2009

# The Coordination Pyramid: A Perspective on the State of the Art in Coordination Technology

Anita Sarma
*University of Nebraska-Lincoln*, asarma@cse.unl.edu

David Redmiles
*University of California, Irvine*, redmiles@ics.uci.edu

André van der Hoek
*University of California, Irvine*, andre@ics.uci.edu

# *The Coordination Pyramid: A Perspective on the State of the Art in Coordination Technology*

TR-UNL-CSE-2009-0017

*Anita Sarma*
Department of Computer Science & Engineering
University of Nebraska, Lincoln
Lincoln, NE 68588 USA
asarma@cse.unl.edu

*David Redmiles, André van der Hoek*
Department of Informatics
University of California, Irvine
Irvine, CA 92697-3440 USA
{redmiles, andre}@ics.uci.edu

# *The Coordination Pyramid: A Perspective on the State of the Art in Coordination Technology*

*Anita Sarma*
Department of Computer Science & Engineering
University of Nebraska, Lincoln
Lincoln, NE 68588 USA
asarma@cse.unl.edu

*David Redmiles, André van der Hoek*
Department of Informatics
University of California, Irvine
Irvine, CA 92697-3440 USA
{redmiles, andre}@ics.uci.edu

Effective coordination is essential for any group work and has been studied widely in the fields of Management Science and Organizational Behavior. A classic definition of coordination is "the management of task dependencies" as proposed by Thompson [16], who distinguished among pooled tasks, where the results of individual actions are simply aggregated; sequential tasks, where the output for one is the input for another; and reciprocal tasks, where work flows back and forth between tasks. More recently, Malone and Crowston [12] developed detailed catalogs of general types of interdependencies that occur across many domains, and particular types of dependencies that occur only in specific domains.

In the software development domain, coordination needs arise because of dependencies among artifacts, which require developers to coordinate each others' changes to produce a coherent whole. Coordination efforts have been shown to constitute a significant percentage of a developer's work time [9, 11]. When developers are distributed across subteams, different buildings, or across countries, there is a manifold increase in the coordination efforts required. It is therefore not surprising that coordination breakdowns frequently occur [3, 9], even though a host of coordination tools (e.g., configuration management systems, bug trackers, workspace awareness tools) are available and even though many development projects follow well-established processes (e.g., Rational Unified Process, agile development). For example, many expertise recommendation systems point to the same expert time and again, without paying attention to the resulting workload. As another example, configuration management systems do not detect and cannot help resolve mutually incompatible changes to different artifacts, a situation that occurs even in the case of rapid change cycles [1]. Similar, and often non-trivial, gaps exist in other technologies.

To provide optimal coordination capabilities of communicating the "right" sets of information, to the "right" sets of stakeholders at appropriate times, different kinds of coordination tools have emerged. The general objective behind these tools is the same: to reduce both the number of occurrences of coordination problems as well as the impact of any occurrences that remain. We define coordination problems here as deviations from the coordination optimum; that is, the hypothetically perfect project in terms of coordination overhead, work integration, and project progress. Whether intentional or accidental, such deviations lead to more effort spent and project delays, as exhibited by the need to address a merge problem caused by overlapping changes, a build failure stemming from

dependency violations, or an integration test failure resulting from a test suite that was developed independently from the code and made inadvertent differing assumptions.

This paper introduces the Coordination Pyramid, a comprehensive framework that provides a novel and efficient way to compare and contrast different classes of coordination tools for collaborative software development. To date, most of these classes of tools have been discussed separately. In carefully placing them in a joint perspective, the Pyramid: (1) illustrates the evolution of coordination tools from minimal infrastructure and an early focus on strict control over explicit coordination activities to the flexible model of empowered self-coordination that underlies modern technology; (2) highlights critical technological and organizational assumptions underlying different coordination tools; and (3) elucidates different aspects of coordination that particular classes of tools support.

The Coordination Pyramid explicitly recognizes several distinct paradigm shifts that have taken place to date, as prompted by technological advancements and changes in organizational and product structure. These paradigm shifts have strongly driven the development of new generations of coordination tools, each enabling new forms of coordination practices to emerge and bringing with it increasingly sophisticated tools through which developers coordinate their day-to-day activities.

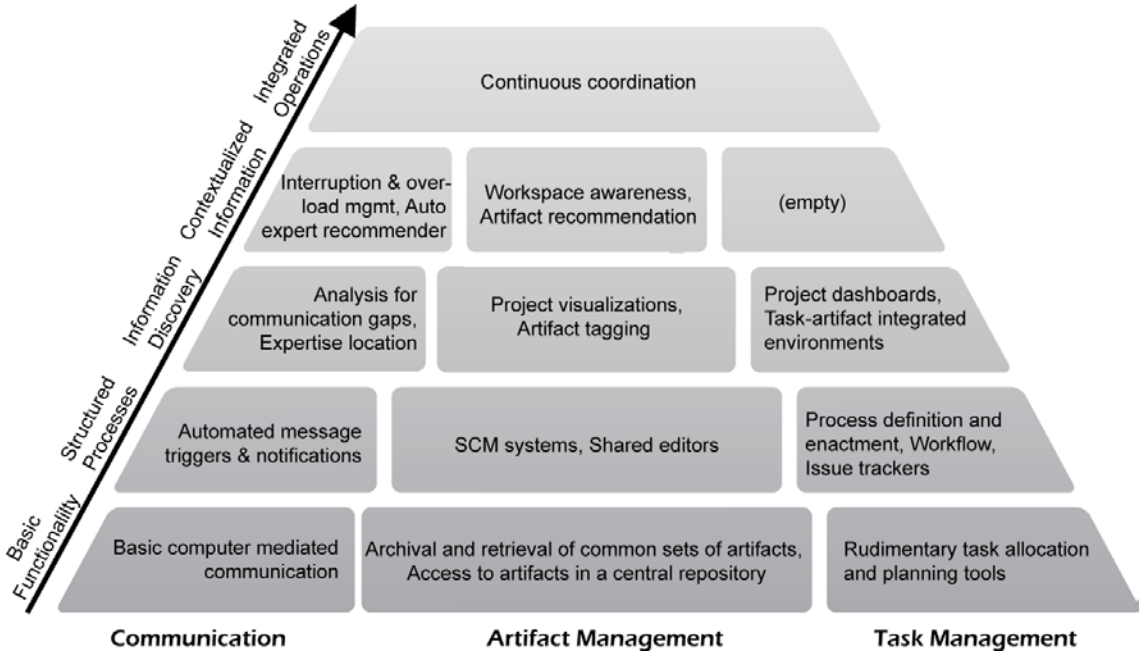


**Figure 1. The Coordination Pyramid.**

## *COORDINATION PYRAMID*

Existing frameworks for categorizing coordination technologies have typically only examined a specific class of coordination technology. For instance, both DeSantis and Gallupe [4] and Grudin [7] classify communication and text editing tools from the perspective of time versus space; Storey et al. [14] and Tam and Greenberg discuss the different coordination dimensions of groupware [15]; and Dewan focuses on the different aspects of conflict management [5]. The Coordination Pyramid takes a different, complimentary,

and unified perspective in relating different classes of coordination tools based on the *underlying paradigm of coordination*, that is, the overarching philosophy and set of rules according to which coordination takes place. Particularly, the Pyramid organizes the paradigm shifts that have historically emerged in a hierarchy of existing and emerging coordination tools.

Looking back over time, four distinct paradigms have come to the forefront and a fifth one is slowly emerging. These five paradigms are represented by the layers of the Coordination Pyramid, as shown in Figure 1. Each layer identifies the kinds of technical capabilities that support a paradigm and is further categorized along three strands. Each strand represents a fundamental aspect of coordination: *communication, artifact management*, and *task management*. These three strands abstract the basic coordination activities in software development, irrespective of different development practices, be it global or agile software development. That is, in any setting, developers need to coordinate their individual access and changes to a common set of interdependent artifacts (middle strand). While some of that can be dealt with through tools, at a broader level they always will need to communicate with each other (left strand), as well as plan and manage their tasks (right strand).

We note that in the remainder of the discussion we primarily focus on tools that support coordination during the implementation phase, as support for coordination in this phase of the life cycle has garnered the greatest amount of attention and made the greatest amount of progress to date. We generally see potential solutions first being explored and adopted in support of the implementation activity, and only then being feathered out to other phases.

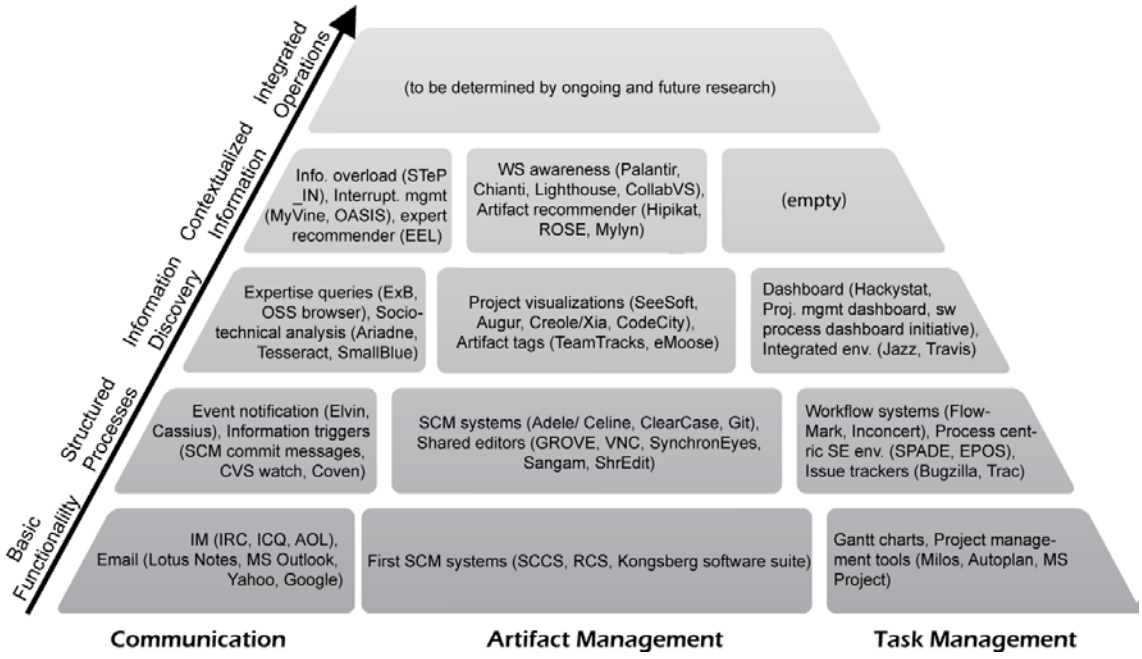


**Figure 2. Representative Examples of Coordination Tools.**

A key consideration in interpreting the Coordination Pyramid is that layers represent increasingly sophisticated support for coordination. That is, the functionality of tools at a given layer either directly builds upon or has been made possible by the capabilities of

technology at the previous layer. Tools at higher layers aim to provide increasingly effective automated assistance for avoiding, detecting, and resolving coordination problems. This is not to say that these tools can completely avoid problems or otherwise always automate their resolution. At certain times, in fact, individuals may be required to expend more effort than they normally would (e.g., adding tags, specifying more precisely which task they are working on). However, in moving up to a higher layer, the aim is for a reduction in the overall organizational effort by the team that is devoted to coordination.

A symbiotic relationship exists between the technical capabilities that comprise a layer and the context in which these capabilities are used. Technical advances enable new forms of coordination, which in turn may lead to new organizational structures. Object-oriented programming and robust build and test systems, for instance, have made possible agile development.

By the same token, new organizational structures demand new forms of coordination, which may require development of new technical capabilities. As an example, geographical separation has prompted the development of video conferencing and distributed synchronization algorithms for configuration management repositories.

It is generally difficult to articulate whether technical advances drive changes in organization structures or vice versa, as they typically evolve in unison. The relationship, however, is evident in the way past paradigms have evolved.

The Pyramid provides a historical perspective on the evolution of coordination paradigms, but we note that the evolution and use of some classes of tools does not necessarily match the historical evolution of the paradigms. In some cases, such as instant messaging, the capability existed long ago (e.g., talk or IRC), but its use as a coordination tool gained popularity only over the past decade because of higher bandwidth availability, larger incidence of distributed development, and technical savvy among users. While instant messaging could be said to fit the Information Discovery paradigm in that it often is used exactly for the purpose of finding out something, we still classify it under the Basic Functionality paradigm as ultimately the tool merely enables communication among developers, but does not itself provide the information that the developers seek.

A deliberate structural feature of the Coordination Pyramid is that its strands slowly but surely blend at higher layers. This blending indicates that advanced coordination tools have begun to integrate aspects of communication, artifact management, and task management in order to make more informed decisions and provide insightful advice regarding potential coordination problems. A side effect of our chosen depiction is that individual cells in the higher layers are smaller in size. This does not represent that problems are simpler or that adoption of coordination tools is easier at higher levels. Rather, it depicts that the integration across strands slowly but surely will lead to an optimum of sorts. Further integration is the key to future advances, which is why we left the top of the Pyramid open – new paradigms are bound to emerge as the field forges ahead.

Figure 2 presents the Coordination Pyramid again, but with each cell containing a small set of representative tools. The sidebar provides a more comprehensive set of examples of coordination technologies in the third and fourth layers of the Pyramid.

***Basic Functionality***

The first layer in the Coordination Pyramid is that of *Basic Functionality*. Technology at this layer focuses on enabling computerized coordination, allowing a team to move from purely manual coordination strategies to strategies that involve automated tools. These tools, however, automate just the minimally needed functionality and each tool tends to focus on one particular aspect of coordination. Developers still must decide when to coordinate, with whom, regarding what, and how – decisions that require time and effort. The effectiveness of this ad-hoc process depends on developers' detailed knowledge of the product, their willingness to proactively share this understanding, and the organizational context in which this sharing takes place.

Canonical examples of tools that fit in this layer are email, discussion boards, shared file systems, first generation configuration management tools[1], project allocation tools, and scheduling tools. The technical capabilities that define this layer were among the first to provide some level of automated support for coordination in teams. Many organizations in its early stages may still find themselves using modernized versions of at least some of these tools (e.g., Google groups instead of Usenet, modern IM tools instead of talk or IRC). Despite such modernizations, the underlying functionality supported by the tools in this layer remains that of primarily asynchronous communication (IM being the exception), basic sharing of artifacts, and standalone project management.

***Structured Processes***

Tools at the second layer of the Pyramid, *Structured Processes*, revolve around automation of the decisions that technology at the Basic Functionality layer leaves open. The focus is on encoding these decisions in well-defined processes that guide developers step-by-step in their engagement with the product and their peers. The processes are typically modeled and enacted in one of two ways: explicitly, through a process or workflow environment, or implicitly, as encoded in the prescribed protocol of user interaction with a particular tool (e.g., copy-edit-merge in the case of configuration management systems, open-resolve-close in the case of issue trackers). Examples of tools that fit in this layer are, among others, shared editors, second-generation configuration management systems, issue trackers, workflow engines, and process modeling environments. The underlying goal of these tools is to enforce a particular protocol for editing, managing, and relating changes to project artifacts.

Most mature organizations will find themselves using tools from this layer (and as necessary the layer below), an outcome that is practically mandated by the Capability Maturity Model. The presence of well-articulated processes allows the organization and its projects to scale considerably. It is interesting to observe that many open source software projects also choose to use suites of tools that reside at this layer. Even their "minimal processes" must exhibit a structure that allows its distributed participants to contribute alike.

As compared to the Basic Functionality layer, average coordination effort per developer is reduced because many rote decisions are now encoded in the processes that are enacted by the tools. On the other hand, explicit process modeling involves significant initial efforts to precisely set up the desired processes. And when a suite of tools is adopted, care must be taken to align their underlying protocols of use. Thus, the initial cost involved in

[1] Early systems like SCCS (Bell labs) and the Kongsberg software suite provided merely central archiving and build abilities. These systems applied concepts of hardware configuration management to software.

adopting this kind of approach to coordination may be high, but it can be gained back in the long term if the processes are carefully chosen and change infrequently.

***Information Discovery (User Initiated)***

Processes are but one component of coordination. It is well-known that informal practices surround the more formal processes established [13] and it is these informal processes that the tools at the *Information Discovery* layer aim to support. Informal coordination takes place at all times, and relies on users gaining information that establishes a context in which they perform their work. Tools at the Information Discovery layer empower users to proactively seek out and assemble the information necessary to build this context. Example tools are software visualization systems, project dashboards, and tool support for locating expertise and gaps in coordination.

As with the Structured Processes layer, the Information Discovery layer represents automation of tasks that otherwise have to be performed manually (e.g., identifying an expert by querying friends and friends of friends, requesting status reports regarding completion of tasks and overall progress). The tools typically rely on information that is already specified by developers as part of other tasks (commit logs, work item status), though some more recent tools rely on developers adding special information that is used later on (e.g., developers annotating code with tags). Other users may directly query the collected information or use visualizations that employ relevant and contextualized formats. For instance, new IDEs contain features that enable developers to leave clearly identifiable tags in the code that other developers can search for and interpret. As another example, visualization tools allow users to investigate development history or patterns to plan their work. Particularly in a distributed setting, where subconscious build-up of context is hindered by physical distances [9], the availability of these kinds of tools is critical.

It is in this layer that the benefits of blending strands become visible. A visualization that shows the traditionally buggy parts of the code is useful for a developer, who can assess the potential for introducing new bugs with their changes, and for project managers, who may or may not put more personnel on those parts of the code. Similarly, social-technical network analysis may reveal coordination gaps, but also identify artifacts that usually are modified together. As a final example, tags that link to discussions on design rationale are useful to communicate information about the state of an artifact, but also provide important information for overall task management.

***Contextualized Information***

Whereas the tools at the Information Discovery layer enable *developers* to be proactive, at the *Contextualized Information* layer the *tools themselves* are proactive. That is, tools at this layer focus on automatically predicting and providing the "right" coordination information to create a context for work and gently, but effectively, guide developers in performing their day-to-day activities. Key properties of these tools are that they attempt to share only relevant information (i.e., they aim to provide the right information to the right person at the right time) and do so in a contextualized and unobtrusive manner. For instance, workspace awareness tools provide information about potential conflicting activities undertaken by other users in parallel by embellishing the development environment [14]. As another example, interruption management systems inform users of the

level of availability of other users; some such systems indeed attempt to automatically detect those levels by closely monitoring communication and work patterns [10].

The crux of this layer lies in the interplay of subtle awareness cues, as presented by the tools, with developers' responses to these cues. The stronger a context for one's work provided by the tools, the stronger the opportunity for developers to self-coordinate with their colleagues to swiftly resolve any emerging coordination problems.

Representative tools at this layer are interruption management technologies, workspace awareness tools, change impact predictors, and recommender systems[2]. At this time, most tools at this layer are still in the exploratory phase. The notion of situational awareness was the driver for much of the early work, work that directly juxtaposed awareness with process-based approaches [14]. More recent work has concentrated on integrating awareness with process, yielding more powerful, scalable, and contextualized solutions. Note that this integration to date has focused on identifying artifacts and experts to facilitate individual task management, but not yet on improving a team's task management as a whole. We are unaware of any task management tools that fit the proactive nature required by level 4, which is why this cell remains empty at this time in the Pyramid.

Whereas the previous layer saw explorative integration of aspects from different strands, this layer is where the blending of strands is virtually required to make progress. Tools must draw upon multiple and diverse sources of information to enable organic forms of (self-)coordination to take place.

### *Emerging Layers*

We deliberately leave the top of the Coordination Pyramid open as we believe new paradigms of coordination will emerge as technology and organizational practices continue to evolve. The ultimate goal is to achieve *continuous coordination*: flexible work practices supported by tools that continuously adapt their behavior and functionality so coordination problems are minimized in number and impact [13]. No longer will developers need to use separate coordination tools or even know about the tools' presence. Their workbench simply provides the necessary information and functionality in a seamless and effective manner, in effect bringing coordination and work together in a single concept. This ideal will certainly not be reached in one paradigm shift, but will require multiple, incremental generations of coordination technology, approaches, and work practices to emerge first – leading to additional layers in the Pyramid.

## *MOVING UP IN THE PYRAMID*

The Coordination Pyramid brings out an interesting evolutionary pattern: paradigm shifts show a distinct interleaving pattern that alternates between enabling new capabilities that generally require significant manual effort and encoding those manual work patterns into automated tool capabilities. Layer 1 enables basic forms of coordination, but any complicated processes require much manual involvement. Layer 2 brings automated, highly-structured processes, but results in approaches that are too rigid. Layer 3 overcomes this

[2] Recommender systems at this layer automatically provide suggestions for experts regarding a certain piece of code or automatically adjust task context, instead of users having to initiate and assist in such activities.

rigidity by enabling developers to proactively look for information, but requires significant context switching and up-front knowledge of what to look for. Layer 4 addresses this problem by making the tools themselves more proactive and integrating them carefully in the development environment. We fully expect this pattern to continue in future.

In many situations, organizations will incorporate a mix of tools residing at different layers. For instance, on finding a potential conflict using a workspace awareness tool (Layer 4), a developer may contact their colleague over IM (Layer 1) and, upon realizing that both of them are working on different parts of the file, decide to make their modifications in sequence, with the first developer using tags (Layer 3) to inform the other developer of the nature and impact of their changes. This is not surprising as tools tend to be picked individually. In an attempt to move up in the Pyramid, however, it should be kept in mind that simply picking up the next set of coordination tools is not a guarantee for successful adoption of its envisioned associated work practices. A certain amount of organizational and individual readiness is required. For example, in our years of collaborating with industry we met managers who actively promote parallel changes to the same artifacts. Likewise, we know managers who insist on using locks to prohibit any form of parallel work. Infusing the same workspace awareness technology in these managers' respective groups is bound to lead to very different outcomes.

Of course, organizational readiness is not just a managerial issue. Successful insertion of particular coordination technologies requires individual developers to be willing to share information about their development practices or to, at times, expend extra effort to benefit the team. For example, some workspace awareness tools rely on non-empty commit comments [6]; others on developers adding themselves to and participating in a team portal; and yet others on setting one's activity level to reflect current work [2]. Organizations must provide adequate training and education, especially if they desire to use the more sophisticated tools in the upper layers of the Coordination Pyramid for overall team benefits, as these tools often require changes in work practices.

We note that not all changes to work practices are driven top-down. Sometimes, grassroots adoption of higher-level technology led to organizational change. IM is the prime example, with developers in some cases having established informal conventions that are remarkably effective [8] and changed the organization from within.

Much success in adopting more sophisticated tools in the Coordination Pyramid also depends on generating trust and respecting privacy. Since tools at higher layers typically rely on sharing detailed information about individuals, their tasks, and their progress with those tasks, it is possible and sometimes tempting to "misuse" this information. For instance, managers may exert too much control over detailed activities or, worse, create a situation where developers compete with each other to produce the "best" public image of themselves (e.g., most lines of code written, most work items processed), which of course does not necessarily coincide with applying best coordination practices). Such misuse, once known, will instantly become an impediment to the effective use of the tools. In fact, even a perception of potential misuse may already lead to distrust and unanticipated and counterproductive practices. Care must be taken to establish an increasingly open and honest work environment with corresponding organizational policies as an organization moves up in the Pyramid.

## *CONCLUSIONS*

The full cost of coordination problems has not been quantified to date, and may well be impossible to precisely determine. However, rework, unnecessary work, and missed opportunities are clearly part of a developer's everyday experience. Even when a problem is considered simply a nuisance, as when an expert who is recommended over and over again chooses to ignore questions or to only answer select developers, it generally involves invisible consequences that impact the overall effectiveness of the collaborative effort. A developer looking for an answer and not receiving one may as a result interrupt multiple other developers, or spend significant amounts of time and effort trying to solve the problem themselves, time and effort that could have easily been saved. Numerous, often crucial coordination problems have been documented, with severe time delays, serious expenses in developer effort, critical reductions in code quality, and even failed projects as a result [3, 9].

Our Coordination Pyramid helps organizations and individuals better understand desired coordination practices and match these to available tools and technologies. It furthermore charts a roadmap towards improving an organization's coordination practices, by enabling organizations to locate what coordination paradigms they presently follow, whether they might want to adopt other paradigms, and, if so, what some necessary conditions are for making such transitions. Finally, the Pyramid highlights the necessity of the informal practices surrounding the more formal tools and processes that one can institute: effective coordination is always a matter of providing the right infrastructure, yet allowing developers to compensate for shortcomings in the tools by establishing individual strategies of self-coordination.

Our classification of existing coordination tools also helps provide inspiration and guidance into future research. By charting how technologies have evolved, i.e., how they have matured and expanded from cell to cell and layer to layer over time, one can anticipate next steps – attempting to increase coordination support by moving up in the Pyramid and blending strands in the process. With the increasing pressures of global software development, ever-increasing size and complexity of software systems, and never-ending advances in technology, new coordination needs will always arise. The Coordination Pyramid charts the coordination paradigms through which these needs were addressed to date, and provides the impetus for continued development towards future paradigms that address upcoming challenges and opportunities.

## *REFERENCES*


[1] J. Biehl, *et al.*, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," in *Human Factors in Computing Systems* 2007, pp. 1313-1322.

[2] L.-T. Cheng, *et al.* (2003, December) Building Collaboration into IDEs. Edit -> Compile -> Run -> Debug -> Collaborate? *ACM Queue*. pp. 40-50.

[3] C. R. B. de Souza and D. Redmiles, "An Empirical Study of Software Developers' Management of Dependencies and Changes," in *Thirteeth International Conference on Software Engineering*, 2008, pp. 241-250.

[4] G. DeSanctis and R. B. Gallupe, "A Foundation for the Study of Group Decision Support Systems," *Management Science*, vol. 33, 1987, pp. 589-609.
[5] P. Dewan, "Dimensions of Tools for Detecting Software Conflicts," in *International Workshop on Recommendation Systems for Software Engineering*, 2008, pp. 21-25.
[6] G. Fitzpatrick, *et al.*, "Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections," in *ACM conference on Computer Supported Cooperative Work*, 2002, pp. 447-474.
[7] J. Grudin, "CSCW: History and Focus," *IEEE Computer*, vol. 27, May 1994, pp. 19-27.
[8] J. Herbsleb, *et al.*, "Introducing Instant Messaging and Chat in the Workplace," in *SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves* 2002, pp. 171-178.
[9] J. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally-Distributed Software Development," *IEEE Transactions on Software Engineering*, vol. 29, 2003, pp. 1-14.
[10] S. T. Iqbal and B. P. Bailey, "Effects of Intelligent Notification Management on Users and their Tasks," in *Twenty-sixth SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 93-102.
[11] R. Kraut, *et al.*, "Patterns of contact and communication in scientific research collaboration," in *Conference on Computer-Supported Cooperative Work*, 1988, pp. 1-12.
[12] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Computing Surveys*, vol. 26, 1994, pp. 87-119.
[13] D. Redmiles, *et al.*, "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects," *Withschaftsinformatik*, vol. 49, 2007, pp. S28–S38.
[14] M.-A. Storey, *et al.*, "On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework," in *ACM Symposium on Software Visualization*, 2005, pp. 193-202
[15] J. Tam and S. Greenberg, "A Framework for Asynchronous Change Awareness in Collaborative Documents and Workspaces," *International Journal of Human-Computer Studies*, vol. 64, 2006, pp. 583-598.
[16] J. Thompson, *Organizations in Action: Social Science Bases of Administrative Theory*, 1st ed. New York: McGraw-Hill, 2003.p. 192.