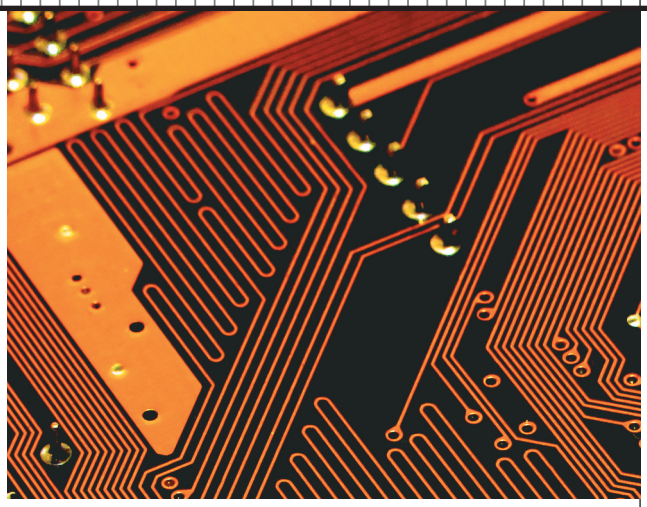
Categorizing the Spectrum of Coordination Technology



Anita Sarma, *University of Nebraska, Lincoln*
David Redmiles and André van der Hoek, *University of California, Irvine*

Most frameworks that categorize technology for collaborative software development look at only one aspect of coordination support. The Coordination Pyramid classifies technologies according to multiple coordination paradigms, offering a unified, complementary perspective and a structure for evaluating emerging technology.

Effective coordination, central to any group work, is essentially the management of task dependencies,[1,2] with tasks being pooled, sequential, or reciprocal. In software development, the lack of coordination can result in project delays and increased effort. Even coordinating a single colocated project takes a significant percentage of development work because the team must manage multiple dependencies for each artifact. Distributing developers across subteams, buildings, or countries greatly increases coordination efforts, and coordination breaks down, despite a plethora of tools and adherence to well-established processes: recommendation systems point to the same expert, configuration management systems fail to detect incompatible changes to different artifacts, and so on.

The coordination *optimum* is to communicate the right information to the right stakeholders at the right time. It is the hypothetically perfect project in terms of coordination overhead, work integration, and overall progress. Coordination *problems* are any intentional or accidental deviations from that optimum. Thus, the objective of coordination tools is twofold: to decrease the occurrences of coordination problems and to mitigate the impact of problems that do occur. Such problems might be overlapping changes that cause a merge conflict, dependency violations that cause a build failure, or the development of a test suite independently of the code, which can cause an integration failure.

Coordination technology consists of tools that support collaborative software development, and an overwhelming number of such tools are available. Frameworks are useful in comparing and contrasting coordination technology according to a specific paradigm, such as structured processes or information discovery. However, traditional frameworks address only a single paradigm, which offers a limited perspective.

To broaden that perspective, we have created the *Coordination Pyramid*, which relates five distinct coordination paradigms and classifies coordination technologies according to which paradigm they primarily support. The Coordination Pyramid helps organizations assess and articulate their coordination needs and find the toolset that matches those needs. Because it views technology in a hierarchy of coordination paradigms, the pyramid helps in documenting and framing trends and in identifying promising new research directions and application areas.

The Coordination Pyramid also explicitly recognizes the technological advancements and changes in organiza-

0018-9162/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

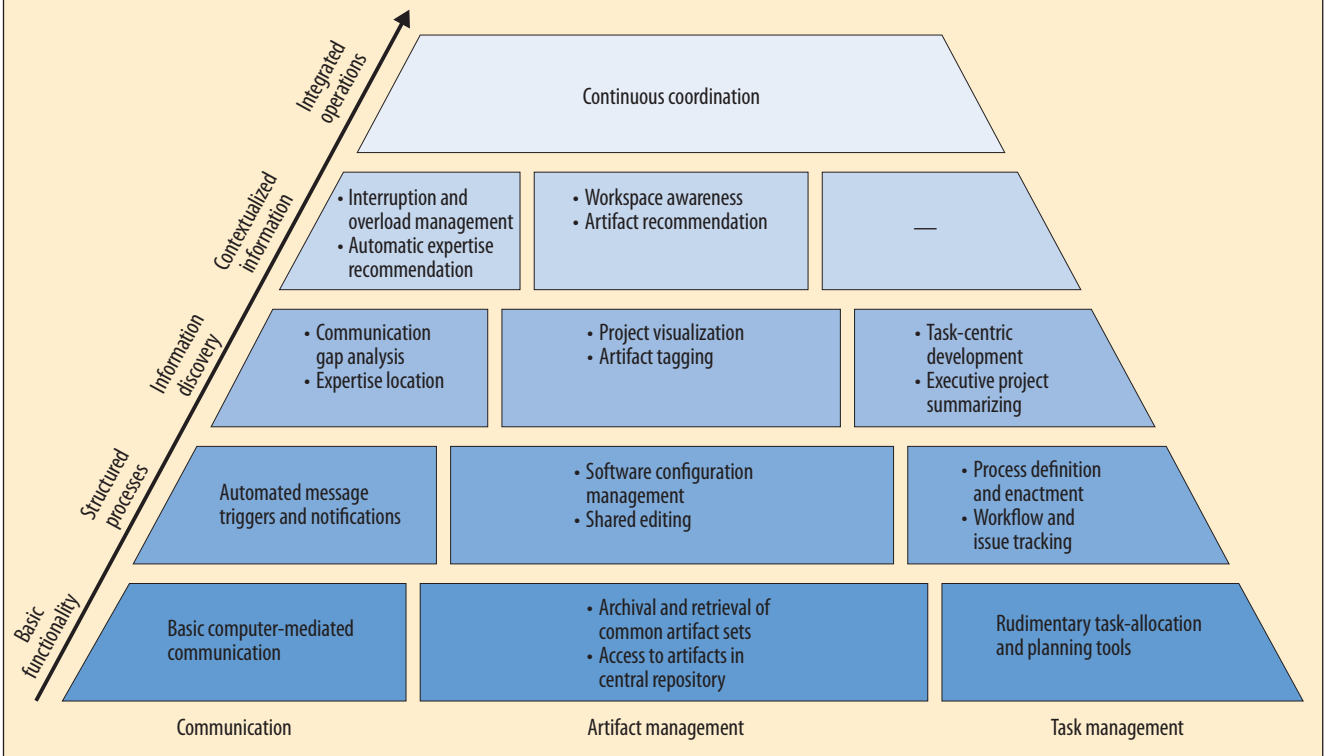


Figure 1. The Coordination Pyramid. Unlike other frameworks that classify coordination technology, the Coordination Pyramid takes a complementary perspective, organizing technology according to five coordination paradigms. Each layer identifies the kinds of technical capabilities that support a paradigm and is cross-categorized along three coordination aspects: communication, artifact management, and task management.

tional and product structure that prompt paradigm shifts. Typically, these shifts drive new generations of coordination technology, enabling more sophisticated tools and practices.

A PROGRESSIVE FRAMEWORK

Most frameworks for categorizing coordination technologies examine only a specific class, such as groupware or conflict management, or view tools from only one perspective, such as time versus space.

As Figure 1 shows, the Coordination Pyramid takes a complementary, unified perspective in relating tool classes. It organizes existing and emerging coordination technology hierarchically according to the underlying coordination paradigm—the overarching philosophy and rules that govern coordination. Over time, four paradigm shifts have emerged, and a fifth one is nascent. Each layer in the pyramid identifies the technical capabilities that support a paradigm, which are cross-categorized along three coordination aspects: communication, artifact management, and task management. These three aspects, or strands, abstract the basic coordination activities in software development, irrespective of development practice. This additional categorization recognizes that all software developers must coordinate individual access and changes to a common set of interdependent artifacts (artifact management). Although tools can handle some of that coordination work, developers will always need to communicate with one another (communication), as well as plan and manage their tasks (task management). As the figure implies, our focus is primarily on tools that support coordination during the implementation phase—coordination support during this phase has received the most attention and is the most mature to date.

The three strands blend at the higher layers. This blending reflects the reality that coordination tools have begun to integrate these three aspects to provide the insight into potential coordination problems that will lead to more well-rounded solutions. Integration at the higher layers is evident in the shrinking cells. The problems are no less complex, nor is tool adoption easier; rather, the cells' diminishing size implies that, as the strands increasingly integrate, coordination grows closer to the optimum. Increased integration is also key to future advances, which is why the pyramid is open—new paradigms will emerge as coordination support matures.

PARADIGM-BASED CLASSIFICATION

Table 1 gives a sample of the coordination technology that corresponds to each pyramid cell, and the "Sampling

Table 1. Representative tools corresponding to cells in the Coordination Pyramid.

Layer aspect	Communication	Artifact management	Task management
Basic functionality	IM: IRC, ICQ, AOL E-mail: LotusNotes, Outlook, Yahoo, Google	SCCS, RCS, Kongsberg software suite	Milos, Autoplan, MS Project
Structured processes	Event notification: Elvin, Cassius Information triggers: SCM commit messages, CVS watch, Coven	SCM systems: Adele/Celine, Clear Case, Git Shared editors: Grove, VNC, SynchronEyes, Sangam, ShrEdit	Workflow systems: FlowMark, Inconcert Process environments: SPADE, Epos Issue trackers: Bugzilla, Trac
Information discovery	Expertise queries: Expert browser, OSS browser Sociotechnical analysis: Ariadne, Tesseract, SmallBlue	Project visualization: SeeSoft, Augur, Creole/Xia, CodeCity Artifact tags: TeamTracks, eMoose	Dashboards: Hackystat, project management dashboard, software process dashboard initiative Integrated environments: Jazz, Travis
Contextualized information	Information overload: Step_In Interruption management: MyVine, Oasis Expert recommender: EEL	Workspace awareness: Palantir, Chianti, Lighthouse, CollabVS Artifact recommender: Hipikat, ROSE, Mylyn	—

*CVS: Concurrent versions system; EEL: emergent expertise locator; ICQ: Internet chat query; IRC: Internet relay chat; IM: instant messaging; RCS: revision control system; ROSE: reengineering of software evolution; SCCS: source-code control system; SCM: software configuration management; SPADE: Software Process Analysis Design and Enactment; VNC: virtual network computing

of Advanced Coordination Technologies" sidebar describes tools in the upper layers in more depth. Because layers move from basic functionality to increasingly more sophisticated coordination support, tools at one layer either directly build on or have been enabled by the technology in the preceding layer. Tools at higher layers often use automation to avoid, detect, or resolve coordination problems that preceding layers did not address. Of course, tools cannot avoid problems entirely or always automate their resolution. Indeed, in some cases, manual effort increases, such as when an individual must add tags or specify a current task in more detail. However, the higher the layer, the less time the team should be spending on coordination overall.

Basic functionality

Technology at this layer focuses on moving a team from purely manual coordination to strategies that minimally involve automated tools. These tools provide bare-bones functionality, generally with each tool focusing on a single coordination aspect. Developers must still decide when to coordinate, with whom, about what, and how. The remaining time and effort depend on the developers' product knowledge, their willingness to share that knowledge, and the organizational context.

Tools that fit this layer are e-mail, discussion boards, shared file systems and tools for first-generation configuration management, project allocation, and scheduling. Some of these tools, among the first to support team coordination, have modern versions (Google groups vs. Usenet), but the underlying functionality remains primarily asynchronous communication, basic artifact sharing, and stand-alone project management.

Structured processes

Tools in this layer revolve around automating the decisions that tools in the basic functionality layer left open. The focus is on encoding these decisions in well-defined coordination processes that are typically modeled and enacted explicitly through a workflow environment or implicitly as part of a user interaction protocol for a particular technology. The latter form could be a copy-edit-merge in configuration management systems or an open-resolve-close in an issue tracker. Some tools that fit this layer are shared editors, second-generation configuration management systems, issue trackers, workflow engines, and process-modeling environments. The tools' underlying goal is to enforce a particular protocol for editing, managing, and relating changes to project artifacts.

Most mature organizations will use tools from this layer because of their desire to conform to the Capability Maturity Model. Well-articulated processes also make it easier to scale an organization and its projects. Many open source software projects also use suites of tools that reside at this layer. Even the minimal processes espoused by the open source community must have enough coordination structure to allow operation in a distributed setting.

Relative to the basic functionality layer, tools at this layer reduce a developer's coordination effort because many rote decisions are now encoded in the processes that the tools enact. On the other hand, it takes time to set up the desired process, and adopting a tool suite requires carefully aligning protocols for its use. Thus, the cost of technology in this layer might be initially high, but an organization can recoup that cost by choosing its processes carefully and changing them infrequently.

SAMPLING OF ADVANCED COORDINATION TECHNOLOGIES

The tools in the Coordination Pyramid's basic functionality and structured processes layers are well established. Except for Mylyn, Hackystat, and Jazz, tools in the top two layers are research prototypes, so their functions are less well known. Table A describes tools in the information discovery layer. Table B describes tools in the contextualized information layer.

Table A. Sampling of tools in the information discovery layer.

Tool	Description	Source
Communication		
Expertise Browser	Stand-alone interactive expertise recommender that provides a list of experienced developers once a user selects a code artifact	A. Mockus and J. Herbsleb, "Expertise Browser: A Quantitative Approach to Identifying Expertise," *Proc. 2002 Int'l Conf. Software Eng.* (ICSE 02), ACM Press, 2002, pp. 503-512.
Tesseract	Interactive project exploration environment that visualizes entity relationships among code, developers, bugs, and communication records	A. Sarma et al., "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," *Proc. Int'l Conf. Software Eng.* (ICSE 09), ACM Press, 2009, pp. 23-33.
Artifact management		
SeeSoft	Stand-alone tool for visualizing aspects of the evolution of a source code base	T. Ball and S.G. Eick, "Software Visualization in the Large," *Computer*, vol. 29, no. 4, 1996, pp. 33-43.
Creole	Stand-alone tools for visualizing change frequencies and dependencies among source code modules	R. Lintern et al., "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse," *Proc. 2003 ACM Symp. Software Visualization* (SoftVis 03), ACM Press, 2003, pp. 47–56.
CodeCity	Interactive 3D-visualization tool that uses a city metaphor to depict object-oriented software systems; classes are "buildings" and packages are "districts."	R. Wettel and M. Lanza, "CodeCity: 3D Visualization of Large-Scale Software," *Companion of the 30th Int'l Conf. Software Eng.* (ICSE 08), ACM Press, 2008, pp. 921-922.
Team Tracks	Recommender system that analyzes file and class browsing activities to identify artifact visiting patterns and related task context	R. DeLine et al., "Towards Understanding Programs through Wear-Based Filtering," *Proc. 2005 ACM Symp. Software Visualization* (SoftVis 05), ACM Press, 2005, pp. 183-192.
Task management		
Jazz	IDE features that facilitate collaboration via integrated planning, tracking of developer effort, project dashboards, reports, and process support	http://www-306.ibm.com/software/rational/jazz
Hackystat	Open source framework for collecting, analyzing, visualizing, and interpreting software development process and product data, operating through embedded sensors in development tools with associated Web-based queries	P.M. Johnson and S. Zhang, "We Need More Coverage, Stat! Classroom Experiences with the Software ICU," *Proc. 3rd Int'l Symp. Empirical Software Eng. and Measurement* (ESEM 09), IEEE Press, 2009, pp. 168-178.

Table B. Sampling of tools in the contextualized information layer (communication and artifact management).

Tool	Description	Source
Communication		
Oasis	Interruption management system that defers notifications until users performing interactive tasks reach a breakpoint	S.T. Iqbal and B.P. Bailey, "Effects of Intelligent Notification Management on Users and their Tasks," *Proc. 26th Ann. SIGCHI Conf. Human Factors in Computing Systems* (CHI 08), ACM Press, 2008, pp. 93-102.
Step_In	Expertise recommendation framework to guide tool design; considers information overload, interruption management, and social network benefits	Y. Ye et al., "A Socio-Technical Framework for Supporting Programmers," *Proc. 6th Joint Meeting European Software Eng. Conf. and the ACM SIGSOFT Int'l Symp. Foundations Software Eng.* (ESEC-FSE 07), ACM Press, 2007, pp. 351-360.
EEL	Expertise recommender that uses emergent team information and artifact structure to propose experts as a user works on a task	S. Minto and G.C. Murphy, "Recommending Emergent Teams," *Proc. 4th Int'l Workshop Mining Software Repositories* (MSR 07), IEEE Press, 2007, p. 5.
Palantír	Eclipse extension that supports early detection of emerging conflicts through peripheral workspace awareness	A. Sarma et al., "Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management," *Proc. 16th ACM SIGSOFT Int'l Symp. Foundations Software Eng.* (SIGSOFT 08/FSE-16), ACM Press, 2008, pp. 113-123.
Artifact management		
Chianti	Testing tool integrated with the development environment that identifies test cases needing regeneration because of local changes	X. Ren et al., "Chianti: A Tool for Change Impact Analysis of Java Programs," *Proc. 19th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 04), ACM Press, 2004, pp. 432-448.
Mylyn	Task-centric, Eclipse-based environment that leverages information to reduce context switching and provide developers with just the artifacts and information necessary for their current code modification task	M. Kersten and G.C. Murphy, "Using Task Context to Improve Programmer Productivity," *Proc. 14th ACM SIGSOFT Int'l Symp. Foundations Software Eng.* (SIGSOFT 06/FSE-14), ACM Press, 2006, pp. 1-11.
CollabVS	Visual Studio extension that informs users of emerging direct and indirect conflicts via a "conflict inbox" and enables IM conversation	P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development," *Proc. 10th European Conf. Computer-Supported Cooperative Work* (ECSCW 07), Springer, 2007, pp. 159-178.

Information discovery

Processes are but one coordination component. The informal practices that surround formal processes also need support[3]—the primary aim of tools in the information discovery layer. Because informal coordination requires users to gain information that establishes a context for their work, tools in this layer empower users to proactively seek and assemble information to build this context. Some examples are software visualization systems, project dashboards, and tools that help locate coordination expertise and identify process gaps.

Technology in the information discovery layer aims to automate tasks such as identifying an expert and requesting status reports on task completion and overall progress. The tools typically rely on information that developers already provide in other tasks (commit logs, work item status), although some of the more recent tools require developers to add information for later use, such as annotating code with tags.

So far, the focus has been on making individual task management easier. Efforts have not addressed how to improve task management for the team as a whole.

Other users can directly query the collected information or use visualizations with relevant and contextualized formats. New interactive development environments let developers leave clearly identifiable tags in the code that other developers can search for and interpret. Visualization tools let users investigate development history or patterns to plan their work. Such tools are critical to supporting coordination in a distributed setting, where distance hinders the essentially subconscious buildup of context.[4]

In this layer, the benefits of blending communication, artifact management, and task management become clear. A visualization that shows where code is traditionally buggy is useful to both developers and project managers. A developer can assess the possibility that a new change will introduce bugs; a project manager can decide to put more personnel on those parts. A sociotechnical network analysis might not only reveal coordination gaps[5] but also identify artifacts that developers usually modify together. Finally, tags that link to discussions on design rationale not only communicate information about an artifact's state but also provide important information for overall task management.[6]

Contextualized information

Tools in the information discovery layer enable developers to be proactive; but tools at this layer are *themselves* proactive because they automatically provide the right coordination information to create a work context and gently guide developers as they perform their daily activities. The idea is to provide the right information to the right person at the right time, unobtrusively sharing only what is relevant. The tools in this layer present subtle awareness cues to which the developers respond. The crux of this layer is in the interplay between those cues and responses: the stronger the work context the tools provide, the greater the opportunity for developers to proactively and swiftly resolve any emerging coordination problems.

This layer includes tools for interruption management and workspace awareness, change impact predictors, and expertise recommendation systems. Workspace awareness tools provide information about potentially conflicting activities that users are performing in parallel to embellish the development environment.[7,8] Interruption management systems inform a user of other users' availability; some such systems even attempt to detect availability automatically by closely monitoring users' communication and work patterns.[9]

Most tools in this layer are still in the exploratory phase. The notion of situational awareness drove much of the early work, which directly juxtaposed awareness with process-based approaches.[7,8] More recent work has concentrated on integrating awareness with process, yielding more powerful, scalable, and contextualized solutions. To date, integration work has focused on identifying artifacts and experts to make it easier for individuals to manage their tasks, but it has not yet addressed how to improve task management for the team as a whole. No task management tools that we know of fit the proactive nature that this layer requires, which is why Table 1 has no entry for this layer and strand.

In the previous layer, the integration of coordination aspects was explorative; in the contextualized information layer, such integration is required. Tools must draw on multiple and diverse information sources to enable organic forms of self-coordination.

Emerging layers

The pyramid's top layer is empty and open, which signifies our belief that paradigms for coordination support will continue to evolve with new technology and organizational practices. The ultimate goal is continuous coordination—flexible work practices supported by tools that continuously adapt their behavior and functionality so as to minimize the number and impact of coordination problems.[3] In this ideal, developers will not even realize that separate coordination tools exist because the development workbench will seamlessly merge coordination and work. Of course, coordination technology will not reach this vision in one paradigm shift; attaining it will require multiple, incremental generations of coordination technology, approaches, and work practices.

CAPABILITIES AND CONTEXT

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 can lead to new organizational structures. Object-oriented programming and robust build and test systems, for example, have enabled agile development. Likewise, new organizational structures demand new forms of coordination, which in turn require new technical capabilities. Geographical separation, for example, has prompted the development of videoconferencing, and distributed synchronization algorithms have fueled configuration management repositories.

It is not clear if technical advances drive changes in organization structures or vice versa, since the two tend to evolve in unison. However, some kind of interrelationship is evident from the way existing paradigms have evolved.

TOOL USE AND WORK PRACTICES

The Coordination Pyramid shows that paradigm shifts alternate between enabling new capabilities that require significant manual effort and encoding manual work patterns into automated tool capabilities. As its name implies, the basic functionality layer enables only basic forms of coordination; any complicated processes still require considerable manual work. The structured processes layer brings automated, highly structured processes, but the resulting approaches are rigid. The next layer overcomes this rigidity by enabling developers to proactively look for information, but it requires significant context switching and knowledge of what to look for. Contextualized information addresses this problem by making the tools themselves more proactive and by carefully integrating the tools into the development environment. We fully expect this interleaving pattern to continue for the reasons described in the "Capabilities and Context" sidebar.

Organizations often mix tools from different layers. On finding a potential conflict using a workspace awareness tool (contextualized information layer), a developer might contact a colleague over IM (basic functionality layer). After Developer A realizes that Developer B is working on different parts of the file, the two might decide to modify their parts in sequence, with Developer A using tags (information discovery layer) to inform Developer B of a change's nature and impact. This mixed-layer tool use is not surprising, since most organizations acquire tools one at a time.

When moving up the Coordination Pyramid, however, organizations must not assume that a suite of coordination tools will automatically ensure the adoption of sophisticated work practices. For example, a company might adopt some sophisticated recommendation tool, but developers could still refuse to divulge the work details that the tool needs for a recommendation, perhaps fearing competition from colleagues. Evidence is emerging that tools at higher layers offer benefits: workspace awareness tools save time and decrease the number of conflicts,[7,8] recommender systems boost productivity,[10] and interruption management systems shorten the time to resume activities and decrease the team's frustration.[9] But reaping these benefits requires a certain amount of organizational and individual readiness. In our years of industry collaboration, we have met managers who actively promote parallel changes to the same artifacts as well as managers who insist on using locks to prohibit parallel work. If these managers adopt the same workspace awareness technology, the outcomes are bound to be very different.

Organizational readiness is not just a managerial issue. Successfully inserting coordination technologies requires that individual developers be willing to share information about their development practices and at times spend extra effort to benefit the team. For example, some workspace awareness tools rely on nonempty commit comments;[11] others expect developers to add themselves to a team portal and actively participate in it; still others mandate that individuals set their activity level to reflect current work.[12] Training and education are essential, particularly if an organization wants to use the more sophisticated tools in the pyramid's upper layers, since these tools often necessitate work practice changes.

Sometimes the grass-roots adoption of higher-level technology leads to organizational change. IM is the prime example: in some cases, developers have established remarkably effective informal conventions that have changed the organization from within.

Successfully adopting sophisticated tools also requires generating trust and respecting privacy. Because tools at the higher layers typically rely on sharing detailed information about individuals, their tasks, and their task progress, it can be tempting to misuse this information. Managers might exert too much control over detailed activities or, worse, create a competitive situation in which developers feel pressured to produce the best public image by writing the most lines of code or processing the most work items. This competitive atmosphere is at odds with effective tool use. Indeed, even the perception of potential misuse can lead to distrust and unanticipated counterproductive practices. In moving up the Coordination Pyramid, an organization must carefully establish an increasingly open and honest work environment with corresponding organizational policies.

No one has yet precisely quantified the cost of coordination problems, and such an effort might very well be impossible. Most organizations accept that rework, unnecessary work, and missed opportunities are all too common in development projects. Even when a problem is simply

a nuisance, such as when a repeatedly recommended expert chooses to ignore questions or to answer only select developers, the consequences can undermine a collaborative effort. A developer looking for an answer and not receiving one might interrupt other developers or spend considerable time trying to solve the problem without help. The literature is rife with accounts of crucial coordination problems that have resulted in severe time delays, serious developer expenses, decreased quality of critical code, and even failed projects.

Our Coordination Pyramid can serve as a road map for improving an organization's coordination practices. An organization can more easily locate the coordination paradigms it currently follows, review the tools in other paradigms, and gain the insight into the conditions for transitioning to a new paradigm. The pyramid also highlights the necessity of the informal practices surrounding the more formal tools and processes that an organization can institute. We believe that effective coordination is a matter of providing the right infrastructure yet allowing developers to compensate for tool shortcomings by establishing individual strategies for optimum coordination.

Our coordination tool classification can also help inspire and guide future research. Charting how technologies have evolved—matured and expanded from cell to cell and layer to layer—makes it easier to anticipate next steps. An organization might feel more confident about increasing coordination support by moving up the pyramid and in so doing blend the three key coordination aspects. With the increasing pressures of global software development, burgeoning size and complexity of software systems, and never-ending technological advances, new coordination needs will always arise. The Coordination Pyramid provides the impetus for reaching toward future paradigms that address the next generation of coordination challenges and opportunities.

References

1. J. Thompson, *Organizations in Action: Social Science Bases of Administrative Theory*, McGraw-Hill, 2003.
2. C.R.B. de Souza and D. Redmiles, "An Empirical Study of Software Developers' Management of Dependencies and Changes," *Proc. 30th Int'l Conf. Software Eng.* (ICSE 08), IEEE CS Press, 2008, pp. 241-250.
3. D. Redmiles et al., "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects," *Wirtschaftsinformatik*, vol. 49, 2007, pp. S28-S38.
4. J. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Trans. Software Eng.*, vol. 29, 2003, pp. 1-14.
5. M. Cataldo et al., "Software Dependencies, Work Dependencies and Their Impact on Failures," *IEEE Trans. Software Eng.*, vol. 99, 2009, pp. 864-878.
6. M.-A. Storey et al., "Shared Waypoints and Social Tagging to Support Collaboration in Software Development," *Proc. 20th Anniversary Conf. Computer-Supported Cooperative Work* (CSCW 06), ACM Press, 2006, pp. 195-198.
7. P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development," *Proc. 10th European Conf. Computer-Supported Cooperative Work* (ECSCW 07), Springer, 2007, pp. 159-178.
8. A. Sarma et al., "Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management," *Proc. 16th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.* (SIGSOFT 08/FSE-16), ACM Press, 2008, pp. 113-123.
9. S.T. Iqbal and B.P. Bailey, "Effects of Intelligent Notification Management on Users and Their Tasks," *Proc. 26th Ann. SIGCHI Conf. Human Factors in Computing Systems* (CHI 08), ACM Press, 2008, pp. 93-102.
10. M. Kersten and G.C. Murphy, "Using Task Context to Improve Programmer Productivity," *Proc. 14th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.* (FSE 06), ACM Press, 2006, pp. 1-11.
11. G. Fitzpatrick et al., "CVS Integration with Notification and Chat: Lightweight Software Team Collaboration," *Proc. 20th Anniversary Conf. Computer-Supported Cooperative Work* (CSCW 06), ACM Press, 2006, pp. 49-58.
12. L.-T. Cheng et al., "Building Collaboration into IDEs," *ACM Queue*, vol. 1, no. 9, 2003, pp. 40-50.

Anita Sarma *is an assistant professor in the Department of Computer Science and Engineering at the University of Nebraska, Lincoln. Her research interests include understanding and facilitating coordination in distributed work. Sarma received a PhD in information and computer science from the University of California, Irvine. She is a member of IEEE and the ACM. Contact her at asarma@cse.unl.edu.*

David Redmiles *is an associate professor and chair of the Department of Informatics in the Donald Bren School of Information and Computer Sciences at the University of California, Irvine. His research interests include software engineering, human-computer interaction, computer-supported cooperative work, and most recently collaborative software engineering. Redmiles received a PhD in computer science from the University of Colorado. He is a member of IEEE, the ACM, and the Association for the Advancement of Artificial Intelligence. Contact him at redmiles@ics.uci.edu.*

André van der Hoek *is a professor in the Department of Informatics in the Donald Bren School of Information and Computer Sciences and a faculty member of the Institute for Software Research, both at the University of California, Irvine. His research focuses on understanding and advancing the role of design, coordination, and education in software engineering. Van der Hoek received a PhD in computer science from the University of Colorado. He is a member of IEEE and the ACM. Contact him at andre@ics.uci.edu.*