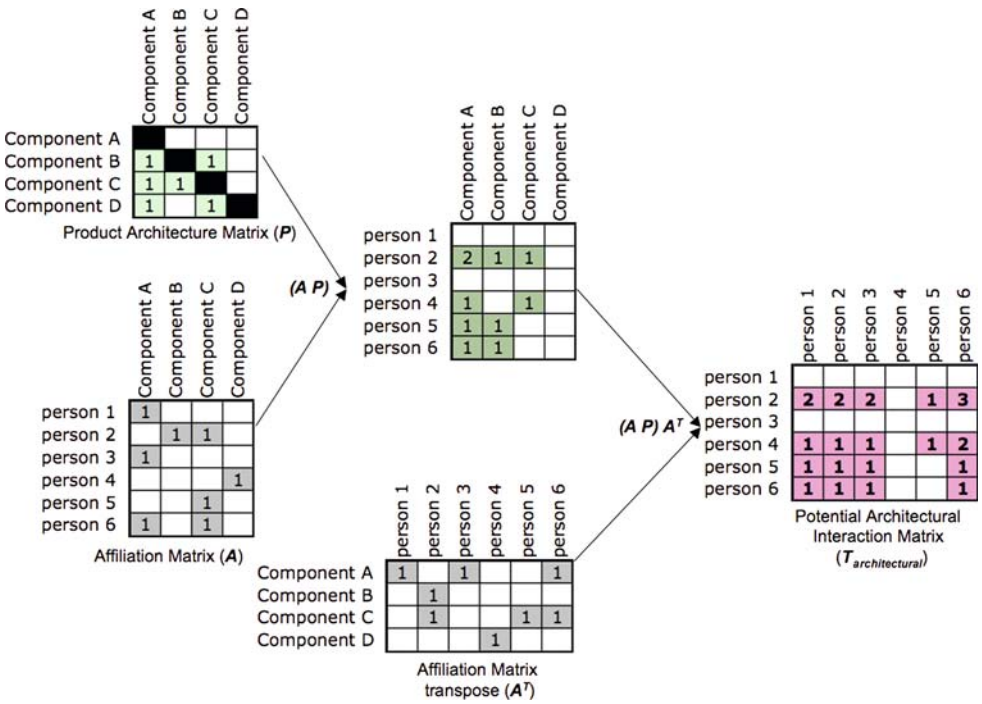

Fig. 5 A structured approach to predict potential architectural interactions

steps above it is important to compare the set of potential technical interactions between development actors against the actual technical interactions that take place during the development effort. This not only allows us to determine when a "match" of potential and actual interaction occurs, but more importantly it also allows us to uncover "mismatched" interactions, which are defined by the lack of overlap of potential and actual technical interactions. There are two types of mismatched interactions: (1) *unpredicted interactions*, which occur when an actual interaction is not predicted by a potential interaction, and (2) *unattended interactions*, which occur when a potential interactions does not correspond to an actual interaction. Hence, to identify matched and mismatched interactions two additional steps need to be carried out.

3.1.4 Capture actual organizational interactions

By surveying the m development actors involved in the development of the product, their actual product-related interactions (or the actual intentions to interact) are captured and documented onto a square (person to person) actual communication matrix ($C_{m,m}$). To be consistent with the convention used in steps 2 and 3, the sequence of the rows and columns of this matrix is identical to the sequence of rows in the affiliation matrix. In addition, to be consistent with the convention used in the product architecture matrix, the rows of the actual communication matrix ($\boldsymbol{C}$) 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).

3.1.5 Compare potential and actual interactions

In general, by overlaying binary versions of a potential interaction matrix and the actual communication matrix, one can systematically identify the set of potential mismatched interactions in the comparison matrix that emerges from such a comparison. However, we would need to perform different comparisons to determine the two types of mismatched interactions of interest.

Identifying unpredicted interactions Because unpredicted interactions are defined as those actual interactions that take place even though there are no potential interactions associated with them, it is important to compare actual interactions with all possible potential interactions identified. Hence, by overlaying the *total potential interaction matrix* ($\boldsymbol{T} = \boldsymbol{T}_{\text{common-component}} + \boldsymbol{T}_{\text{architectural}}$) and the *actual communication matrix* ($\boldsymbol{C}$), one can systematically document not only unpredicted interactions but also matched interactions in the preliminary comparison matrix. Figure 6 shows how the binary version of the total set of potential interactions maps onto the actual interactions in our hypothetical numerical example. The preliminary comparison matrix shows that the interactions from *person 3* to *person 2* and from *person 2* to *person 4* (both interactions labeled "O") are unpredicted by the architecture of the product and the design task involvement of the organization. Uncovering this type of mismatch is important for managers because their existence indicates that there might be unidentified product interfaces about which developers are interacting. Unpredicted interactions could also be the result of the "unofficial" involvement of some developers in the design of other components not assigned to them and therefore not captured in the affiliation matrix. The preliminary comparison matrix also shows a set of six cells (labeled "#"), in which the paired developers involved share both potential and actual interactions. Finally, there is a significantly high proportion of potential interactions that are not attended by actual interactions (labeled "X"). Which of these are truly potential unattended interactions? That is the question I address next.

Identifying truly potential unattended interactions I define truly potential unattended interactions as the subset of unattended potential interactions associated with product interfaces whose corresponding potential interactions are *all* unmatched by actual interactions. To identify the set of truly potential unattended interactions, we first filter out common-component potential interactions because we are interested in identifying the absolute minimum set of potential interactions that needs to take place to coordinate the identified set of product interfaces. First remember that, by definition, common-component potential interactions

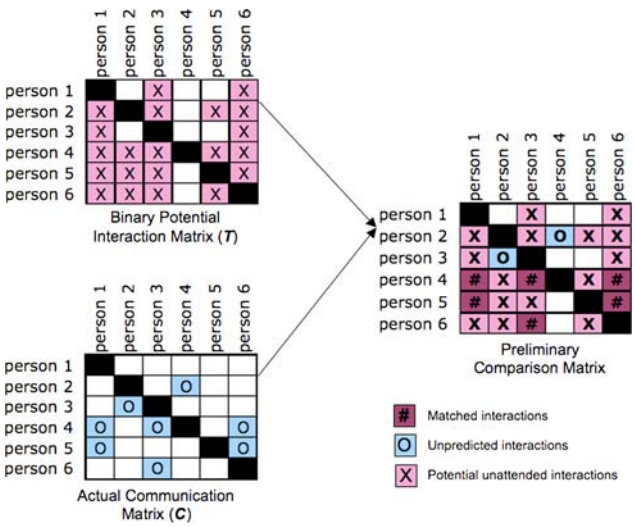

Fig. 6 Comparing potential and actual technical interactions

are not associated with any of the product interfaces, so we do not consider them when identifying truly potential unattended interactions.[6] Next, we filter out "redundant interactions" associated with each product interface identified. I define redundant interactions as those that do not necessarily take place because other people are already coordinating the product interface that generates such potential interactions. There are two types of redundant interactions: (1) *common-contributor redundant interactions*, which are potential interactions associated with product interfaces that have one or more people involved in the design of both interdependent components; and (2) *common-partner redundant interactions*, which are potential interactions associated with product interfaces in which at least one potential interaction is matched by an actual interaction.

Figure 7 illustrates how to identify systematically the two types of redundant interaction in order to obtain the *truly potential unattended interactions* in our hypothetical example. First, using the affiliation matrix we can identify the pairs of components that share developers involved in both interdependent components ("common-contributor components"). Such a matrix is the common-contributor product matrix ($\boldsymbol{P}_{\text{common-contributor}}$) defined in Eq. (1). Figure 7 shows that *components A* and *C* and *components B* and *C* have *person 6* and *person 2* as "common contributors" respectively. Note that the potential interactions associated with the interfaces corresponding with non-zero cells in the common-contribution matrix are common-contributor redundant interactions. To determine the common-partner redundant interactions, we must subtract the common-contributor interfaces from the product architecture matrix ($\boldsymbol{P}$) to obtain a subset of interfaces whose potential interactions are either common partner redundant interactions or truly potential unattended interactions. Hence, the preliminary set of potential unattended interfaces are defined by the following expression

$$\left[P - A^T A\right]^+ \tag{5}$$

where the $[\bullet]^+$ operator ensures that only the positive cells of the resultant matrix contained within brackets are considered potential unattended interfaces. In the example shown in Fig. 7, there are only three preliminary potential unattended interfaces between *components D* and *C*, *D* and *A*, and *B* and *A*, respectively.

To determine the common-partner redundant interactions, the potential interactions associated with each preliminary potential unattended interface are compared to the actual organizational interactions, similar to the method shown in Fig. 6. For example, the potential unattended interface from *component D* to *component C* shown in Fig. 7 generates three potential interactions (from *person 4* to *person 2*, to *person 5*, and to *person 6*. Yet, the latter one is the only potential interactions *matched* by an actual interaction. This makes the other two potential interactions common-partner redundant interactions. In general, we obtain a comparison matrix for each preliminary potential unattended interface. If such a comparison matrix contains at least one "matched" interaction, the potential interactions are common-partner redundant interactions because there is at least one pair of people that could coordinate such a product interface. Otherwise, the preliminary potential unattended interface is indeed a *potential unattended product interface* and its corresponding potential interactions are truly potential unattended interactions. Figure 7 shows that the potential interactions associated with the first two preliminary potential unattended interfaces are common-partner redundant interactions, while the interface from *component B* to *A* has three potential interactions that are truly potential unattended interactions. Note that these three potential interactions are neither common-contributor redundant interactions nor common-partner redundant interactions and as a result are truly potential unattended technical interactions. I have kept on using the term "potential" to refer to both unattended interfaces and unattended interactions because there might be alternative coordination mechanisms, such as interface standardization, that would not require the use of actual organizational interactions to handle these interfaces. Yet, this approach aims to help managers identify the subset of product interfaces that have higher risk of being overlooked in case there is no alternative mechanisms put in place to handle them.

Finally, Fig. 8 shows the final comparison matrix, which does not contain any redundant potential interactions. Out of the eight actual interactions, six were matched by potential interactions and two were unpredicted interactions. Interestingly, only three out of a total of 15 potential unattended interactions (see Fig. 6) were truly potential unattended interactions.

3.2 Managing technical interactions

If the approach described above indeed predicts systematically potential technical interactions, engineering managers could proactively select a subset of potential interactions to coordinate a subset of product interfaces that would be likely to change during the (re)design of a

[6] If we were to determine the truly potential unattended "common-component" interactions, we could do so by comparing the $\boldsymbol{T}_{\textit{pure-common-component}}$ matrix and the actual communication matrix ($\boldsymbol{C}$), where the entry $\boldsymbol{T}_{\textit{pure-common-component}}(i,j) = 1$ if $\boldsymbol{T}_{\text{common-component}}(i,j) > 0$ and $\boldsymbol{T}_{\text{architectural}}(i,j) = 0$. Such a comparison would yield the common-component potential interactions that were unattended by actual interactions.

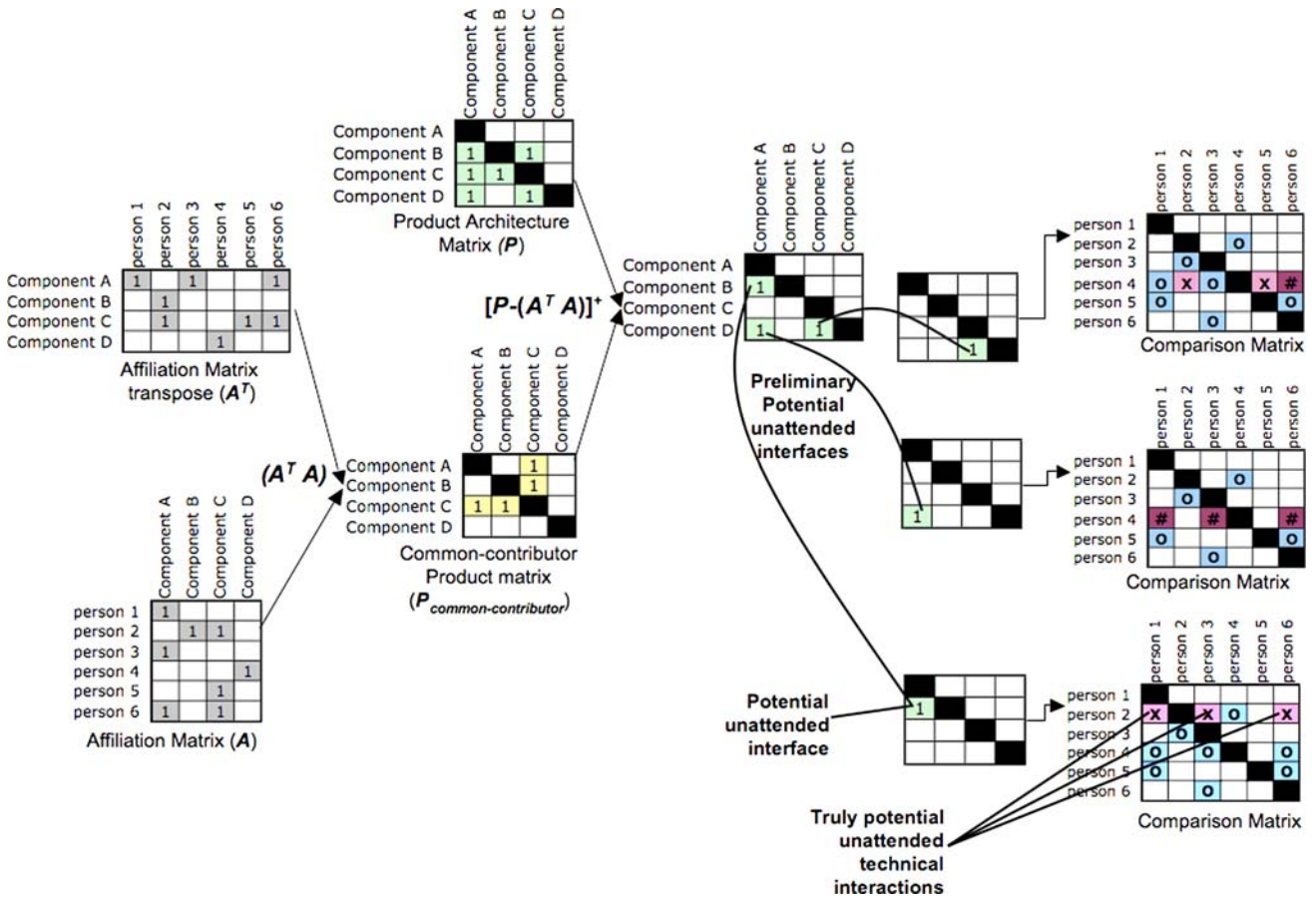

Fig. 7 Identifying truly potential unattended interactions

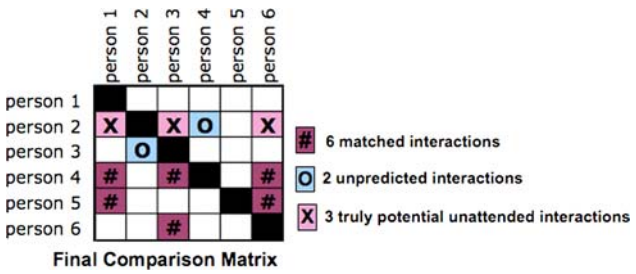

Fig. 8 Final comparison matrix

product. This is certainly relevant in software development because of the additive and flexible manner in which software products are developed (MacCormack et al. 2001; Sosa et al. 2007a). This is also important in the rapid redesign of hardware products in which some components are modified or added to an existing product architecture (Chen et al. 2007).

To illustrate how potential interactions can be managed proactively, let us consider again the example in Fig. 5, "Who should talk to whom if *component B* is redesigned?" (For simplicity I will assume that there are no architectural changes, that is, product components interfaces may change but they will not appear or disappear.) Using Eq. (4) with the same hypothetical affiliation matrix as before, Fig. 9 shows the potential interactions associated with the interfaces of *component B*. The top of Fig. 9 shows the potential interactions associated with all the interfaces of *component B* while the bottom shows the potential interactions associated with two subsets of interfaces separately. These two subsets correspond to the interfaces of *component B* with *components A* and *C*, respectively. The strong message that emerges from Fig. 9 is that *person 2* needs to coordinate some of (or perhaps all) the interfaces of *component B*. Certainly, *person 2* would need to coordinate with either *person 1*, *3*, and/or *6* on the interface between *components B* and *A*. In addition, the bi-directional interface between *components B* and *C* could be handled entirely by *person 2* (because she or he is involved in the design of these two components) or by having person 2 interacting with *persons 5* and *6*.

This approach does not provide a prescriptive recommendation about which product interface to facilitate; however, it systematically predicts the subset of potential organizational interactions from which to choose to fulfill product architectural requirements. By combining these

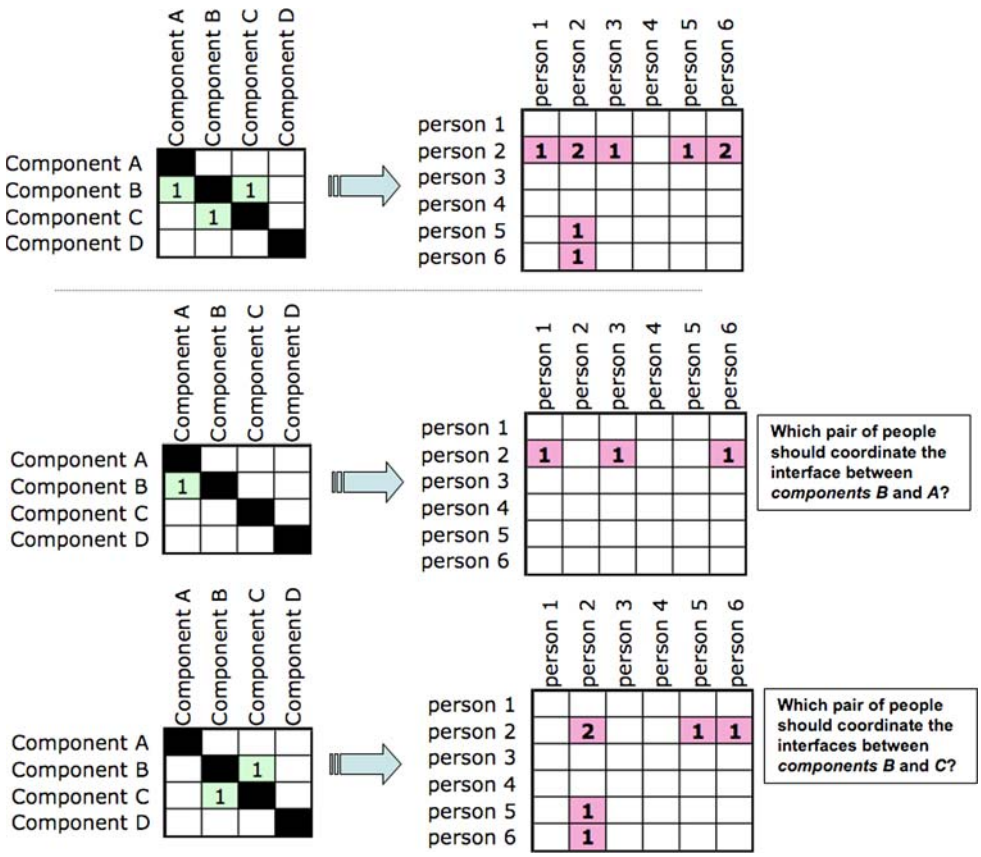

Fig. 9 Managing potential interactions to handle redesign of *component B*

coordination requirements based on the product architecture and design task involvement with other organizational requirements, such as individual design capability and individual workload, engineering managers can decide how to assign responsibilities and coordination mechanisms to ensure that critical interfaces are dealt with. As mentioned, some of those coordination mechanisms can be interface standardization. That is, some potential interactions might not require any actual organizational interactions because the corresponding product interfaces are standardized somehow. Moreover, by examining the number of potential interactions associated with certain product interfaces managers would be able to decide which interfaces to standardize in order to remove the need for managing certain organizational interactions. Finally, it is important to emphasize that this use of potential interactions is relevant if the predictive power of the approach described in the first phase of the approach (steps 1, 2, and 3) is significant when applied to a real setting. The next section provides empirical evidence that this may well be the case.

4 A software development example

I illustrate the implementation of the approach described above in the context of the development of a new software application. The hosting firm is a mature public European company and one of the leaders in the market for a specific type of application for business customers. The firm's portfolio of development projects included seven distinct software applications. At the time of data collection, the firm was allocating about 60% of its development resources to the development of one radically new product, an effort that had started within the previous 12 months. The product comprised 34 interdependent modules and the development organization included 66 people, many of whom contributed to the conception and design implementation of the 34 modules of the product. Two important factors informed the selection of this project. 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 test the validity of the structured approach detailed earlier. In addition, it is important to emphasize that understanding software development is valuable for two reasons: (1) complex products contain both software and hardware subsystems with software-related components playing increasingly important functional roles in product performance; (2) software development is somewhat different than hardware development because it is

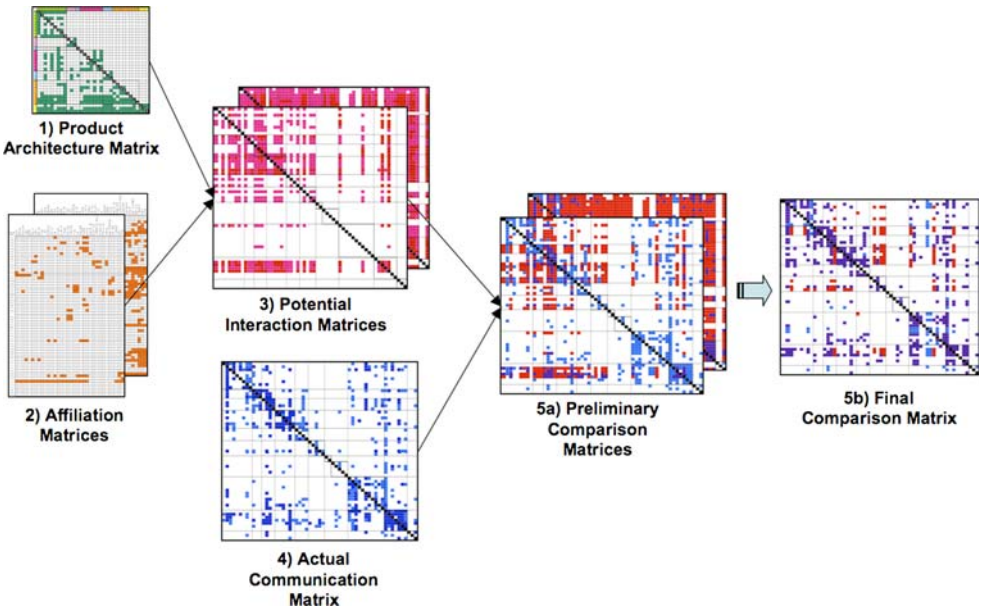

Fig. 10 Five steps to predict and validate technical interactions

typically faster, more flexible, and the mapping between the product architecture and the organizational structure is not one-to-one.

I implemented the structured approach described in the previous section in five steps (see Fig. 10). First, the software architecture was documented. Then, affiliation matrices were constructed to capture various levels of design involvement of the development actors. In step 3, potential interaction matrices were determined based on the product architecture and the affiliation matrices. Next, I documented the actual technical communication patterns in the development organization. Finally, by comparing potential and actual interactions, comparison matrices were created and matched and mismatched interactions were identified.

4.1 Step 1: Capturing the software architecture

After a long concept development phase, during which the firm assessed its market needs and technological opportunities, the architecture of the product to be analysed in this study was established. The product comprised 34 *modules*, the detailed design of which 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, a 34 × 34 product architecture binary matrix was constructed, where the 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 (Sangal et al. 2005; Sosa et al. 2007a). The product architecture matrix captures the directionality of the dependencies between product modules because system architects documented their dependency structure by explicitly considering their directionality so that "[developers] responsible for implementing and testing the specifications of module *j* should also find out about the specifications of module *i.*"For example, because product modules included in groups 5 and 6 (shown in Fig. 11) depended on most of the components included in group 1, managers were expecting developers of the former groups to seek technical information from developers involved in the design implementation of the components included in group 1.

In sum, I built a partitioned product architecture matrix to capture the dependency structure of the 34 modules that formed the software product studied. Figure 11 shows how the 34 modules of the product are organized into six groups of components. System engineers 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 posed significant managerial challenges due to the iterative constraints they would impose on some of the components. These interfaces are shaded in Fig. 11. In addition, the product architecture matrix highlights sub-system boundaries to show whether design interfaces occur with other components within the group or across group boundaries. In addition, Fig. 11 shows both a matrix representation and a block diagram of the product studied.

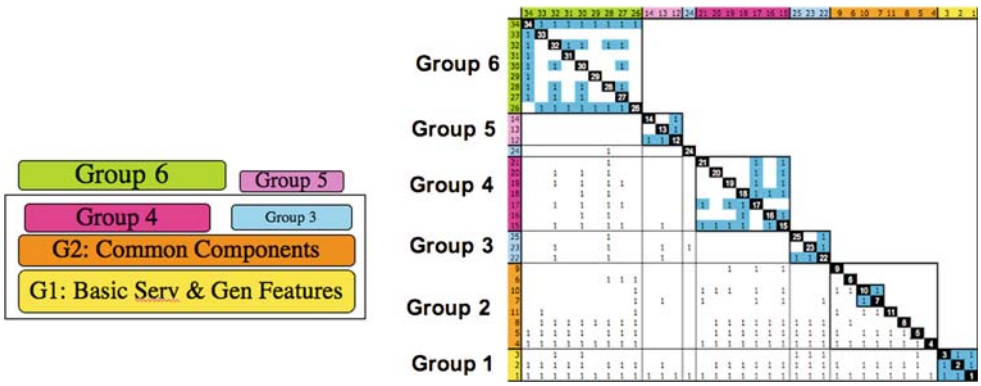

Fig. 11 Software product architecture studied

4.2 Step 2: Capturing the design task involvement of the organization

The development organization studied was formed by 66 people organized into 11 functional groups distributed in three different sites in Europe. Eight groups were dedicated to software development (i.e., programming), six of which were mainly 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 marketing, and technical documentation and IT support. A comprehensive web survey among all the 66 people involved in the development organization was distributed to capture not only their level of involvement in the design of the components of the product studied but also their product-related interactions with all the other people in the development organization. The web survey was completed by 59 respondents resulting in an 89% overall response rate. The survey was part of a comprehensive study that related the workload and the formal and informal organizational structure of this organization. Responses were missing from a few people on vacation during the data collection period or members of the support group whose input to the development process was less relevant. Moreover, the response rate among people involved in programming and testing activities was over 95%.

As part of the web survey questionnaire, respondents were asked to rate their level of involvement in the conception and implementation design of each of the 34 product modules. The six-point scale used to capture their level of involvement included the values "Not involved", "Barely involved", "Somewhat involved", "Involved", "Very involved", and "Strongly involved". I documented these data in a valued affiliation matrix. The columns of the affiliation matrix are labelled identically to the 34 columns of the product architecture matrix (see step 1 above) while the rows of the affiliation matrix are labelled with the 59 development actors that filled out the web survey. Hence, cell (i,j) in this matrix indicates the level of involvement of the person in row i in the conception and design implementation of the software module in column j. Finally, binary affiliation matrices were built for the following two cases: (1) design involvement rated as "strongly involved" only; (2) design involvement rated at least as "barely involved". Figure 12 shows the two binary affiliation matrices for these two cases respectively.

4.3 Step 3: Determining potential organizational interactions

Using the affiliation matrix, as indicated in Eq. (2), one can determine the potential interactions that could take place among developers involved in the design of the same components (i.e., potential common-component interactions). Using the two binary affiliation matrices built in step 2, I obtained 212 and 2124 potential common-component interactions for *strong-only* and *all-levels* of design involvement respectively. By combining the product architecture matrix and the affiliation matrices, as indicated in Eq. (4), one can determine the total number of interfaces between product modules on which any pair of developers need potentially to coordinate (i.e., potential architectural interactions). Hence, for the case of *strong-only* design involvement, the potential architectural interaction matrix captures 594 potential interactions. Such a matrix has a density of 17%. For *all-levels* design involvement, the potential architectural interaction matrix shows 2,306 potential interactions, which results in a communication network density of 67% (Fig. 13).

4.4 Step 4: Capturing the formal and informal development organizational structure

As described in step 2, the actual formal and informal organizational structures were captured by surveying almost all members of the development organization studied. The data were documented into an actual

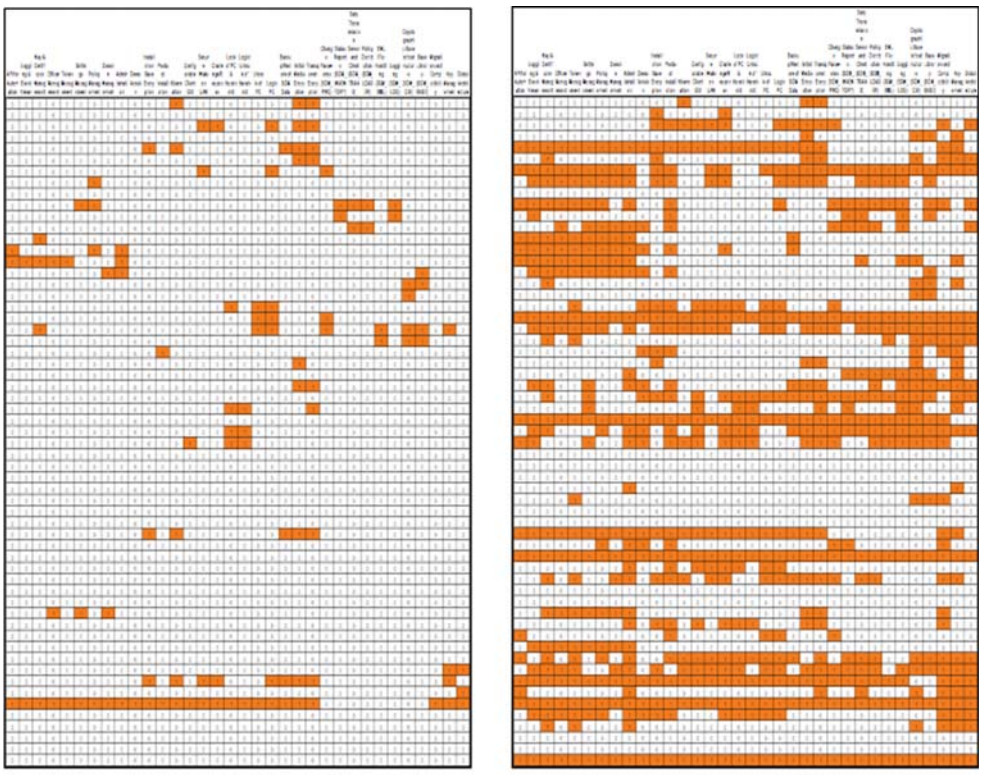

Fig. 12 Affiliation matrices in the organization studied

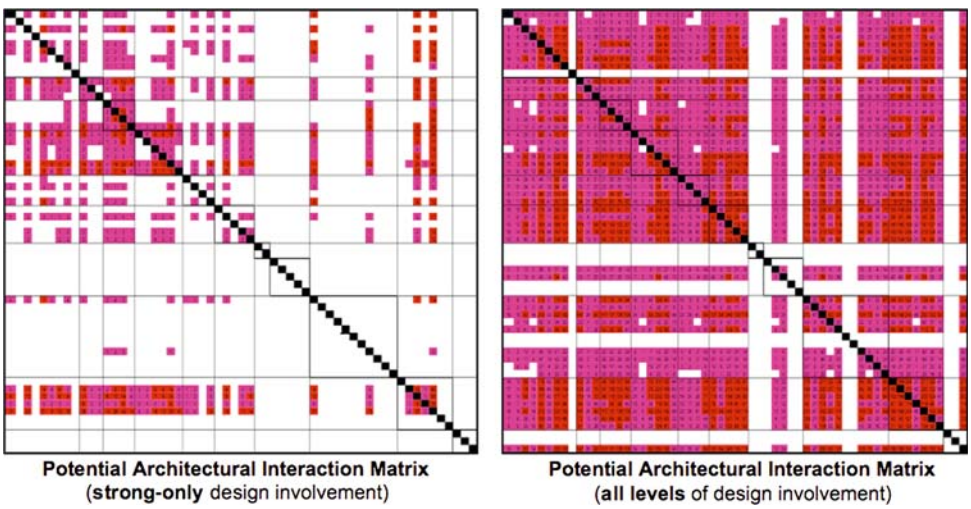

Fig. 13 Potential architectural interactions matrices

communication matrix where 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 the sequencing of the rows and columns of this matrix is identical to the sequence used in the rows of the affiliation matrices (as well as the sequencing obtained in the potential interaction matrices). Figure 14 shows the actual technical communication patterns associated with the development of the product studied in a 59 × 59 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 14 also shows the formal structure of the development organization into 11 functional groups. Note that the actual communication matrix highlights group boundaries with boxes along the diagonal so that interactions within boundaries (enclosed by those boxes) are distinguished from interactions across organizational groups.

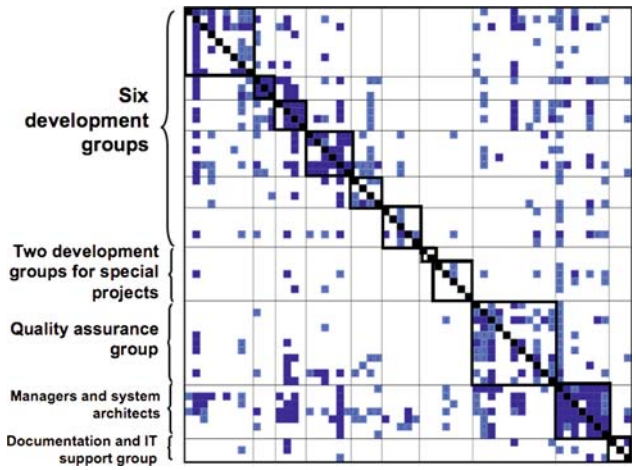

Fig. 14 Actual communication matrix

4.5 Step 5: Comparing actual and potential interactions

As described in the previous section, two distinct comparisons are needed to identify matched and mismatched interactions. The first comparison is focused on identifying *unpredicted technical interactions* while the second is used to uncover *truly potential unattended interactions*. Remember that *unpredicted interactions* are those that take place between development actors, even though they are not involved in the design of components that share interfaces with the other actors' components, while *truly 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.

Before describing how I identified mismatched interactions, it is important to mention two singularities in the organizational data collected. First, one of the respondents from the first development group (see Fig. 14) filled out completely the questions relevant to building the affiliation matrices; however, he or she did not manage to fill out the portion of the survey regarding the reporting of product-related interactions. As a result the column corresponding to this person is artificially empty in the actual communication matrix shown in Fig. 14. This column is therefore excluded from the comparison analysis described below, but the row corresponding to this person is kept in the analysis because it captures all the interactions from other members of the organization who requested technical information from this person. Second, the "managers and system architects" group (see Fig. 14) included two "technical marketing" managers who were "strongly involved" (from a marketing viewpoint) in the conceptualization and specification of over 75% of the product modules, however they were not expected to be involved in the design implementation of any of the product modules. This resulted in significantly (yet "artificially") dense rows in the affiliation matrix corresponding to these two technical marketing managers. As explained below, this will be an important factor to consider when identifying truly potential unattended interactions.

4.5.1 *Identifying unpredicted interactions*

To identify unpredicted interactions it is important to compare actual interactions with all potential interactions to rule out as far as possible any reasons that would justify the existence of actual product-related interactions. As a result, I compare actual organizational interactions with all the potential interactions for "all-levels" of design involvement (see Fig. 15). Note that the potential interaction matrix used combines both common-component and architectural potential interactions as determined by Eqs. (2) and (4) respectively. In this case, the total potential interaction matrix shows a high communication density of 68% (2315 potential interactions) 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 or if they are involved in the design of the same components. Yet, even after controlling for such a possibility, I still found, 71 *unpredicted interactions* (i.e., 14% of the actual interactions were unpredicted by the architecture of the product). These interactions took place between people who interacted (or planned to interact), even though they did not contribute to the design of the same components, nor did the components they designed share technical interfaces. This comparison also yields the set of potentially matched interactions, because I am assuming that even being "barely involved" in the design of a component provides enough justification

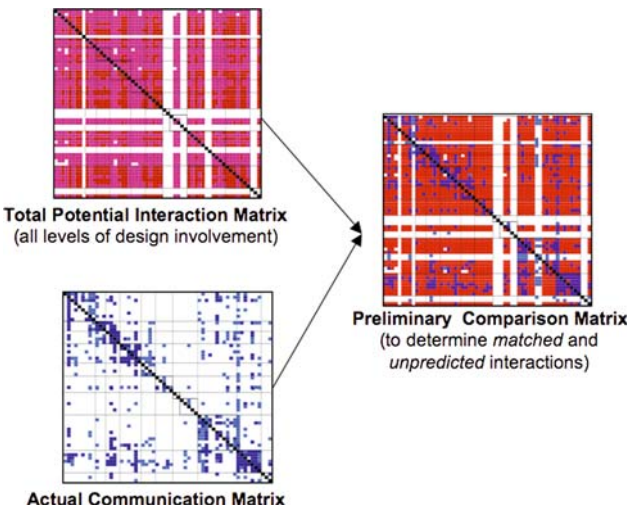

Fig. 15 Identifying matched and unpredicted interactions

for two interdependent actors to have technical organizational interaction. As a result, a total of 440 actual interactions were matched by potential interactions. Yet, the main objective of this comparison is to uncover the actual interactions that took place even though there were no technical reasons captured in both the product architecture and "all-levels" affiliation matrix to do so. In the next section I will discuss the organizational factors associated with the 71 unpredicted interactions uncovered here, but before doing so let us identify the other type of mismatched interactions.

4.5.2 *Identifying truly potential unattended interactions*

To identify the interactions that are truly potentially unattended, one needs to compare actual interactions with the minimum set of potential interactions that are expected to be truly necessary in the design process to coordinate the set of product interfaces identified. I started this comparison by considering the set of potential interactions among developers who are "strongly involved" in the design of the product components. Yet, in order to obtain the final set of *truly* potential unattended interactions, it is imperative to filter out potentially "redundant interactions" associated with each product interface identified. Hence, the underlying objective of this comparison is to identify the product interfaces whose potential interactions are not redundant interactions. Remember that a product interface may have several potential interactions associated with it because there might be several actors strongly involved in the design of the two components connected by such an interface. As a result some of these potential interactions may well be redundant interactions. As mentioned in the previous section, there are two types of redundant interactions: (1) potential interactions associated with an interface connecting two components that have at least one person strongly involved in the design of both components (common-contributor redundant interactions); and (2) potential interactions associated with an interface of which at least one of those potential interactions is matched by an actual interaction (common-partner redundant interactions).

Following the approach described in the previous section, to systematically identify truly potential unattended interactions I first filtered out the "common-contributor" redundant interactions using Eq. (1) with the "strong-only" affiliation matrix shown in Fig. 12, and then, filtered out "common-partner" redundant interactions. Interestingly, when doing this, the comparison yielded no truly potential unattended interactions. Does this mean that *all* the potential unattended interactions identified were redundant interactions? Very unlikely. Because the original "strong-only" affiliation matrix captured the involvement of the two "technical marketing" managers who were strongly involved in the specification of over 75% of the software modules but would not contribute to their design implementation, it was not realistic to assume that the potential interactions associated with them qualified as redundant interactions. As a result, it was necessary to revise the "strong-only" affiliation matrix by removing the entries in the rows corresponding to these two "technical marketing" managers. Note that this is not an issue with any other member of the organization because none else could report "strong involvement" in the design of a component without actually being involved in its design implementation.

Finally, after revising the "strong-only" affiliation matrix, I was able to identify the truly potential unattended interactions (see Fig. 16). First, I filtered out the "common-contributor" redundant interactions. Using Eq. (1) with the "revised" affiliation matrix, I identified 72 product interfaces (out of 250 product interfaces captured in the product architecture matrix) that had the same group of people strongly involved in the corresponding pair of interdependent components (see Fig. 16). That yielded a set of 178 product interfaces which were examined one by one, as illustrated in Fig. 7, to determine which of them would be associated with common-partner redundant interactions. The result of this exercise yielded 38 product interfaces which generated 100 potential interactions that were not matched by actual interactions (i.e., 100 truly potential unattended interactions). (Remember that those product interfaces, whose totality of potential interactions are not matched by actual interactions, are defined as *potential unattended product interfaces*, and those potential (unmatched) interactions are *truly potential unattended interactions*.) Identifying this set of potentially unattended interfaces is important because engineering managers can check whether they were intentionally unattended (because other coordination mechanisms, such as interface standardization, were associated with them) or they were indeed unintentionally unattended by the organization, in which case managerial action would need to follow.

The aggregated results of the two comparisons are documented in a *final comparison matrix* shown in Fig. 17. The cells with an "X" indicate *truly unattended potential interactions*, the cells with an "O" show the *unpredicted interactions*, and the cells with an "#" mark *matched interactions*.

5 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 potential *mismatches* between product and organizational architectures in cases where

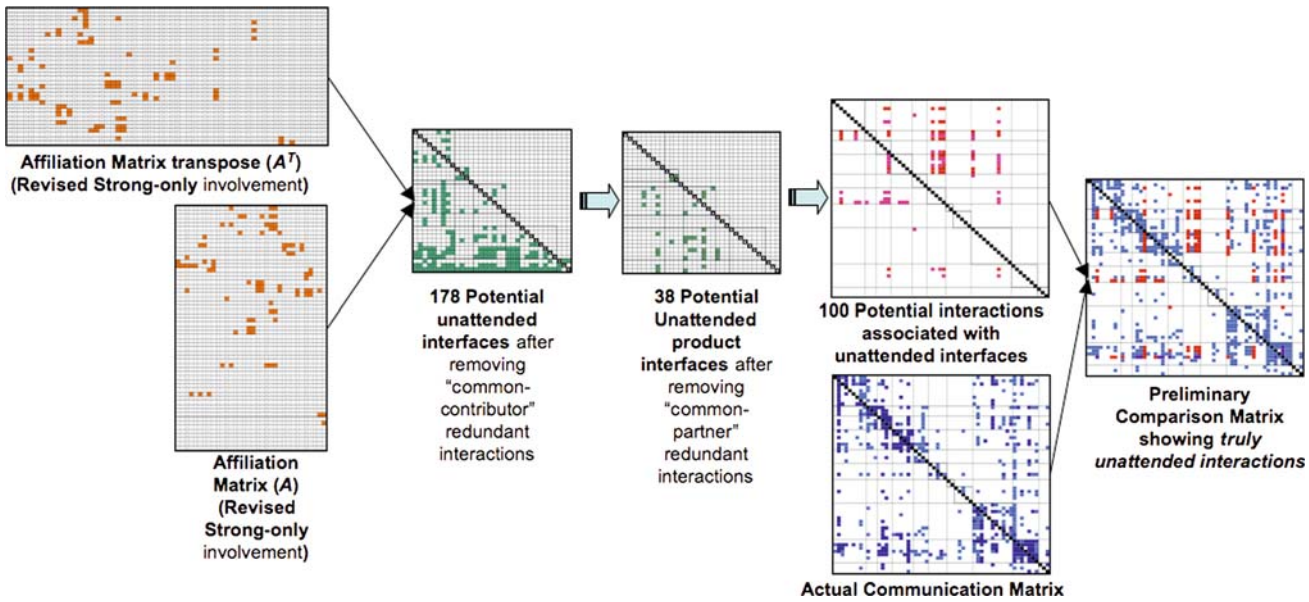

Fig. 16 Identifying truly potential unattended interactions

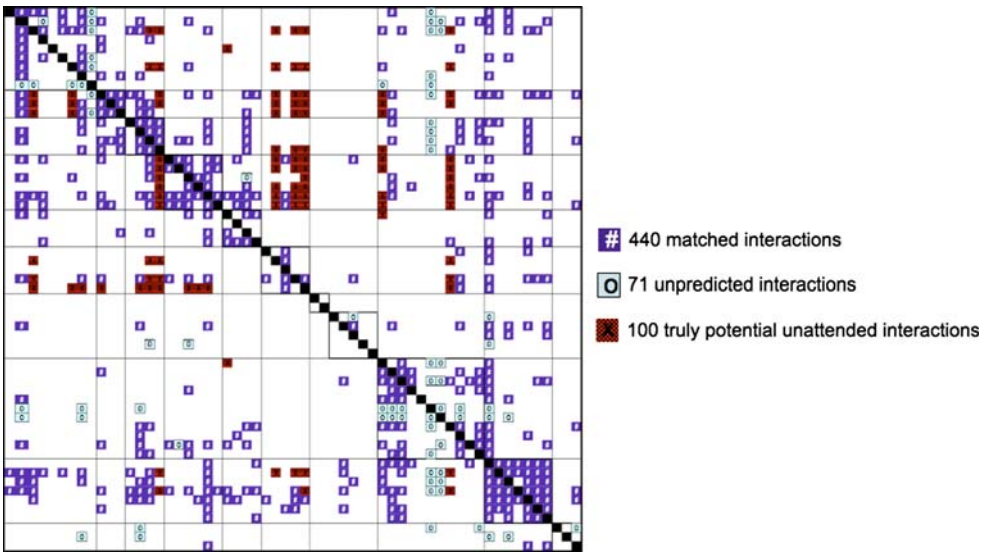

Fig. 17 Final comparison matrix

their structures do not map one-to-one. Identifying these mismatches in a systematic way can help managers steer their attention to areas within the product and the organization that may require special managerial action. More specifically, I found that only 14% of the 511 actual product-related interactions were unpredicted by potential interactions, while 29% of the 348 potential unattended interactions identified were truly potentially unattended. Moreover, in order to determine truly potential unattended interactions, I had to identify the product interfaces that were not associated with actual organizational interactions. In particular, of the 250 product interfaces identified by system architects, 15% of them had not been matched by actual interactions of people significantly involved in the design of such interdependent software modules.

Analyzing the final comparison matrix further allows us to test whether unattended and unpredicted interactions are concentrated in a few actors or are distributed throughout the development network. Fig. 18 shows a reordered final comparison matrix that clusters the 11 functional groups of the organization into three major groups according to their type of involvement in the product studied. First, the *technical group*, formed by the 31 members of the six development groups and the 11 members of the quality assurance group. These are the groups that are responsible for design, testing, and integration (i.e., programming, bug fixing, and product integration) of new (or redesigned) software modules of the products under development. Hence, they were expected to concentrate the majority of the technical interactions associated with the design

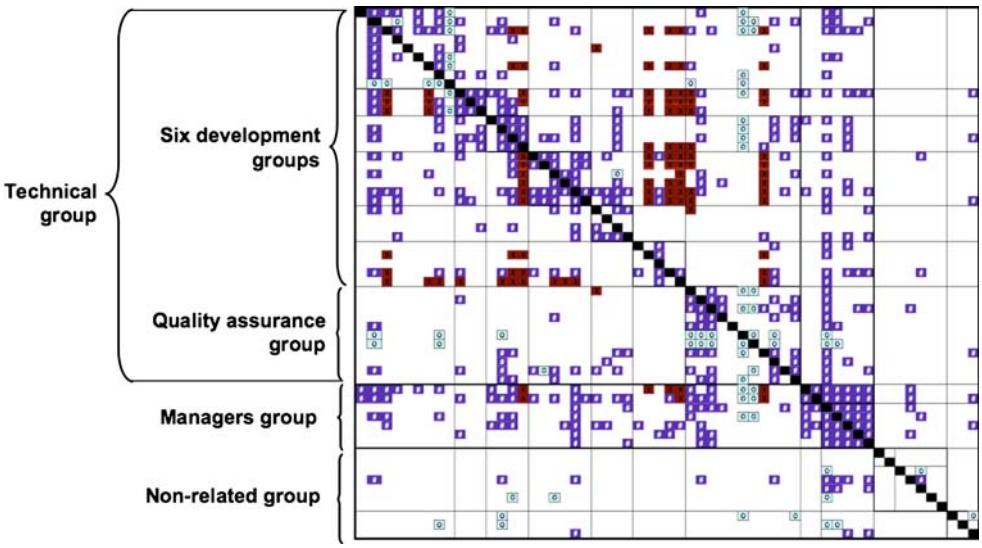

Fig. 18 Reordered final comparison matrix

implementation of the product studied. Second, the *managers group*, formed by the people who did not have a technical responsibility on the design implementation and testing of the product studied, yet have managerial responsibilities in the definition of the software modules (and their interfaces) to be implemented. This group included the seven managers of the organization including the director of the development organization, two technical marketing managers, two product line managers, and two system architects. Finally, the *non-related group*, formed by the seven members of the special projects groups and the three members of the documentation and IT support group. This group was not expected to have any significant interactions related with the conceptualization or design implementation of the product studied.

Table 1 shows how matched and mismatched interactions are distributed across the three groups defined above. First, note that 86% of the actual product-related communications were associated with potential interactions with a statistically significant large proportion of matched interactions occurring among members of the technical and managers groups.[7] Although it is not surprising to see that actual interactions were highly correlated with the existence of potential interactions, the benefit of identifying matched interaction is that when doing so, one can uncover unpredicted interactions. Table 1 shows that a statistically significant large proportion of unpredicted interactions occurred between people in the technical group, which suggests that technical interactions occurred for reasons that were not captured by either the architecture of the product or the affiliation network of the organization.[8] Table 1 also shows that 92% of the truly potential unattended interactions occurred within the technical group, which suggests that potentially unattended product interfaces are likely to be associated with lack of interaction among technical people in the organization that were supposed to be significantly involved in the design implementation of the product.

More generally, because mismatched interactions reveal important information about potentially unattended identified product interfaces as well as unanticipated technical interactions, the results reported in Table 1 suggest that it is particularly valuable to applying the structured approach presented in this paper to the group of technical people that are supposed to be directly involved in the iterative design activities of the product under development. Accordingly, it is important to examine further how mismatched interactions are distributed within the technical group in order to understand the driving forces behind their occurrence.

5.1 Factors associated with potentially unattended and unpredicted interactions

I found that truly potential unattended interactions were significantly concentrated in a small group of actors. Ninety-one percent of the 92 truly potential unattended interactions within the technical group were associated with nine actors. This is good news for managers because

[7] To test statistically the significance of the difference between the proportions of matched interactions (within technical and managers group versus non-related group), I carried out a chi-square test over the 440 matched interactions. The expected values were determined by the probability that a matched interaction would randomly occur between technical and manager actors instead of with non-related actors. Such probabilities were defined by the available set of possible interactions in these two categories. The test resulted in a $\chi 2$ of 136.9, which is clearly greater than the critical value of $\chi 2(0.99, 1) = 6.63$.

[8] To test the significance of this result, a chi-square test was carried out over the 71 unpredicted interactions. This test resulted in a $\chi 2$ of 13.2, which is greater than the critical value of $\chi 2(0.99, 1) = 6.63$.

Table 1 Overall distribution of technical interactions

	Counts of matched interactions	Counts of unpredicted interactions	Counts of truly potential unattended interactions
Interactions within technical group	248 (56%)	49 (69%)	92 (92%)
Interactions with managers group	162 (37%)	9 (13%)	8 (8%)
Interactions with non-related group	30 (7%)	13 (18%)	0 (0%)
Total	440	71	100

they can focus their attention on a small set of actors to minimize the risk of overlooking critical product interfaces. Interestingly, 100% of these potentially unattended interactions occurred across group boundaries, which confirms the importance of carefully identifying and managing cross-boundary interfaces; these suffer from communication barriers imposed by organizational boundaries between development groups which typically hinder the attention that needs to be paid to technical interdependencies (Sosa et al. 2004). As mentioned before, an important benefit of the suggested approach to identifying truly potential unattended interactions is that it requires the identification of product interfaces that are not matched by actual interactions. I found that 38 product interfaces (out of 250) were not attended by actual interactions. Figure 19 shows these potentially unattended product interfaces, of which 27 (or 71%) involved components from subsystem "group 4", which clustered a set of particularly novel software modules that were subsequently highlighted for special attention during the detailed design phase.

Certainly, potentially unattended product interfaces can be coordinated in many different ways. In addition to actual interactions between the people significantly involved in the design implementation of these interdependent components, alternative coordination mechanisms available for organizations include: interface standardization, indirect interactions through intermediary actors (either within the technical group or with other members of the organization), and interface coordination through actual interactions between people with lower level of design task involvement. In the product studied, interface standardization was less likely to play an important role at this stage of the development process because the architecture of the product was based on functional components to be designed and implemented. I also tested for the possibility that actors with lower levels of design involvement would coordinate some of these potentially unattended interfaces and found that over 21% of the 38 potential unattended interfaces could have not been handled by other actors with lower levels of involvement in the design of these components. More generally, it is worth emphasizing that the value of identifying potentially unattended interfaces is to "raise a flag" around a subset of product interfaces that have higher risk of being overlooked by the organization so that managers investigate whether they are intentionally or unintentionally unattended.

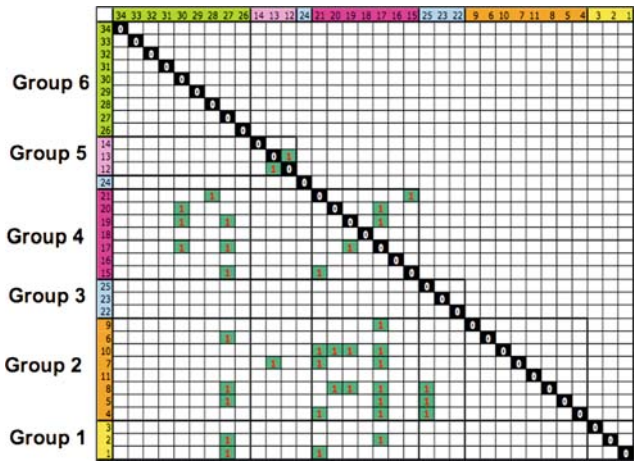

Fig. 19 Potentially unattended product interfaces

As for the unpredicted interactions within the technical group, three people (one developer and two members of the quality group) were involved in 92% of them. There are two distinct explanations for this: either the product architecture matrix did not capture some important interfaces that motivated these unpredicted interactions or the affiliation matrix did not capture the involvement of these people in some design activities associated with some product components. In this case, the empirical evidence suggests that the second explanation is the most plausible one. These three development actors did not report *any* level of involvement in the design of any of the components of the product studied (i.e., their corresponding rows in the affiliation matrix were empty for all levels of involvement) because they were supposed to be involved in the design and testing activities of other products in the firm's portfolio. However, the member of the development group in question was likely to be involved in technical discussions about the product studied with other developers in his group because he was considered "expert" in some of the technologies relevant for the product studied, and

this was reflected in his actual communication patterns. As for the two quality assurance engineers, they were likely to be involved in some "testing" tasks associated with the product studied to help cover the excess demand for quality related activities associated with this product development effort. After understanding the main reasons behind the unpredicted interactions with these three people, the other unpredicted interactions became candidates to be investigated by engineering managers to find out whether they were related to any previously unidentified product interface not captured in the original product architecture matrix.

It is worth highlighting the fact that the predictive power of our approach to determine and validate potential interactions depends significantly on the accuracy of the data collected. Certainly, measurement errors in any of the matrices can systematically generate mismatched interactions. As a result, managers should first double check that the systematic occurrence of mismatched interactions is not simply the result of systematic errors during the data collection. Fortunately, the codified nature of software products and the increasing use of information technology to manage software development projects may help reducing inaccuracies in the data collected.

6 Managerial and academic implications

This research has important implications for both managers and academics. Research in engineering design has suggested that the identification of design iterations is essential to managing them effectively (Eppinger et al. 1994; Eckert et al. 2004). In this paper I argue that to manage planned design iterations, it is essential for engineering managers to identify the set of actors who need to interact and the interfaces they need to interact about. This is particularly relevant in software development in which design and integration activities take place concurrently as products are built. This paper presents a structured approach to this challenge. Moreover, the systematic implementation of this approach to small "portions" of the product and within the relevant technical group in the development organization 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 the potential interactions that would address those interfaces effectively.

Figure 20 illustrates how, by bringing together the process, product, and organizational views, managers can effectively facilitate the management of design iterations associated with a subset of product components. Figure 20 starts with a representation of the development process in which its main iterative phases are highlighted, including the "design integration" phase in which most of the planned design iterations take place. To manage those iterations effectively, managers capture the architecture of the specific product whose product components are to be designed, tested, and integrated. For illustration purposes, let us assume that managers are interested in facilitating the design iterations associated with designing, testing, and integrating the product components of the particularly novel subsystem "group 4". As a result, the product architecture shown in Fig. 20 only shows the interfaces (both within and across subsystem boundaries) associated with the product components of subsystem "group 4". 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 the technical people of the development organization who significantly contribute to the design, test, and integration of the software modules whose interfaces are highlighted in the product architecture matrix. Based on the results discussed in the previous section, I have resized both the affiliation matrix and the potential interaction matrix to consider interactions within the development and quality assurance groups only (i.e., the technical group). The potential interaction matrix provides managers with the set of people and technical interactions from which to choose to coordinate the identified product interfaces effectively. Moreover, the top half of Fig. 21 shows the potential interactions, some of which, would be needed to coordinate the interfaces between components within the "group 4" subsystem. On the other hand, the bottom half of Fig. 21 shows the potential interactions, some of which, would be needed to coordinate the interfaces between the components of "group 4" and components in other subsystems. Note that most of the within-subsystem boundary interfaces can be handled by members of two development groups (labeled as G4 and G6 in Fig. 21) while to coordinate the set of cross-boundary interfaces developers from five of the six development groups would potentially be involved as well as a couple of members of the quality assurance group. Having considered these potential interaction patterns, then it is up to engineering managers to establish the appropriate coordination mechanisms to facilitate the management of such product interfaces.

As illustrated in Fig. 20, to implement a project management framework like this, it is essential to document both the product architecture and the design task affiliation of the organization. With product architecture and affiliation matrices, a project management tool can be implemented to automate the systematic prediction and management of potential interactions. The automation of

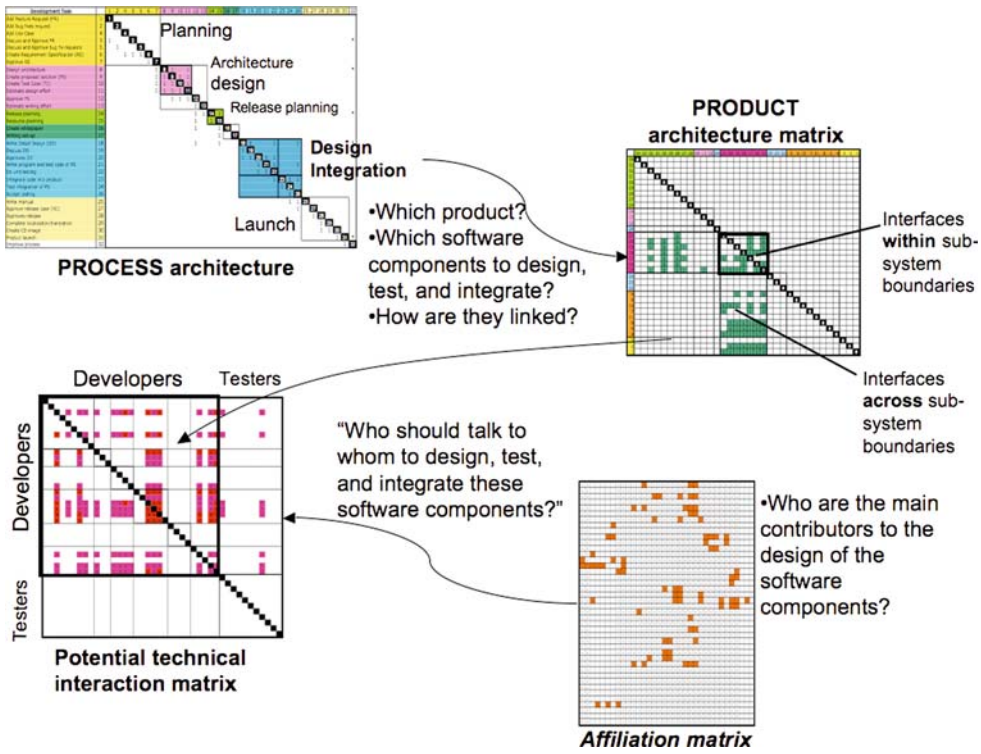

Fig. 20 Aligning process, product, and organizational views in software development

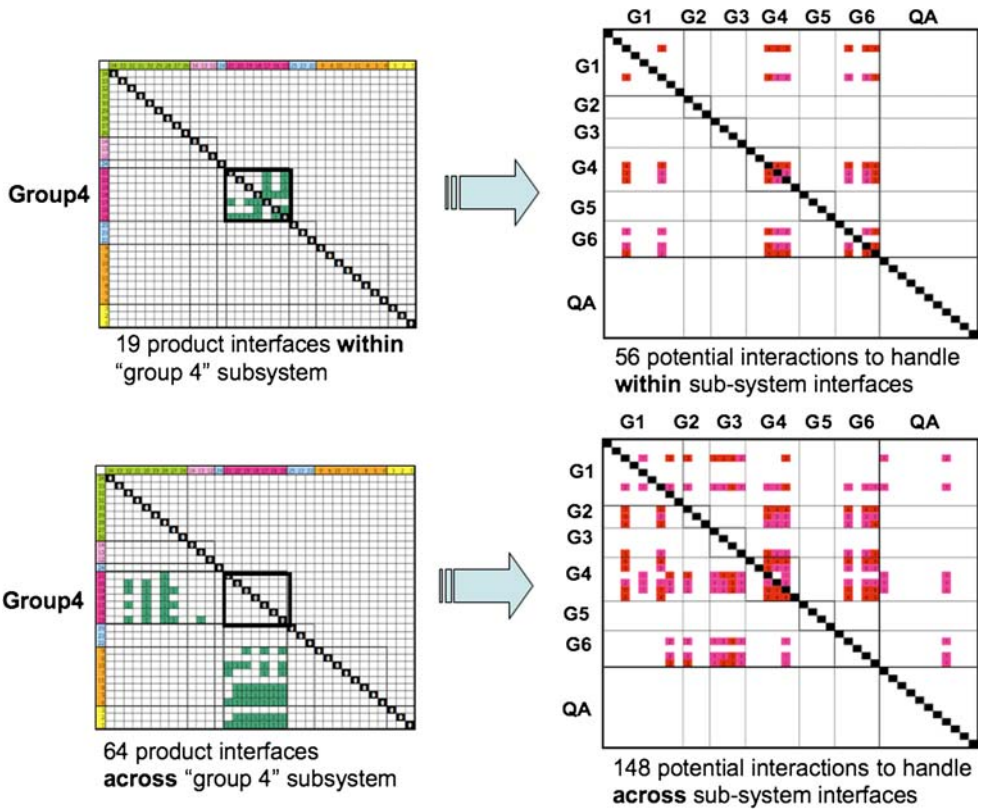

Fig. 21 Facilitating potential technical interactions

the structured approach introduced in this paper would allow for the rapid evaluation of the organizational impact of changes in either the architecture of the product or the affiliation network of the organization.

In practical terms, this paper highlights the importance of capturing both the connectivity of product components and the involvement of developers in the design of those components. Doing so is facilitated in software

development due to the highly *codified* nature of software products. This paper has shown how the software architecture can be documented based on functional components of the product, however after product components are implemented into "pieces" of source code, the source code itself can be used to capture the architecture of the product being implemented and maintained (Refer to Sangal et al. 2005; MacCormack et al. 2006; Sosa et al. 2007a for examples of how to document software architectures based on source code). Once a product architecture matrix is constructed, then engineering managers and developers can easily determine which other components are likely to be affected if a group of components are to be changed. Then, with the affiliation matrix managers and developers can also identify which people to contact to coordinate any particular interface between the focal component and any other components. Again, this sort of project management framework is particularly relevant in software development where products are developed in an additive fashion and product changes are implemented over an established product architecture that is relatively easy to document.

More generally, the approach presented in this paper allows engineering managers to identify potential mismatches between actual and potential interactions. This is important because mismatched interactions offer important guidance to engineering managers to update product (or process) information when new interfaces are uncovered by unpredicted interactions, and to reorganize development actors to attend interfaces that could otherwise be overlooked. Although the approach has been illustrated in an in-depth case study in a software development organization, additional validation in other types of technical organizations would be required before generalizing the results presented here. From a theoretical viewpoint, the implications of this approach rest on the analytical usage of the affiliation matrix (in its binary and valued forms) to explore alternatives ways to cluster organizational groups to minimize tension across organizational boundaries.

Acknowledgments I appreciate the active participation of the executive team and members of the development organization of the firm where the empirical study was conducted. I thank Jürgen Mihm for his insightful feedback on an earlier version of this article. Portion of this manuscript was presented at the *14th International Conference in Engineering Design* (*ICED '07* Paris, France). Finally, I appreciate the comments and suggestions of three anonymous reviewers which helped significantly in improving the final version of this article.

References

Allen TJ (1977) Managing the flow of technology. MIT Press, Cambridge

Baldwin CY, Clark KB (2000) Design rules: volume 1: the power of modularity. MIT Press, Cambridge

Breiger RL (1991) Explorations in structural analysis: dual and multiple networks of social structure. Garland Press, New York

Browning TR (2001) Applying the design structure matrix to system decomposition and integration problems: a review and new directions. IEEE Trans Eng Manage 48(3):292–306

Browning TR, Ramasesh RV (2007) A survey of activity network-based process models for managing product development projects. Prod Oper Manage 16(2):217–240

Carley KM (2002) Smart agents and organizations of the future. In: Lievrouw L, Livingstone S (eds) The handbook of new media. Chap. 12. Sage, Thousand Oaks, pp 206–220

Cataldo M, Wagstrom P, Herbsleb JD, Carley KM (2006) Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of ACM conference on computer-supported cooperative work, Banff Canada, pp 353–362

Chen L, Ding Z, Li S (2005) A formal two-phase method for decomposition of complex design problems. ASME J Mech Des 127:184–195

Chen L, Macwan A, Li S (2007) Model-based rapid redesign using decomposition patterns. ASME J Mech Des 129:283–294

Clarkson PJ, Simons CS, Eckert CM (2004) Predicting change propagation in complex design. ASME J Mech Des 126(5):765–797

Eckert CM, Clarkson PJ, Zanker W (2004) Change and customization in complex engineering domains. Res Eng Des 15(1):1–21

Eppinger SD, Salminen VK (2001) Patterns of product development interactions. In: International conference on engineering design, ICED '01, vol 1, pp 283–290

Eppinger SD, Whitney DE, Smith RP, Gebala DA (1994) A model-based method for organizing tasks in product development. Res Eng Des 6(1):1–13

Henderson R, Clark K (1990) Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. Adm Sci Q 35(1):9–30

Jarratt T, Clarkson J, Eckert C (2005) Engineering change. In: Design process improvement—a review of current practices, pp 266–285

Lai X, Gershenson JK (2006) Representation of similarity and dependency for assembly modularity. In: Proceedings of the ASME design engineering technical conferences—18th international conference on design theory and methodology, Philadelphia

MacCormack A, Verganti R, Iansiti M (2001) Developing products on internet time: the anatomy of a flexible product development process. Manage Sci 47(1):133–150

MacCormack A, Rusnack J, Baldwin C (2006) Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Manage Sci 52(7):1015–1030

Michelena N, Papalambros PY (1995) A network reliability approach to optimal decomposition of design problems. ASME J Mech Des 117(3):433–440

Michelena N, Papalambros PY (1997) A hypergraph framework for optimal model-based decomposition of design problems. Comput Optim Appl 8(2):173–196

Mihm J, Loch C, Huchzermeier A (2003) Problem-solving oscillations in complex engineering projects. Manage Sci 46(6):733–750

Morelli MD, Eppinger SD, Gulati RK (1995) Predicting technical communication in product development organizations. IEEE Trans Eng Manage 42(3):215–222

Olson J, Cagan J, Kotovsky K (2006) Unlocking organizational potential: a computational platform for investigating structural interdependence in design. In: Proceedings of ASME conference on design theory and methodology

Pimmler TU, Eppinger SD (1994) Integration analysis of product decompositions. ASME conference on design theory and methodology, Minneapolis, pp 343–351

Sanchez R, Mahoney JT (1996) Modularity, Flexibility, and Knowledge Management in Product and Organization Design. Strateg Manage J 17:63–76

Sangal N, Jordan E, Sinha V, Jackson D (2005) Using dependency models to manage complex software architecture. In: Proceedings of the 20th annual ACM SIGPLAN conference on object oriented programming, systems, languages, and applications, San Diego

Seidman S (1981) Structures induced by collections of subsets: a hypergraph approach. Math Soc Sci 1:381–396

Sharman D, Yassine A (2004) Characterizing complex products architectures. Syst Eng 7(1):35–60

Simmel G (1955) Conflict and the web of group affiliations. Free Press, Glencoe

Sosa ME, Eppinger SD, Rowles CM (2003) Identifying modular and integrative systems and their impact on design team interactions. J Mech Des 125(2):240–252

Sosa ME, Eppinger SD, Rowles CM (2004) The misalignment of product architecture and organizational structure in complex product development. Manage Sci 50(12):1674–1689

Sosa ME, Browning T, Mihm J (2007a) Studying the dynamics of the architecture of software products. In: Proceedings of the ASME design theory and methodology conference

Sosa ME, Eppinger SD, Rowles CM (2007b) A network approach to define modularity of components in complex products. ASME J Mech Des 129(11):1118–1129

Sosa ME, Gargiulo M, Rowles CM (2007c) Component connectivity, team network structure, and attention to technical interfaces in complex product development. INSEAD working paper 2007/68/TOM/OB

Steward D (1981) The design structure matrix: a method for managing the design of complex systems. IEEE Trans Eng Manage EM 28(3):71–74

Terwiesch C, Loch CH, De Meyer A (2002) Exchanging preliminary information in concurrent engineering: alternative coordination strategies. Org Sci 13(4):402–419

Wasserman S, Faust K (1994) Social network analysis. Cambridge University Press, NY