Coordination Technologies

Anita Sarma

Abstract Coordination technologies improve the ability of individuals and groups to coordinate their efforts in the context of the broader, overall goal of completing one or more software development projects. Different coordination technologies have emerged over time with the objective of reducing both the number of occurrences of coordination problems as well as the impact of any occurrences that remain. This chapter introduces the Coordination Pyramid, a framework that provides an overarching perspective on the state of the art in coordination technology. The Coordination Pyramid explicitly recognizes several 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 technology, each enabling new forms of coordination practices to emerge and bringing with it increasingly effective tools through which developers coordinate their day-to-day activities.

1 Introduction

Software development environments used to build today's software-intensive systems routinely require hundreds of developers to coordinate their work [18, 22, 48]. For example, Windows Vista, comprising of over 60MLOC, was developed by over 3000 developers, distributed across 21 teams and 3 continents [11]. It is rare for a project to be completely modularized, and the dependencies across software artifacts create complex interrelationships among developers and tasks. These dependencies are called socio-technical dependencies [19] and create the need for coordination among development tasks and developers. An example of evolving task dependencies across 13 teams in IBM Jazz [40] is shown in Fig. 1, where lines indicate a task (work item) that has been sent to another team. Note that,

A. Sarma
Oregon State University, Corvallis, OR, USA
e-mail: anita.sarma@oregonstate.edu

S. Cha et al. (eds.), *Handbook of Software Engineering*,
https://doi.org/10.1007/978-3-030-00262-6_10

Fig. 1 The evolution of task dependencies in the IBM Jazz project. Each pane shows one 3-month period. Teams become increasingly interdependent because of shared tasks

as time progresses, the teams become increasingly entwined, leading to increased coordination costs across the teams.

This example shows that after nearly 40 years of advances in collaboration environments, Brooks law [14]—the cost of a project increases as a quadratic function of the number of developers—remains unchanged. Empirical studies have shown that the time taken by a team to implement a feature or to fix a bug grows significantly with the increase in the size of a team [19, 48, 78].

Therefore, creating a seamless coordination environment is more than simply instituting a set of coordination tools, even when they are state of the art. This is not surprising and has been observed before [46]. Recent empirical and field studies have begun to articulate how organizations and individuals within organizations use and experience coordination technology. We derive three key observations from the collective studies as follows.

First, existing coordination technologies still exhibit significant gaps in terms of the functionality they offer versus the functionality that is needed. For instance, configuration management systems do not detect and cannot resolve all of the conflicts that result from parallel work. Process environments have trouble scaling across geographically distributed sites [22]. Many expertise recommender systems point to the same expert time and again, without paying attention to the resulting workload. Similar nontrivial gaps exist in other technologies.

Second, an organization's social and cultural structures may be a barrier to successful adoption of coordination technology. For instance, individuals are hesitant to share information regarding private work in progress and may not even do so, despite the benefits that tools could bring them. On a larger scale, numerous organizations still prohibit parallel work on the same artifact, despite clearly documented disadvantages of this restriction and an abundance of tools that support a more effective copy-edit-merge model.

Third, the introduction of specific coordination technologies in a particular organizational setting may have unanticipated consequences. Sometimes such consequences benefit the organization, as in the case when bug tracking systems and email archives are adopted as sources for identifying topic experts or personnel responsible for parts of a project. But, these consequences often also negatively impact an organization, such as when individuals rush changes and make mistakes only to avoid being the person who has to reconcile their changes with those of

others. These kinds of effects are subtle and difficult to detect, taking place silently while developers believe they are following specified procedures [27].

Combined, these three observations highlight how difficult it is for organizations to introduce the right mix of coordination technology and strategies. Tools do not offer perfect solutions, tool adoption cannot always be enforced as envisioned, and tool use may lead to unintended consequences. It is equally difficult for those inventing and creating new technology to place their work in the context of already existing conventions, work practices, and tools. It is not simply a matter of additional or improved functionality. Instead, a complex interaction among social, organizational, and technical structures determines what solutions eventually can succeed and in which context.

In the rest of the chapter, we provide an organized tour of the coordination technology that have been developed and how they relate to one another, so that researchers and end users can make informed choices when selecting their coordination technologies. When presenting these technologies, we also trace which of these technologies emerged from research and which from industry to showcase the close connection between industry and research development.

2 Organized Tour of Coordination Technologies

Coordination technologies have been and continue to be developed—in both academia and in industry—with the goal to reduce both the number of occurrences of coordination problems and their impact. While some of these technologies have been explicitly designed to facilitate coordination, others have been appropriated to do so (e.g., developers using the status messages in instant messaging systems to signal when they are busy and should not be interrupted).

Organizations need to select a portfolio of individual tools that best matches and streamlines their desired development process from the hundreds of coordination technologies that are now available. Each of these technologies provides functionalities, whose success and adoption depend on the organizational readiness of the team, its culture, working habits, and infrastructure.

We categorized existing coordination technologies into the *Coordination Pyramid framework*.[1] The Coordination Pyramid recognizes several distinct paradigms in coordination technology that have been prompted by technological advancements and changes in organizational and product structures. These paradigm shifts are not strictly temporal. However, each paradigm shift has enabled new forms of coordination practices to emerge and brings with it increasingly effective tools through which developers coordinate their day-to-day activities.

The Coordination Pyramid organizes these paradigm shifts in a hierarchy of existing and emerging coordination technology. In doing so, the Pyramid affords

[1] This chapter is an extension of the framework in [83].

two insights. First, it illustrates the evolution of the tools, starting from minimal infrastructure and an early focus on explicit coordination mechanisms to more flexible models that provide contextualized coordination support. Second, it articulates the key technological and organizational assumptions underlying each coordination paradigm.

3 The Coordination Pyramid

The Coordination Pyramid classifies coordination technology based on the *underlying paradigm of coordination,* that is, the overarching philosophy and set of rules according to which coordination takes place. It is complementary to other frameworks that have classified coordination technology: based on the temporality of activities, location of the teams, and the predictability of the actions [52, 76]; the interdependencies between artifacts and activities and how tools support these relationships [60]; and the extent to which tools model and support the organizational process [33, 46].

Looking through the suite of existing coordination technology, we find that they can be categorized into *five distinct paradigms to coordination*, with the first four paradigms being more mature with a variety of tools and the fifth one emergent. These five paradigms are represented by the layers of the Coordination Pyramid as shown in Fig. 2 (the vertical axes of the pyramid). Each layer articulates the kinds of technical capabilities that support a paradigm and is further classified along three strands, where each strand represents a fundamental aspect of coordination in which humans need tool support: *communication, artifact management, and task management.* Finally, each cell presents a small set of representative tools.

Note that a symbiotic relationship exists between the technical capabilities that comprise a layer and the context in which these capabilities are used. A key consideration in interpreting the Coordination Pyramid is that layers represent incrementally more effective forms of coordination. Tools at higher layers provide increasingly effective coordination support since they recognize and address the interplay between people, artifacts, and tasks, and they provide more automated support that is contextualized to the (development) task. We believe that the ability of an organization to reduce the number and impact of coordination problems increases with the adoption of each new paradigm. 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. However, in moving up to a higher layer, overall organizational effort devoted to coordination is reduced as some of the work is offloaded to the coordination technology that is used.

A key structural feature of the Coordination Pyramid is that its strands blend at higher layers. This indicates that *advanced coordination technologies tend to integrate aspects* of communication, artifact management, and task management in order to make more informed decisions and provide insightful advice regarding

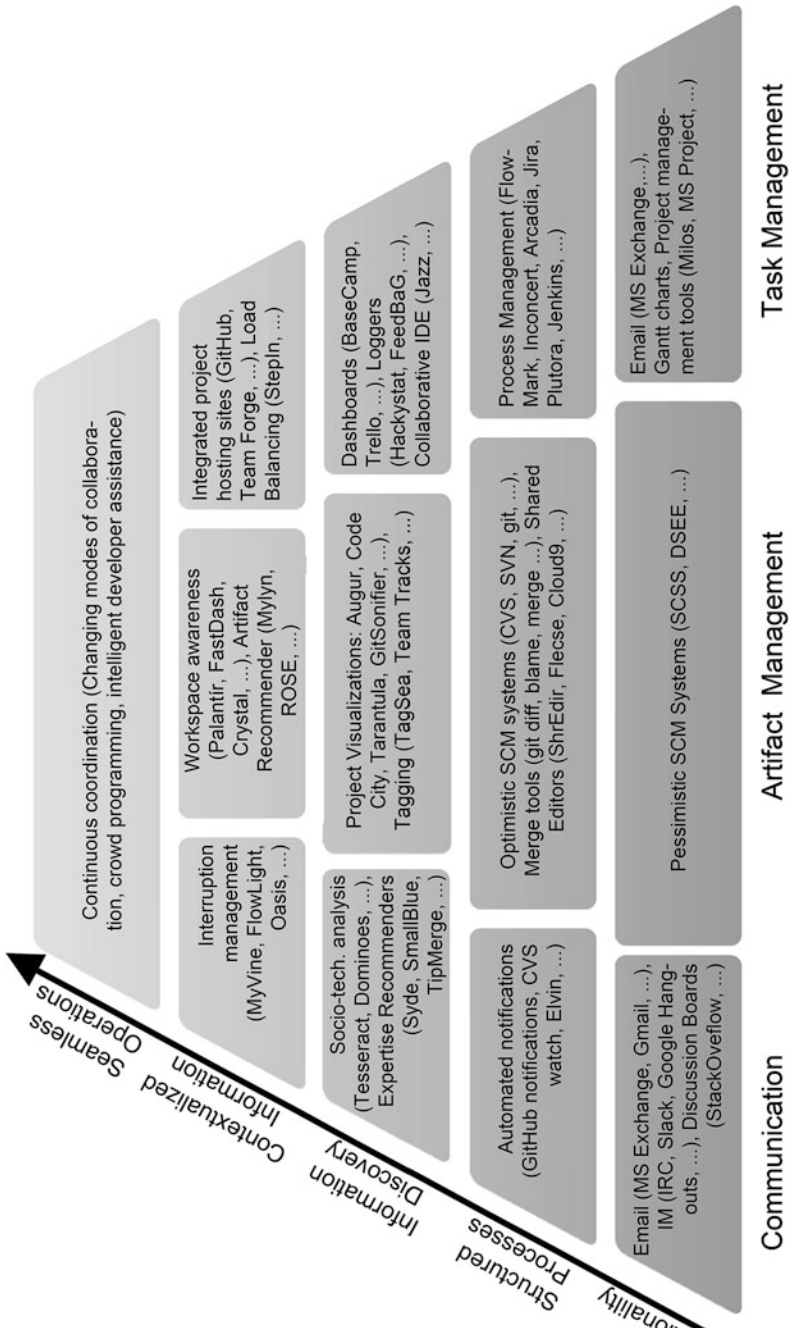

Fig. 2 The coordination pyramid

potential coordination problems. Table 1 provides example of tools categorized based on their primary support for a programming paradigm (layer) and a coordination aspect (strand). Many of these tools support multiple coordination aspects, especially in the higher layers; in such cases, we place them in the pyramid/table based on their primary focus. Further, note that some of the tools have had their beginnings in research (marked in bold in Table 1), whereas others have originated in industry (marked in italic in Table 1); this shows how research concepts drive innovation in industry as well as how ideas and innovation from industry drive research in coordination needs and technology.

We left the top of the Pyramid open as new paradigms are bound to emerge as the field forges ahead that strive to seamlessly integrate the different aspects of coordination support to provide an integrated, contextualized collaborative development environment.

3.1 Layer 1: Basic Functional Support

This layer is the foundation and provides basic functional support for collaboration. Tools in this layer bring a shift from manual to automated management of collaboration in the workplace. These tools support communication among developers, access and modification rights for common artifacts, and task allocation and monitoring for managers. Supports in these domains are the minimal necessities for a team to function. Many successful open-source projects operate by relying only on tools in the functional layer [69].

Communication A large part of coordination is communication, email being the predominant form of communication. Email provides developers with means to communicate easily and quickly across distances. Although asynchronous, it automatically and conveniently archives past communications and allows sharing the same information with the entire team. Even after 40 years, these benefits still make email the most popular communication medium. Various email clients exist, starting from the early versions, such as ARPANET [28] and the Unix mail [92], to more recent ones, such as Google's Gmail [44] and Microsoft Exchange [65].

Apart from emails, news groups and discussion forums are heavily used for communication. In the open-source community, such forums are more convenient to convey general information meant for a large number of people. These forums and discussion lists also allow people to choose what information they are interested in and subscribe to specific topics or news feeds.

A special kind of forum is one dedicated to Q&A. A popular example of this is Stack Overflow [88]. It allows developers to ask programming questions and the community to post replies along with code examples. It also provides features like up-vote (similar to "like") to signal the usefulness of the answers. To date, there are billions of questions and answers available in Stack Overflow.

The asynchronous nature of emails and discussions forums, however, can be an obstacle for large distributed teams that need to communicate quickly. Instant

Table 1 Research tools are marked in bold; industrial tools are marked in italic

Layer aspect	Communication	Artifact management	Task management
Basic functionality support	E-mail: **ARPANET [28],** *UnixMail [92], Gmail [44], Microsoft Exchange [65]* Instant message: **IRC [75],** *Google Hangouts [45], Slack [86]*	Pessimistic SCM systems: *SCCS [80],* **RCS [93], DSEE [59]**	Project management: **Milos [43],** *MS Project [66]*
Structured processes	Automated event notification: **Elvin [36],** *GitHub notifications [42],* **CVS watch [8]**	Optimistic SCM systems: **CVS [8],** *SVN [91], Adele/Celine [34], ClearCase [2],* **Git [41], Mercurial [64]** Shared editors: **ShrEdit [62], Flecse [31],** *Cloud9 [4]*	Workflow systems: *FlowMark [70],* **Inconcert [61]** Process environments: **Arcadia [90],** *Plutora [79]* Issue trackers: *Bugzilla [16], JIRA [6], GitHub issues [42]* Continuous integration: *Jenkins [87]*
Information discovery	Sociotechnical analysis: **Ariadne [94], Tesseract [81], Tarantula [53],** *SmallBlue [49]* Expertise queries: **Expert recommender [68], Visual Resume [85], SmallBlue [49], TipMerge [23], EEL [67], Syde [47], InfoFragment, Dominoes [25]**	Project visualization: **SeeSoft [32], GitSonifier [72], CodeCity [96], Augur [39], Evolution Matrix [56]** Artifact tags: **Team Tracks [29], TagSEA [9]**	Logger: *Hackystat [51],* **FeedBaG [3]** Dashboard: *Trello [95], BaseCamp [7]* Collaborative IDE: *JAZZ [40],* **SocialCDE [17]**
Contextualized information	Interruption management: **MyVine [37], Oasis [50], FlowLight [99]**	Workspace awareness: **Palantír [84], CollabVS[30], Crystal [15], FastDash [10]** Artifact recommender: **Hipikat [24], ROSE [98], Mylyn [54]**	Integrated project hosting sites: *GitHub [42], CollabNet TeamForge [21], Atlassian BitBucket [5]* Load balancing: **StepIn [97]**

messaging applications were developed to enable synchrony in communication. The first messaging app was Internet Relay Chat (IRC) [75]. Instant messaging has since rapidly evolved to the more feature-enriched and multifunctional messaging apps like Google Hangouts [45] and Slack [86]. Slack, a cloud-based communication application, has gained popularity as it allows seamless integration with many other applications and supports different media types and multiple workspaces (channels) for communication per team. Slack also allows users to set availability status (per channel) to manage interruptions per group.

Artifact Management Tools for artifact management need to satisfy three major requirements: access control to common artifacts, a personal workspace to make changes, and a version control system. These requirements are exactly what a Software Configuration Management (SCM) system provides. The codebase is stored in a central repository. Developers *check out* relevant artifacts from this repository into their personal workspaces, where they can make changes without getting interrupted by changes that others are making. When the desired modifications are complete, developers can synchronize (*check-in*) their changes with the central repository making the changes available to the rest of the team.

SCM systems also provide *versioning* support, where every revision to an artifact is preserved as a new version along with who made the changes and the rationale for the changes. These functionalities—maintenance of software artifacts, preserving change history, and coordinating changes by multiple developers—form a key support for the development process.

Despite the benefits that SCM systems bring, they also introduce new problems; since developers can check out the same artifact, they can make conflicting changes. To avoid such cases, SCM systems enforced a process where artifacts that were checked out were locked. Such lock-based systems are called pessimistic SCM systems and include the following technology: SCCS [80], RCS [93], and DSEE [59]. Although the pessimistic nature of these systems limits parallel changes, they were the first kind of automated support for artifact management. As a result, they gained popularity and became a key requirement for software artifact management [35]. (Optimistic (non-lock-based) SCM systems are discussed in the next layer.)

Task Management The basic support for task management includes two parts: allocation of resources and monitoring of tasks. In some cases, the managers are responsible for allocating work, such that tasks are independent and optimally assigned based on developer's experience. In other cases, developers themselves choose tasks. In either case, task completions need to be monitored to ensure that bottlenecks do not occur and the project is on track.

Prior to tools in this layer, the process of task management was manual with notes, schedules, and progress kept by pen and paper. The very early versions of project management tools (e.g., Milos [43], MS-project [66]) provided basic project planning and scheduling tools. Milos started as a research project, and industry quickly picked up the concept to come up with tools providing much more functionality.

Note that teams also use email to coordinate who is working on which task, schedule meetings, and monitor updates about tasks [69]. We therefore place email in both the communication and task management strands in the pyramid.

Summary This layer provides the basic functions necessary for a team to collaborate. Email provides the basic communication medium; developers and managers spend a large portion of their time reading and responding to emails. However, when real-time responses are needed, instant messaging applications are favored over slower, formal email communication. Pessimistic SCM systems provide basic access control and versioning to manage changes to artifacts. Project management tools and emails help teams manage tasks. Some of the early tools in all three strands emerged from research projects, but the current tools in use today are largely industry driven.

3.2 Layer 2: 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 based on specific interaction protocols embedded in the tools.

Communication In this layer, we see the entwining of the strands since tools provide better support when they integrate multiple aspects of coordination. While email still remains an important part of one-on-one communication, a lot of communication among developers includes aspects of the artifact and its associated changes. Communication archived along with artifacts maintains its context and is easier to retrieve and reason about at a later stage. As a result, we place tools here that send notifications because of changes to a specific artifact (e.g., new version created) or because of a specific action (e.g., check-in).

Developers can monitor contributions to a repository through the "watch" feature in GitHub [42] or changes to a specific file through the CVS watch system [8]. In a similar fashion, Elvin [36] notifies developers about commits by the team. Such notifications triggered based on changes to artifacts facilitate communication among developers working on the same project, artifact, or issue.

Artifact Management Tools in this layer went through a revolution because of the need to support coordination in parallel software development, which required concurrent access to artifacts. This in turn sparked changes in tools, making artifact management central to communication and task management functionalities. For example, parallel development made it important for developers to know whether someone else was working on the same artifact, what were the latest changes to an artifact, and when changes needed to be synchronized.

This led to optimistic SCM systems that support parallel development by allowing developers to "check out" any artifact at any time into their personal workspace even if these were being edited in other workspaces. Initial systems, such as CVS [8], SVN [91], Adele/Celine [34], and ClearCase [2], follow a centralized model where the main line of development is maintained in the trunk: Developers check out artifacts, make changes, and synchronize the changes back to the main trunk. Later systems, such as Git and Mercurial, follow a decentralized model, where development can occur in multiple repositories and the team (or developer) can choose which repository to "pull" changes from or "push" changes to.

In these SCM systems, multiple developers can make concurrent changes to the same artifact (causing merge conflicts) or make changes that impact ongoing changes to other artifacts (causing build or test failures). Therefore, to help developers synchronize their changes, these systems provide mechanisms (e.g., git diff) that allow developers to identify what has changed between two versions of an artifact or who has made changes to a specific artifact (e.g., git blame). Many systems provide automated merge facilities [63]. Some SCM systems also provide different access rights to different developers based on developers' roles in the project [35].

Shared editor systems also facilitate parallel development by enabling collaborative editing in a cloud platform, such that all changes are continuously synchronized in the background. These systems, therefore, avoid issues where changes that are performed in isolation cause conflicts when merged. Table 1 lists some synchronous editing platforms such as Flecse [31], ShrEdit [62], and Cloud9 [4].

Task Management A typical mechanism of supporting task management is to decompose a complex task into smaller set of steps, such that a process can be defined (and automated) regarding how a work unit flows across these steps [73]. Workflow modeling systems were developed to automate this process. A key part of these systems is the workflow model that formalizes the processes that are performed at each development step and models how a unit of work flows between the steps [73]. Systems, such as FlowMark [70] and Inconcert [61] (see Table 1), provide editing environments and features to create and browse a workflow model. They also help capture requirements, constraints, and relationships for each individual step in the workflow.

Similarly, process-centered software engineering environments (PSEEs) are environments that provide a process model (a formal representation of the process) to support development activities. Table 1 provides some examples of these environments. Arcadia [90] is an example of such a system that also additionally supports experimentation with alternate software processes and tools.

Today process support is largely handled through policies encoded in a project hosting site or via DevOps tools. Issue tracking systems (e.g., JIRA [6], BugZilla [16], GitHub Issues [42]) codify processes for identifying bugs or features, mechanisms to submit code (patches), code review, and linkages between changes and the issue. Continuous integration is a common practice used in DevOps, such that all modifications are merged and integrated at frequent intervals to prevent

"integration hell." Tools like Hudson [77] and Jenkins [87] allow automated continuous integration and deployment features. Plutora [79] supports automated test management tools and release of a software product. These and other tools shown in Table 1 are heavily used in today's software development to automate the parts of the development processes.

Summary 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, while the cost of technology in this layer might be initially high, an organization can recoup that cost by choosing its processes carefully. 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. As in the prior layer, the majority of current tools in this layer are now industry driven.

3.3 Layer 3: Information Discovery

Structured processes create the scaffolding around which other forms of informal coordination take place. Such coordination occurs frequently and relies on users gaining information that establishes a context for their work. The technology at the Information Discovery layer aims to support this informal coordination that surrounds the more formal processes established in the team [82]. Tools at this layer empower users to proactively seek out and assemble the information necessary to build the context surrounding their work. As with the Structured Processes layer, the Information Discovery layer also represents automation of tasks that otherwise would be performed manually. The tools make the information needed readily accessible by allowing users to directly query for it or by providing developers with visualizations. The availability of these kinds of tools is critical in a distributed setting, where subconscious buildup of context is hindered by physical distances [48]. Table 2 presents examples of tools in this layer categorized based on their primary focus.

Communication A key focus of tools in this layer is to allow developers to find pertinent information about their team and project so that they can self-coordinate, which requires an integration with the artifact and task management strands. For example, communication about project dependencies, past tasks, and design decisions are some of the ways that developers can self-coordinate [48].

Table 2 Sampling of tools in the information discovery layer

Tool	Description
Communication	
Syde [47]	Stand-alone interactive expertise recommender system that provides a list of experienced developers for a particular a code artifact by taking into account changing code ownerships
Tesseract [81]	Interactive project exploration environment that visualizes entity relationships among code, developers, bugs, and communication records and computes the level of congruence of communication in the team
TipMerge [23]	Tool that recommends developers who are ideal to perform merges, by taking into consideration developers' past experience, dependencies, and modified files in development branches
Artifact management	
CodeCity [96]	Interactive 3D visualization tool that uses a city metaphor to depict object-oriented software systems; classes are "buildings" and packages are "districts"
GitSonifier [72]	Interactive visualization that overlays music on top of visual displays to enhance information presentation
Tarantula [53]	A prototype that uses visual displays to indicate the likelihood of a statement containing a bug based on data collected from past test runs
Task management	
BaseCamp [7]	Web-based task management platform that supports instant messaging, email, task assignment, to-do, status reports, and file storage
Hackystat [51]	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

A key part of self-coordination is planning tasks so that it doesn't interfere with other ongoing work. To do this, developers need to understand the socio-technical dependencies in their project—social dependencies caused because of technical dependencies among project artifacts. Understanding these dependencies can help developers plan their work, communicate their intentions, and understand the implications of their and others' ongoing changes [26]. In fact, it has been found that developers who are aware of these socio-technical dependencies, and manage their communications accordingly, are more productive [19]. Several tools exist that allow developers to investigate socio-technical dependencies in their projects: Tesseract [81] uses cross-linked interactive displays to represent the various dependencies among artifacts, developers, and issues based on developers' past actions. It also calculates the extent to which team communication is congruent—the fit between the communication network (email and issue tracker comments) and the socio-technical dependency network (developers who should communicate because they work on related files). Similar other tools exist. Ariadne [94] visualizes relationships among artifacts based on who has changed which artifacts in the past.

Information Fragment [38] and Dominoes [25] allow users to perform exploratory data analysis in order to understand project dependencies to answer specific questions.

Developers may also need to communicate with someone who either has more experience in a specific artifact or has code ownership of that artifact. Tools such as Syde [47] track the code ownership on an artifact based on past edits. Similarly, Expertise Browser [68] quantifies the past experience of developers and visually presents the results so that users can distinguish those who have only briefly worked on an area of code from those who have extensive experience. Tools, such as Emergent Expertise Locator [67] and TipMerge [23], use team information of who has made what changes and the code architecture to propose experts as a user works on a task. Other tools, such as Visual Resume [85], synthesize contributions to GitHub projects and Stack Overflow to present visual summaries of developers' contributions. Small blue [49] uses analytics to identify someone's expertise level and provide recommendations for improving expertise.

Artifact Management Research on artifact management has produced visualization tools that aim to represent software systems, their evolution, and interdependencies in an easy-to-understand graphical format, such that users can better understand the project space and self-coordinate. Some of these visualizations concentrate at the code level (Augur [39], SeeSoft [32]), while others visualize the software system at the structural level (CodeCity [96], Evolution Matrix [56]). Many of these tools have additional specific focus: Tarantula [53] presents the amount of "testedness" of lines of code, whereas Syde [47] reflects the changes in code ownership in the project. Tools, such as GitSonifier [72] and code swarm [74], explore the use of music overlaid on top of visual displays to enhance information presentation. More information about such tools can be found in the survey by Storey et al. [89], classifying tools that use visualizations to support awareness of development activities.

Tools that allow developers to tag relevant events or annotate artifacts allow the creation of a richer artifact space that facilitates project coordination. For example, the Jazz IDE [20] allows the annotation of an artifact with "chat" discussions about that artifact. TagSea [9] uses a game-based metaphor, where developers are challenged to tag parts of the codebase that they consider useful for tasks. TeamTracks [29] takes a complementary approach where it records and analyzes developers' artifact browsing activities to identify artifact visiting patterns related to specific task contexts, which can then guide developers' navigation at a later date.

Task Management The tools at this layer support task management by facilitating task assignment and monitoring. Commercial project management tools such as BitBucket [5] and GitHub [42] provide dashboards that provide overview of tasks and contributions. Many of these project management tools allow for work assignment through interactive dashboards that can then be monitored (e.g., BaseCamp [7], Trello [95]). Tools, such as FeeDBaG [3] and HackyStat [51], log developer interactions with the IDE to provide additional feedback and allow developers to

query these logs to enable them to understand past changes and how these affect their own tasks. Some IDEs also provide collaboration support by leveraging the data archived in different project repositories, such as version histories, change logs, chat histories, and so on. For example, Jazz [40] and SocialCDE [17] provide IDE features that facilitate collaboration via integrated planning, tracking of developer effort, project dashboards, reports, and process support.

Summary In this layer, the benefits of blending communication, artifact management, and task management become clear. While we have placed tools in a particular strand based on their primary focus, all these tools encompass multiple coordination aspects (strands). For example, a visualization that highlights code that has been traditionally buggy can communicate to developers or managers useful information: A developer can assess the possibility that a new change to that (buggy) part may introduce new bugs and may need additional testing; a project manager can decide to put additional personnel when modifying those parts. Similarly, a socio-technical network analysis might not only reveal coordination gaps but also identify artifacts that developers usually modify together signaling the need for an architectural or organizational restructuring. We note that organizational use of tools in this layer especially in the communication and artifact management strands has been limited to date, which is not surprising since many of these tools are only now maturing from the research community. In contrast, the tools in the task management strand have mostly originated as commercial systems, and many are being heavily used, especially by smaller, techno-savvy organizations.

3.4 Layer 4: Contextualized Information Provision

The technology at the *Information Provision* layer has two highlights. First, coordination technology at this layer focuses on automatically predicting and providing the "right" coordination information to create a context for work and guide developers in performing their day-to-day activities. Therefore, while the tools in the previous layer allowed developers to proactively self-coordinate, the tools at this layer are themselves proactive. Key properties of these tools are that they aim to share only relevant information (e.g., the right information to the right person at the right time) and do so in a contextualized and unobtrusive manner (e.g., information to be shared is often embedded in the development environment). The crux of this layer, therefore, 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. Table 3 presents select examples of tools in this layer categorized as per their primary focus.

Second, the technology in this layer shines because they draw on multiple and diverse information sources to enable organic forms of self-coordination, representing tighter integration among the different coordination aspects (strands).

Table 3 Sampling of tools in the contextualized information layer

Tool	Description
Communication	
FlowLight [99]	A tool that aims to reduce interruptions by combining a traffic light like LED with an automatic mechanism for calculating an *"interruptibility"* measure based on the user's computer activity
MyVine [37]	Prototype that provides availability awareness for distributed groups by monitoring and integrating with phone, instant message, and email client
Oasis [50]	Interruption management system that defers notifications until users in interactive tasks reach a breakpoint
Artifact management	
Mylyn [54]	An Eclipse plugin that creates a task context model to predict relevant artifacts for a task by monitoring programmers' activity and extracting the structural relationships of program artifacts
Palantír [84]	An Eclipse extension that supports early detection of emerging conflicts through peripheral workspace awareness
ROSE [98]	An Eclipse plugin that uses data mining techniques on version histories to suggest future change locations and to warn about potential missing changes
Task management	
GitHub [42]	Web-based open-source version control system and internet project hosting site. Provides source code management and features like bug tracking, feature requests, task management, and wikis
StepIn [97]	Expertise recommendation framework that considers the development context (information overload, interruption management, and social network benefits) when recommending experts
TeamForge [21]	Web-based lifecycle management platform that integrates version control, continuous integration, project management, and collaboration tools

Communication Tools in this layer facilitate communication by taking into consideration the context of the work in which a developer is engaged. Interruption management tools are representative of this guiding principle, as they allow developers to channel their communication while ensuring someone else's work is not disrupted. For example, MyVine [37] integrates with a phone, instant message, an email client, and uses the context information from speech sensors, computer activity, location, and calendar information to signal when someone should not be interrupted. Flowlight [99] presents the availability of a developer through physical signals using red or green LED lights. Oasis [50] defers notifications until users performing interactive tasks reach a breakpoint.

Artifact Management When programming, developers need to create a context for the changes that they are going to make and an understanding of the impact of their changes on ongoing work and vice versa. Tools at this layer are proactive in helping

developers create such contexts for their tasks. Artifact recommendation systems help by letting developers know which other artifacts they should change as they work on their tasks. For example, Hipikat [24] creates a map of the project artifacts and their relationships to recommend which other artifacts should be changed. The Mylyn tool [54] extends Hipikat to weight the artifact recommendations, such that only relevant files and libraries are presented. Rose [98] uses data mining on version histories to identify artifacts that are interdependent because they were co-committed, which it then uses to suggest future change locations and to warn about potential missing changes.

Dependencies among artifacts (at different granularities) imply that developers need to understand the consequences of previous changes. They can then use this information to evaluate and implement their own changes so as to avoid coordination problems. Workspace awareness tools at this layer help self-coordination by providing information of parallel activities and the impact of these activities on current work by identifying direct conflicts (when the same piece of code has been changed in parallel) and indirect conflicts (when artifacts that depend on each other have been changed in a way which might result in a build or a test failure). Some of these tools use specialized displays (e.g., FastDash [10]) to reduce the effort and context switching that users have to undertake to identify and determine the impact of ongoing changes from the version control system or through program analysis tools, while others (e.g., Palantír [84], Crystal [15], CollabVS [30]) are integrated with the IDE to further reduce the context switching between the development task and change monitoring activities.

Task Management Tools at this layer continue their support via integrated development editors with the goal to provide more contextualized support for development activities, to reduce context switches, and to manage interruptions. For example, environments like GitHub [42] and TeamForge [21] notify developers of new commits or other relevant activities, especially when a developer's name is explicitly mentioned (e.g., a pull request review comment). Some technology hacks go one step further to use external devices (e.g., lava lamps or LED lights) to display the build status and its severity in the project. There is also exploratory work on environments that focus on making context in which code is developed as a central focus. For example, environments like CodeBubbles [13] identify and display relevant fragments of information next to the code being developed so that developers can create and maintain the context of their work.

Expertise recommendation also helps with task management. Here, at this layer, we place a recommendation system, such as StepIn [97], that not only recommends experts but also takes into consideration information overload, interruption management, and social network benefits when making recommendations about which task should be sent to which (expert) developer.

Summary Most tools at this layer are 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. More recent work has concentrated on integrating awareness with software processes, yielding more

powerful, scalable, and contextualized solutions. To be successful, tools at this layer must draw on multiple and diverse information sources to facilitate organic forms of self-coordination. Evidence of the potential of these tools is still limited to initial tests, with fieldwork and empirical studies of actual technology adoption and use much needed at this moment in time.

3.5 Layer 5: Seamless

The merging of strands at the higher layers of the Coordination Pyramid signifies a trend toward integrated approaches to coordination. For instance, in their communications, developers often need to reference a specific artifact or a specific task (issue). Tools at the Seamless (operations) layer enable contextualization of such discussions by integrating communication and artifact management into a single approach. As another example, managers at times need to identify experts who are the most appropriate to certain tasks. Combining the artifact management and the task management strands enables automated suggestions of such experts by mining past development efforts, their interests, schedule, and workload.

We anticipate that future paradigm shifts will eventually lead to what we term continuous coordination [82]: flexible work practices supported by tools that continuously adapt their behavior and functionality. No longer will developers need to use separate tools or have to explicitly interact with coordination tools. Their workbench will simply provide the necessary coordination information and functionality in a seamless and effective manner, in effect bringing coordination and work together into a single concept.

We leave the top of the Coordination Pyramid open as we believe new paradigms of coordination will emerge as technology and development paradigms continue to change. Continuous coordination likely will not be reached in one paradigm shift but will require multiple, incremental generations of coordination technology, approaches, and work practices to emerge first.

4 Conclusion and Future Work

The cost of coordination problems has not been quantified to date, and it may well be impossible to precisely determine. However, rework, unnecessary work, and missed opportunities are clearly a part and parcel of developers' everyday experiences. 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 team's collaborative effort. A developer seeking an answer and on not receiving one may, as a result, interrupt multiple other developers or spend significant amounts of their time and effort, a cost that

could have easily been saved. Larger problems can result in severe time delays, serious expenses in developer effort, critical reductions in code quality, and even failed projects as a result [14].

The Coordination Pyramid helps organizations and individuals better understand desired coordination practices and match these to available tools and technologies. It furthermore charts a road map toward improving an organization's coordination practices, by enabling organizations to locate where they presently are in the Pyramid, where they might want to be, and what some necessary conditions are for making the transition. 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 deduce the next steps: attempting to increase coordination support by moving further up, as well as sideways, in the Coordination Pyramid.

Some emerging coordination technology that we envision can bring contextualized coordination to the fore are as follows:

Crowd programming is a mechanism through which a crowd of programmers works on short tasks. Crowd programming extends the development paradigm of distributed software development, which has been extremely successful. Open-source software is an exemplar of distributed development where volunteers from all over the world coordinate their actions to build sophisticated, complex software (e.g., WordPress, Linux, GIMP). However, the amount of coordination needed to handle artifact dependencies and the size of tasks impacts the number of people who can contribute, which can be a bottleneck. Recent work on crowd sourcing has started to explore mechanisms to decompose complex tasks into small, independent (micro) tasks that can be completed and evaluated by a crowd of workers. CrowdForge [55] introduces a Map-Reduce style paradigm in which the crowd first partitions a large problem into several smaller subproblems and then solves the subproblems (map) and finally merges the multiple results into a single result (reduce). More recently, CrowdCode [57] is a programming environment that supports crowdsourcing of JavaScript programs. It dynamically generates micro-tasks and provides a platform for individuals to seek and complete these microtasks. Crowd programming is an exciting new approach that has the potential to change the current software development paradigm by being able to leverage the programming skills of a much larger set of individuals and reducing the development times. The primary challenges in this line of work include the following. First, it is difficult to find the right mechanism to decompose tasks that are independent yet are small so that they can be easily accomplished in a short session since crowd volunteers cannot be expected to spend significant amount of time on a given task. Second, there is the need for some form of automated evaluation to assess the quality of work that has been performed by the crowd. Finally, complex tasks that require

deeper knowledge of the design or codebase are unlikely to be solved by a crowd worker.

Chat bots have been popular in the online customer service domain and are now gaining popularity in software development [58]. For example, teams now use SlackBots—(ro)bots incorporated into Slack—to teach newcomers the development process of the team or to monitor system statuses (e.g., website outage, build failure). Other bots exist that also help in team building. For example, the Oskar bot inquires about individuals' feelings and shares information with the team so as to prevent isolation and to allow teammates to offer support. Similarly, Ava Bot monitors team morale by privately checking in with team members to make sure that the individual and their work are on track and raising issues with management when appropriate. Bots can also help with managing information overload by summarizing communication; for example, Digest.ai creates daily recaps of team discussions, and TLDR [71] generates summaries of long email messages. A key challenge in creating useful bots is correctly determining the context in which help is to be provided, such that questions are appropriately answered to match the specific questions and avoid rote answers. This is especially difficult if questions or situations are non-routine.

Intelligent development assistants (IDA) are automated helpers that can parse natural language, such that developers can interact with them effortlessly through audio commands and without needing to switch the development context. For example, Devy [12], a conversational development assistant, can automate the entire process of submitting a pull request through a simple conversation with the developer. We can also imagine other complex situations where IDAs can converse with the developer to answer questions, such as where a piece of code has been used, who had edited it last, and whether that person is currently working on a related artifact, and start a communication channel if the developer desires. These assistants can concatenate multiple, low-level tasks that span different repositories and coordination strands, asking for guidance or further information when needed. Another example of intelligent assistant is when the code itself is intelligent, caring about its own health. Code drones [1] is a concept where a class (or file) acts as an autonomous agent that auto-evolves its code based on the "experience" it gains by searching for (technology) news about security vulnerabilities, reading tweets about new performant APIs, and correlating the test environment results with the key performance indicators. The challenges with this line of work, similar to that of bots, revolve around correctly determining the context of a change and the intent of the developer to provide timely help that is appropriate to the situation.

In *summary*, coordination technology to facilitate distributed software development has gone through several paradigm shifts as envisaged by the layers in the Pyramid; this evolution has been driven by both industry and research ventures aimed at creating a seamless, efficient coordination environment where coordination is contextualized with the task at hand. We are at an exciting stage where several new technologies are being developed that have the potential to revolutionize how large, distributed teams seamlessly coordinate between and within teams.

Acknowledgements I would like to thank André van der Hoek and David Redmiles who had contributed to a previous version of the classification framework. A special thank you to Souti Chattopadhyay and Caius Brindescu for their help in reviewing and providing feedback on this chapter.

Key References

1. Empirical study on how distributed work introduces delays as compared to same-site work: Herbsleb and Mockus [48].
2. Empirical study that investigates the effects of geographical and organizational distances on the quality of work produced: Bird et al. [11].
3. A theoretical construct of how social dependencies are created among developers because of the underlying technical dependencies in project elements: Cataldo et al. [19].
4. Empirical study of how developers keep themselves aware of the impact of changes on their and others' works: de Souza and Redmiles [26].
5. A retrospective analysis on the impact of Software Engineering Research on the practice of Software Configuration Management Systems: Estublier et al. [35].
6. Tesseract, an exploratory environment that utilizes cross-linked displays to visualize the project relationships between artifacts, developers, bugs, and communications: Sarma et al. [81].
7. A workspace awareness tool that identifies and notifies developers of emerging conflicts because of parallel changes: Sarma et al. [84].
8. A framework for describing, comparing, and understanding visualization tools that provide awareness of human activities in software development: Storey et al. [89].
9. A first use of data mining techniques on version histories to guide future software changes: Zimmermann et al. [98].
10. Continuous coordination: A paradigm for collaborative systems that combines elements of formal, process-oriented approaches with those of the more informal, awareness-based approaches: Sarma et al. [82].

References

1. Acharya, M.P., Parnin, C., Kraft, N.A., Dagnino, A., Qu, X.: Code drones. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 785–788 (2016)
2. Allen, L., Fernandez, G., Kane, K., Leblang, D., Minard, D., Posner, J.: Clearcase multisite: supporting geographically-distributed software development. In: International Workshop on Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, pp. 194–214 (1995)
3. Amann, S., Proksch, S., Nadi, S.: Feedbag: an interaction tracker for visual studio. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), pp. 1–3 (2016)
4. Amazon Web Services, Inc: AWS Cloud9 (2017). https://c9.io

5. Atlassian Pty Ltd: Atlassian BitBucket (2017). https://bitbucket.org/product
6. Atlassian Pty Ltd: Atlassian Jira (2017). https://www.atlassian.com/software/jira
7. Basecamp, LLC: Basecamp (2017). http://basecamp.com
8. Berliner, B.: CVS II: parallelizing software development. In: USENIX Winter 1990 Technical Conference, pp. 341–352 (1990)
9. Biegel, B., Beck, F., Lesch, B., Diehl, S.: Code tagging as a social game. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 411–415 (2014)
10. Biehl, J.T., Czerwinski, M., Smith, G., Robertson, G.G.: FASTDash: a visual dashboard for fostering awareness in software teams. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07, pp. 1313–1322. ACM, New York (2007)
11. Bird, C., Nagappan, N., Devanbu, P., Gall, H., Murphy, B.: Does distributed development affect software quality? An empirical case study of Windows Vista. In: 2009 IEEE 31st International Conference on Software Engineering, pp. 518–528 (2009)
12. Bradley, N., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. In: Proceedings of the 40th International Conference on Software Engineering (ICSE '18), pp. 993–1003. ACM, New York (2018)
13. Bragdon, A., Zeleznik, R., Reiss, S.P., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., LaViola, J.J. Jr.: Code bubbles: a working set-based interface for code understanding and maintenance. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2503–2512. ACM, New York (2010)
14. Brooks, F.P. Jr.: The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E. Pearson Education India, New Delhi (1995)
15. Brun, Y., Holmes, R., Ernst, M.D., Notkin, D.: Proactive detection of collaboration conflicts. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 168–178. ACM, New York (2011)
16. Bugzilla community: Bugzilla (2017). https://www.bugzilla.org
17. Calefato, F., Lanubile, F.: SocialCDE: a social awareness tool for global software teams. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pp. 587–590. ACM, New York (2013)
18. Carmel, E.: Global Software Teams: Collaborating Across Borders and Time Zones. Prentice Hall PTR, Upper Saddle River (1999)
19. Cataldo, M., Mockus, A., Roberts, J., Herbsleb, J.: Software dependencies, work dependencies and their impact on failures. IEEE Trans. Softw. Eng. **35**, 737–741 (2009)
20. Cheng, L.T., De Souza, C.R.B., Hupfer, S., Ross, S., Patterson, J.: Building collaboration into IDEs. edit - compile - run - debug - collaborate? ACM Queue **1**, 40–50 (2003)
21. CollabNet, Inc: CollabNet TeamForge (2017). https://www.collab.net/products/teamforge-alm
22. Costa, J.M., Cataldo, M., de Souza, C.R.: The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '11, pp 3151–3160. ACM, New York (2011)
23. Costa, C., Figueiredo, J., Murta, L., Sarma, A.: TIPMerge: recommending experts for integrating changes across branches. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, pp. 523–534. ACM, New York (2016)
24. Cubranic, D., Murphy, G.C., Singer, J., Booth, K.S.: Hipikat: a project memory for software development. IEEE Trans. Softw. Eng. **31**(6), 446–465 (2005)
25. da Silva, J.R., Clua, E., Murta, L., Sarma, A.: Multi-perspective exploratory analysis of software development data. Int. J. Softw. Eng. Knowl. Eng. **25**(1), 51–68 (2015)
26. de Souza, C.R.B., Redmiles, D.: An empirical study of software developers' management of dependencies and changes. In: Thirteeth International Conference on Software Engineering, Leipzig, pp. 241–250 (2008)
27. de Souza, C.R.B., Redmiles, D.F.: The awareness network, to whom should i display my actions? and, whose actions should i monitor? IEEE Trans. Softw. Eng. **37**(3), 325–340 (2011)

28. Defense Advanced Research Projects Agency: ARPANET (2017). https://www.darpa.mil/about-us/timeline/arpanet
29. DeLine, R., Czerwinski, M., Robertson, G.: Easing program comprehension by sharing navigation data. In: 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), pp. 241–248 (2005)
30. Dewan, P., Hegde, R.: Semi-synchronous conflict detection and resolution in asynchronous software development. In: ECSCW 2007, pp. 159–178. Springer, London (2007)
31. Dewan, P., Riedl, J.: Toward computer-supported concurrent software engineering. IEEE Comput. **26**, 17–27 (1993)
32. Eick, S.G., Steffen, J.L., Sumner, E.E. Jr.: Seesoft-a tool for visualizing line oriented software statistics. IEEE Trans. Softw. Eng. **18**(11), 957–968 (1992)
33. Ellis, C., Wainer, J.: A conceptual model of groupware. In: Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, CSCW '94, pp. 79–88. ACM, New York (1994)
34. Estublier, J.: The adele configuration manager. In: Configuration Management, pp. 99–133. Wiley, New York (1995)
35. Estublier, J., Leblang, D., van der Hoek, A., Conradi, R., Clemm, G., Tichy, W.F., Weber, D.: Impact of software engineering research on the practice of software configuration management. ACM Trans. Softw. Eng. Methodol. **14**, 1–48 (2005)
36. Fitzpatrick, G., Kaplan, S., Mansfield, T., Arnold, D., Segall, B.: Supporting public availability and accessibility with Elvin: experiences and reflections. In: ACM Conference on Computer Supported Cooperative Work, vol. 11, pp. 447–474 (2002)
37. Fogarty, J., Ko, A.J., Aung, H.H., Golden, E., Tang, K.P., Hudson, S.E.: Examining task engagement in sensor-based statistical models of human interruptibility. In: Proceedings of CHI 2005, pp. 331–340 (2005)
38. Fritz, T., Murphy, G.C.: Using information fragments to answer the questions developers ask. In: Proceedings of the International Conference on Software Engineering (ICSE), pp. 175–184 (2010)
39. Froehlich, J., Dourish, P.: Unifying artifacts and activities in a visual tool for distributed software development teams. In: Proceedings of the 26th International Conference on Software Engineering, ICSE '04, pp. 387–396. IEEE Computer Society, Washington (2004)
40. Frost, R.: Jazz and the eclipse way of collaboration. IEEE Softw. **24**, 114–117 (2007)
41. Git Project: Git (2017). https://www.git-scm.org
42. GitHub, Inc: GitHub (2017). https://github.com/
43. Goldmann, S., Münch, J., Holz, H.: MILOS: a model of interleaved planning, scheduling, and enactment. In: Web-Proceedings of the 2nd Workshop on Software Engineering Over the Internet, Los Angeles (1999)
44. Google LLC: GMail (2017). https://www.google.com/gmail/about/
45. Google LLC: Google Hangouts (2017). https://hangouts.google.com
46. Grudin, J.: Computer-supported cooperative work: history and focus. Computer **27**(5), 19–26 (1994)
47. Hattori, L., Lanza, M.: Syde: a tool for collaborative software development. In: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE '10, pp. 235–238. ACM, New York (2010)
48. Herbsleb, J., Mockus, A.: An empirical study of speed and communication in globally-distributed software development. IEEE Trans. Softw. Eng. **29**, 1–14 (2003)
49. International Business Machines Corporation: IBM Small Blue (2017). http://systemg.research.ibm.com/solution-smallblue.html
50. Iqbal, S.T., Bailey, B.P.: Investigating the effectiveness of mental workload as a predictor of opportune moments for interruption. In: CHI'05 Extended Abstracts on Human Factors in Computing Systems, pp. 1489–1492. ACM, New York (2005)
51. Johnson, P., Zhang, S.: We need more coverage, stat! classroom experience with the software ICU. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp. 168–178 (2009)

52. Johnson-Laird, P.N.: Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness. Harvard University Press, Cambridge (1983)
53. Jones, J.A., Harrold, M.J., Stasko, J.: Visualization of test information to assist fault localization. In: Proceedings of the 24th International Conference on Software Engineering, ICSE '02, pp. 467–477. ACM, New York (2002)
54. Kersten, M., Murphy, G.C.: Using task context to improve programmer productivity. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT '06/FSE-14, pp. 1–11. ACM, New York (2006)
55. Kittur, A., Smus, B., Khamkar, S., Kraut, R.E.: CrowdForge: crowdsourcing complex work. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST '11. pp. 43–52. ACM, New York (2011)
56. Lanza, M.: The evolution matrix: recovering software evolution using software visualization techniques. In: Proceedings of the 4th International Workshop on Principles of Software Evolution, IWPSE'01, pp. 37–42. ACM, New York (2001)
57. LaToza, T.D., Towne, W.B., Adriano, C.M., Van Der Hoek, A.: Microtask programming: building software with a crowd. In: Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology, pp. 43–54. ACM, New York (2014)
58. Lebeuf, C., Storey, M.A., Zagalsky, A.: How software developers mitigate collaboration friction with chatbots (2017). arXiv:170207011 [cs]
59. Leblang, D., McLean, G.: Configuration management for large-scale software development efforts. In: Proceedings of Workshop Software Engingeering Environments for Programming-in-the-Large, pp. 122–127 (1985)
60. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Comput. Surv. **26**(1), 87–119 (1994)
61. McCarthy, D., Sarin, S.: Workflow and transactions in inconcert. IEEE Data Eng. **16**, 53–56 (1993)
62. McGuffin, L., Olson, G.: ShrEdit: a shared electronic workspace. Tech. Rep., Cognitive Science and Machine Intelligence Laboratory, Tech report #45, University of Michigan, Ann Arbor (1992)
63. Mens, T.: A state-of-the-art survey on software merging. IEEE Trans. Softw. Eng. **28**, 449–462 (2002)
64. Mercurial Community: Mercurial (2017). https://www.mercurial-scm.org
65. Microsoft Corporation: Microsoft Exchange (2017). https://products.office.com/en-us/exchange/email
66. Microsoft Corporation: Microsoft Project (2017). https://products.office.com/en-us/project/project-and-portfolio-management-software
67. Minto, S., Murphy, G.C.: Recommending emergent teams. In: Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR '07, 5 pp. IEEE Computer Society, Washington (2007)
68. Mockus, A., Herbsleb, J.: Expertise browser: a quantitative approach to identifying expertise. In: International Conference on Software Engineering, Orlando, pp. 503–512 (2002)
69. Mockus, A., Fielding, R.T., Herbsleb, J.D.: Two case studies of open source software development: apache and mozilla. ACM Trans. Softw. Eng. Methodol. **11**(3), 309–346 (2002)
70. Mohan, C.: State of the art in workflow management research and products. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, SIGMOD'96, 544 pp. ACM, New York (1996)
71. Narayan, S., Cheshire, C.: Not too long to read: the tldr interface for exploring and navigating large-scale discussion spaces. In: 2010 43rd Hawaii International Conference on System Sciences, pp. 1–10 (2010)
72. North, K.J., Bolan, S., Sarma, A., Cohen, M.B.: Gitsonifier: using sound to portray developer conflict history. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 886–889. ACM, New York (2015)
73. Nutt, G.: The Evolution towards flexible workflow systems. Distrib. Syst Eng. **3**(4), 276–294 (1996)

74. Ogawa, M., Ma, K.L.: Code_swarm: a design study in organic software visualization. IEEE Trans. Vis. Comput. Graph. **15**(6), 1097–1104 (2009)
75. Oikarinen, J., Reed, D.: Internet relay chat protocol. RFC 1459, Internet Engineering Task Force (1993). https://tools.ietf.org/html/rfc1459
76. Olson, J.S., Olson, G.M.: Working together apart: collaboration over the internet. Synth. Lect. Human-Centered Inform. **6**(5), 1–151 (2013)
77. Oracle Corporation: Hudson CI (2017). http://hudson-ci.org
78. Perry, D.E., Siy, H.P., Votta, L.G.: Parallel changes in large-scale software development: an observational case study. ACM Trans. Softw. Eng. Methodol. **10**(3), 308–337 (2001)
79. Plutora, Inc: Plutora (2017). http://www.plutora.com
80. Rochkind, M.J.: The source code control system. IEEE Trans. Softw. Eng. **SE-1**(4), 364–370 (1975)
81. Sarma, A., Maccherone, L., Wagstrom, P., Herbsleb, J.: Tesseract: interactive visual exploration of socio-technical relationships in software development. In: 31st International Conference on Software Engineering, pp. 23–33. IEEE Computer Society, Washington (2009)
82. Sarma, A., Al-Ani, B., Trainer, E.H., Silva Filho, R.S., da Silva, I.A., Redmiles, D.F., van der Hoek, A.: Continuous coordination tools and their evaluation. In: Collaborative Software Engineering, pp. 153–178. Springer, Berlin (2010). https://link.springer.com/chapter/10.1007%2F978-3-642-10294-3_8
83. Sarma, A., Redmiles, D.F., van der Hoek, A.: Categorizing the spectrum of coordination technology. IEEE Comput. **43**(6), 61–67 (2010)
84. Sarma, A., Redmiles, D.F., van der Hoek, A.: Palantír: early detection of development conflicts arising from parallel code changes. IEEE Trans. Softw. Eng. **38**(4), 889–908 (2012)
85. Sarma, A., Chen, X., Kuttal, S., Dabbish, L., Wang, Z.: Hiring in the global stage: profiles of online contributions. In: 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), pp. 1–10 (2016)
86. Slack Technologies, Inc: Slack (2017). http://slack.com
87. Software in the Public Interest, Inc: Jenkins (2017). https://jenkins.io/
88. Stack Exchange Inc: StackOverflow (2018). https://stackoverflow.com
89. Storey, M.A., Cubranic, D., German, D.: 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, St. Louis, pp. 193–202 (2005)
90. Tate, A., Wade, K.: Simplifying development through activity-based change management. Tech. rep., IBM Software Group
91. The Apache Software Foundation: Apache Subversion (2017). https://subversion.apache.org/
92. Thompson, K., Richie, D.M.: UNIX Programmer's Manual. Bell Telephone Laboratories, Incorporate, New Jersey (1971)
93. Tichy, W.F.: RCS — system for version control. Softw. Pract. Exp. **15**(7), 637–654 (1985)
94. Trainer, E., Quirk, S., de Souza, C.R.B., Redmiles, D.F.: Bridging the gap between technical and social dependencies with ariadne. In: OOPSLA Workshop on Eclipse Technology eXchange, San Diego, pp. 26–30 (2005)
95. Trello, Inc: Trello (2017). https://trello.com
96. Wettel, R., Lanza, M.: CodeCity: 3D visualization of large-scale software. In: Companion of the 30th International Conference on Software Engineering, pp. 921–922. ACM, Leipzig (2008)
97. Ye, Y., Yamamoto, Y., Nakakoji, K.: A socio-technical framework for supporting programmers. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ESEC-FSE'07, pp. 351–360. ACM, New York (2007)
98. Zimmermann, T., Weisgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. In: Proceedings of the 26th International Conference on Software Engineering, ICSE '04, pp. 563–572. IEEE Computer Society, Washington (2004)
99. Züger, M., Snipes, W., Corley, C., Meyer, A.N., Li, B., Fritz, T., Shepherd, D., Augustine, V., Francis, P., Kraft, N.: Reducing Interruptions at Work: A Large-Scale Field Study of FlowLight, pp. 61–72. ACM Press, New York (2017)