Socio-technical Affordances for Stigmergic Coordination Implemented in MIDST, a Tool for Data-Science Teams

KEVIN CROWSTON, JEFFREY S. SALTZ, AMIRA REZGUI, and YATISH HEGDE, Syracuse University, USA
SANGSEOK YOU, HEC Paris, France

We present a conceptual framework for socio-technical affordances for stigmergic coordination, that is, coordination supported by a shared work product. Based on research on free/libre open source software development, we theorize that stigmergic coordination depends on three sets of socio-technical affordances: the visibility and combinability of the work, along with defined genres of work contributions. As a demonstration of the utility of the developed framework, we use it as the basis for the design and implementation of a system, MIDST, that supports these affordances and that we thus expect to support stigmergic coordination. We describe an initial assessment of the impact of the tool on the work of project teams of three to six data-science students that suggests that the tool was useful but also in need of further development. We conclude with plans for future research and an assessment of theory-driven system design.

CCS Concepts: • **Human-centered computing** → **Computer supported cooperative work**; *Empirical studies in collaborative and social computing*; • **Information systems** → *Data analytics.*

Additional Key Words and Phrases: stigmergic coordination; translucency; awareness; data-science teams

ACM Reference Format:
Kevin Crowston, Jeffrey S. Saltz, Amira Rezgui, Yatish Hegde, and Sangseok You. 2019. Socio-technical Affordances for Stigmergic Coordination Implemented in MIDST, a Tool for Data-Science Teams. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 117 (November 2019), 25 pages. https://doi.org/10.1145/3359219

1 INTRODUCTION

The goal of the project described in this paper is to better support coordination in data-science teams by transferring findings about coordination from another setting, namely free/libre open source software (FLOSS) development. FLOSS development was chosen as a source of ideas about coordination because of the visible success of FLOSS projects in harnessing the efforts of teams of developers. Data science was selected as a target domain because there is a recognized need for guidance on how data scientists should work together: data-science projects need to focus not just on machine-learning algorithms, but also on people and process [33, 34, 62]. As in FLOSS, task coordination is a key challenge for members of a data-science project [31], e.g., dividing the project into manageable pieces, assigning pieces to team members, tracking progress, dealing with interdependencies and integrating the pieces to a final project. Finally, while data science is distinct from software development, there is enough overlap to make the transfer of coordination approaches plausible. Data science is an emerging discipline that combines expertise across a

Authors' addresses: Kevin Crowston, crowston@syr.edu; Jeffrey S. Saltz, jsaltz@syr.edu; Amira Rezgui, arezgui@syr.edu; Yatish Hegde, yhegde@syr.edu, Syracuse University, Hinds Hall, Syracuse, New York, 13220, USA; Sangseok You, you@hec.fr, HEC Paris, 1 Rue de la Libération, Jouy-en-Josas, France, 78350.

2573-0142/2019/11-ART117 $15.00
https://doi.org/10.1145/3359219

range of domains, including software development but also data management and statistics. Data-science projects have goals such as identifying correlations and causal relationships, classifying and predicting events, identifying patterns and anomalies or inferring probabilities, interest and sentiment [27]. A common data-science tool is R [1]: analyses are performed by writing what are essentially programs in the R language that take data as input and output analysis results.

One approach to transfer findings is to simply reuse for the target setting tools that have proven to be useful for the source. For instance, data scientists could use git and GitHub to share R program files and to coordinate their work. However, tools intended for one kind of work product and practice may not be a perfect fit to others. For instance, GitHub may make tacit assumptions about the kinds of files being shared that are not descriptive of a data science context [10]. Specifically, contemporary software development is typically viewed within an object-oriented framework, while data-science analyses are typically more of a data-driven workflow [42]. As well, raw and intermediate data files can be quite large, making them unwieldy to upload or download. Hence, the specific features of tools that would be useful to data-science teams might be different than the features that are important for FLOSS development teams, making it difficult to know how to configure a data-science collaboration environment using existing tools [49].

We therefore explore an alternative approach, namely transferring findings about supporting coordination at more abstract level. Specifically we: 1) identify a novel form of coordination that appears to be part of the success of FLOSS development, namely stigmergic coordination [12], 2) advance stigmergic coordination theory by developing a conceptual framework for the socio-technical affordances that support stigmergic coordination in FLOSS, and 3) show the utility of the resulting conceptual framework by using it to guide design of a system that implements these affordances for a data-science setting. The paper includes a preliminary exploratory study of the use of the system as early-stage guidance for our system design. However, we emphasize that the evaluation is preliminary and exploratory and that the main contribution of the paper is a conceptual framework for system affordances that support stigmergic coordination, the utility of which is demonstrated through its use to guide system design (i.e., steps 2 and 3). The paper concludes with plans for future research and reflections on theory-based system development.

2 LITERATURE REVIEW

The goal of this paper is to present a conceptual framework for affordances that support stigmergic coordination. We therefore first review literature on stigmergic coordination to explain the concept and why it might be of interest for supporting data-science teams. Second, we draw on research on the role of documents in supporting collaborative work to develop a framework describing the affordances needed to support stigmergic coordination.

2.1 Stigmergic Coordination

In this section, we draw on research on coordination in FLOSS teams to describe a paradox that motivates the project: the apparent ability of certain distributed teams to coordinate with little or no explicit communication. This finding emerged from studies of how FLOSS developers coordinate [12, 40, 41]. Somewhat unexpectedly, these studies found little evidence of overt coordination: FLOSS developers seemed to communicate less often than expected about coding tasks. The lack of evidence was surprising considering the transparency of FLOSS projects. It was expected to find direct, discursive communication in email or other discussion fora through which developers interact but there were few examples. The lack of direct interaction around the work has echoes in other research findings. For example, research has found that developers mostly self-assign work rather than have it assigned to them [22, 23] and often make decisions about code without explicitly

evaluating options [37, 38]. Interestingly, when developers do discuss their work, they often refer directly to the software code.

In light of these findings, researchers have theorized that FLOSS development work can be coordinated at least in part through the code, the outcome of the work itself, a mode of coordination analogous to the biological process of stigmergy [35]. Heylighen defines stigmergy thusly: "A process is stigmergic if the work... done by one agent provides a stimulus ('stigma') that entices other agents to continue the job" [39]. Accordingly, stigmergic coordination can be defined as coordination based on signals from the shared work. For example, ants follow scent trails to food found by other ants, thus assigning labour to the most promising sources without the need for explicit interaction. The organization of the collective action emerges from the interaction of the individuals and the evolving environment, rather than from a shared plan or direct interaction.

While stigmergy was formulated to explain the behaviour of social insects following simple behavioural rules, it has also been invoked to explain human behaviours: the formation of trails in a field as people follow paths initially laid down by others (similar to ant trails), or markets, as buyers and sellers interact through price signals [56]. For humans and intelligent systems, the signs and processing can be more sophisticated than for insects [58]. For example, the shared environment can be a complex workspace including annotations. Tummolini & Castelfranchi [73] developed a typology of different kinds of messages possible from signs, such as having the ability to do something, having done something or having a goal. In CSCW, Christensen [18–20] discussed how architects and builders coordinate their tasks through "the material field of work" such as drawings, building on earlier work in CSCW focusing on coordination through the "field of work", including changes in shared databases [64].

Stigmergy has been suggested in particular as an interpretation of how FLOSS developers coordinate [12], what Kalliamvakou et al. called a "code-centric collaboration" perspective [43]. FLOSS developers mostly work with the code that they are developing, managed with source code control systems that provide status about the state of the code and development. Stigmergic coordination is valuable for FLOSS developers as it enables good coordination of the the development process while minimizing the need for explicit communication that can be costly, especially in a distributed environment such as FLOSS development. Stigmergy has also been argued as a mechanism in online work more generally. For instance, Elliot [29] argued that "[c]ollaboration in large groups is dependent on stigmergy," with the specific example of authoring on Wikis.

The question then is how work products can support coordination. From this perspective, we state a more specific question for our theorizing to advance stigmergic coordination theory: What socio-technical affordances of shared-work systems enable stigmergic coordination? By socio-technical affordances, we mean the features of the technology used and the practices around that technology. For example, the source-code control systems commonly used by FLOSS developers provide notifications of code submissions that enable other developers to maintain awareness of the state of the code to support coordination. But to interpret these change messages, developers need some level of technical skill and mental models of the code structure, another kind of affordance. They may also be accustomed to creating code in a way that is easier for others to interpret. The inherent nature of the coding task itself may create the need for specific kinds of coordination that are particularly amenable to stigmergy. Note that only the first of these affordances (notifications) is provided by reusing tools; the rest may or may not be transferable from domain to domain depending on the nature of the work practices and products.

If we can identify the socio-technical affordances important for FLOSS development, we may be able to develop a system and specify associated work practices to support them in another setting, thus enabling stigmergic coordination in that context. The advantage of stigmergic coordination is that people could achieve well-coordinated work with less explicit effort. And as in FLOSS

development, such coordination could be particularly valuable for distributed teams that face greater challenges in communicating for explicit coordination.

2.2 A Conceptual Framework for Socio-technical Affordances to Support Stigmergic Coordination

In this section we advance stigmergic coordination theory by building a conceptual framework for the socio-technical affordances that support stigmergic coordination. Our approach is to build a framework based on evidence from FLOSS development, which we then apply to the context of data-science projects. To theorize what affordances of work support coordination, we turn to the literature on documents and work [55]. Code, the shared work in the case of FLOSS development and data science, is a semiotic product recorded on a perennial substrate that is endowed with specific attributes intended to facilitate specific practices [76], thus making it a kind of document. Code differs from other kinds of documents by serving two audiences, one being a machine, the other programmers. However, we focus on the latter, describing properties of code that allows developers or data scientists to share their work with colleagues and to read, understand and respond to their intentions.

2.2.1 Documents Enable Coordination. Scholars have described how documentation and other accounts of work play a central role in the coordination of work [13, 14, 52–54, 67, 71, 72]. These perspectives have long pointed to the double role of documents as both "models of" work and "models for" work. For the first, documents provide an account of reality as workers manipulate text and other symbolic structures so as to parallel them with reality. For example, data scientists may carefully document the code they have constructed to create a report of the work (analysis) done. This view of work as a document can be seen in the emerging concept of data-science notebooks [45], which integrate code, comments about the code and the results / visualizations of the code.

But documents also provide a basis from which people further manipulate the world. For example, data-science reports are not simply accounts of work completed: the report, no matter how documented, can also guide ongoing work by suggesting what is left to be done, such as suggesting an attribute requiring further analysis. Taking inspiration from Smith [67] and Bakhtin [5], we suggest that a work product is rarely completely original; it is always an answer (i.e., a response) to work that precedes it, and is therefore always conditioned by, and in turn qualifies, the prior work. What the data scientist does when facing work is responsive and partially determined by what has been going on up until now. The analytical reports are thus accounts "for reality", as they provide a blueprint of the analysis taking shape. While typically used for exploratory data analysis, the previously-mentioned notebooks provide a hint at treating the data-science analysis as a document, in that these documents provide both "models of" work done and "models for" work to be done. Documents in this way offer a double accountability: when documenting the analysis of a data set, data scientists mold the account to the reality of the code on their computers and at the same time, mold their ongoing coding to the desires of the client.

Our focus on stigmergic coordination is how documents can serve as a model for work. With this focus in mind, three concepts from document studies stand out as helpful in articulating how documents can serve as a model for work: visibility, genre and combinability. We address each of these in turn.

2.2.2 Visibility of Work. The first key feature of documents is their visibility. Obvious as it may seem, making work visible to others is not a straightforward process. As discussed by Suchman [72], some work may be more visible than other work; some work may cover up previous activity and render it invisible. For example, service work is notoriously hard to make visible: The better such work is done, the less visible it is to those who benefit from it. Understanding what elements

of work are accessible and how its visibility may change over time is central to understanding how work may or may not serve as a model for future work.

Visibility is closely related to the concepts of awareness and of system transparency. There has been a stream of research in CSCW and elsewhere that demonstrates the importance of team member *awareness* for supporting collaborative work [e.g., 16, 17, 19, 28, 36]. Christensen [19] described actions a person might take to make a co-worker aware of an issue, and so distinguishes awareness from stigmergy, as "stigmergy does not entail making a distinction between the work and extra activities aimed solely at coordinating the work", such as drawing a co-worker's attention. Similarly, in contrast to active awareness (one participant calling for the attention of another), Dourish & Bellotti [28] argued for the importance of passive awareness mechanisms, which can be interpreted as supporting stigmergy.

Researchers have proposed awareness displays that allow a team member to develop an awareness of the actions of other team members without requiring the others to call attention to their work. Carroll and colleagues [16, 17] examine how awareness can support development of common ground, community of practice, social capital and human development in team. In this project, we focus more narrowly on how awareness of work supports coordination. Gutwin & Greenberg [36] present a framework for "workspace awareness" for real-time groupware systems. They note that awareness includes both problems of obtaining useful information and making sense of it, though they focus on the former. Their framework includes both what kind of information is available (e.g., who is doing what where?) and how it is gathered, which includes gathering information from intentional communication as well as from what people are doing and from artifacts, which we would interpret as supporting stigmergic coordination.

A second related concept is system *translucency* [30] or *transparency* [21, 25, 26, 69], meaning visibility of details of organizational processes or functions. Researchers have noted that technology enables new forms of transparency, e.g., as in GitHub, a software development site [24]. Consistent with our analysis of stigmergy, research has analyzed transparency as supporting information exchange or communication [69]. Researchers have noted similar problems with awareness and transparency, such as the potential for information overload from having to review too much information or that making too much visible may inhibit the willingness to share work [9, 25].

System transparency provides information that can influence how people work, i.e., one can view transparency as a system feature that might support awareness. Dabbish, et al. [26] note specifically that transparency is helpful for coordination. They list numerous uses of visibility information, such as including dependencies with other projects [25]. They further note that being able to see something means "much less need for routine technical communication" [25], suggesting that transparency is substituting for explicit coordination.

For work to be visible beyond a physically-restricted space, it must become mobile [48]. Most obviously, FLOSS development infrastructures support the mobility of work by being Internet-based. Any FLOSS developer can download the source code from the source-code control system (SCCS) and have access to others' work as a basis on which they can build their own. Further, many SCCS provide a mechanism to push changes to other workspaces, rather than having to wait for others to seek them out. By being available in multiple places, code can coordinate work in different settings. With ubiquitous access to the server containing the code, developers can more easily use others' work as a model for their own work. We expect the technical affordance of sharing files to be easy to translate to a data-science setting, though the size of data files may pose a challenge.

Visibility can include more than just the work itself. For example, SCCS typically provide a revision history: all changes made to each module in the system including what was created or deleted by whom, when. Many changes include short notes that can explain why a change was made, though many changes do not, apparently expecting the reader to examine the code itself.

Such histories not only serve as "models of" work but can also point forward by depicting the generally-accepted work process. For a newcomer, such histories provide a window to how things are done, what tasks tend to follow what tasks and what is regarded as good and opposed to bad (i.e., reverted) work.

Research on visibility and transparency, in particular studies of SCCS, can clearly be quite informative for designing systems to support stigmergic coordination. However, this stream of research has not specifically focused on the socio-technical affordances that enable users to make sense of and to use the provided stigma as models for work to support coordination, which is the goal of the current theorizing. Visibility of FLOSS work in particular is promoted both through the technology and through cultural norms about development that ensure that the work is understandable as a model for further work. For instance, a widely-acknowledged culture norm in open source is to "check in early, and check in often." If people do not share their work often (by checking it in to the SCCS), they are not making it visible to other participants to build on. Further, there are norms for providing "atomic commits," that is, developers are encouraged to address only one change or topic when making a commit, leading to many small commits rather than occasional large ones [3]. Large infrequent commits ("code bombs") are harder for other developers to understand, again hampering visibility. Indeed, a common complaint about a code contribution is that it is too large for developers to easily understand.

This research suggests that a system to support team coordination needs to provide visibility of the work to facilitate awareness of others' activities and mobility of the work to make it useful across settings. However, practices need to be developed to help team members ensure that the activities they make visible will be interpretable by others and to know what kinds of activities performed by others to attend to. For example, the FLOSS practice of making small commits might be difficult to translate to data science, as data scientists are not accustomed to thinking about work in terms of individual bug fixes or features. And simply sharing changes made to one large analysis file will not be effective if there are many dependencies among parts of the code, making it difficult to make atomic changes.

2.2.3 Genres. A second important feature of documents is that they come in different types, which we refer to as genres [75]. The notion of a genre combines together expected regularities of form and purpose [51]. For example, common document genres relating to an academic paper include paper submission, reviews, editor's report, decision letter, reply to reviews, revision, acceptance letter, final submission, galley proof, copyright release and published paper. Each has a characteristic form (e.g., a review template) and typical purpose. As a result, people can recognize a document as a model for possible action when they recognize from the genre of the document what they are expected to do with it [52]. Furthermore, documents related to work (and so we argue, the work itself) are organized into what are called genre systems [51], formalized sequences of documents of particular genres providing more-or-less standardized methods for recognizing what might be done and what does get done as legitimate work, as in the sequence of documents involved in publishing an academic paper.

Turning to FLOSS, we encounter a range of genres: bug reports, source code, commit messages, release notes, user documentation, requirements documents, designs and so on. On completing a piece of code, a developer invokes a specific genre of work (e.g., a push request if using GitHub). Colleagues will be able to pick up and work with the code more easily (i.e., be able to coordinate their own work stigmergically) because it invokes that genre and so comes with certain expectations.

Furthermore, a code module itself has a structure in which each component has more-or-less well-defined purposes associated with particular functionalities. In other words, there are subgenres of source code: each module of a program has its own specific purpose and so its own subgenre (e.g.,

some modules manage the interface, while others deal with interactions among data sources). In a well-structured program, the purpose of each module is clear, i.e., it is recognizable as an instance of the subgenre and thus, the module is useable by others as a model for work. In a poorly structured program, the purpose of particular module may be hard to determine or, in fact, muddled and unclear. This confusion may not directly affect the functionality of the code, but in these cases, the module does not instantiate a genre. Future developers will have difficulty adding new functionality because the current work outcomes do not make it clear how to build further.

Though the specific genres themselves are specific to a work setting, the notion of genres is general, that is, we expect to also find genres of work in data science, though perhaps not as well defined, given the emerging nature of the field. We suggest that in this domain, a genre system can be viewed as a standardized flow of work. In fact, there are two different potential workflows. First, one can view the status of a module (document) as a kind of genre. For example, within a Kanban project management context, one can view a module as flowing from "to do" to "in progress" to "validate" and then finally to "done" [2]. These phases can be seen as genres because the type of tasks that are appropriate or necessary for a module changes as one moves, for example, from "in progress" to "validate".

The second form of genre focuses on the nature of work done by the module, similar to the notion of a code subgenre. The data-science process includes a number of distinct and typified actions, such as gaining access to data sources, code to clean the data, analysis code and so on. These genres are associated with particular purposes. For example, the purpose of the code cleaning is to create a more usable data set, whereas exploratory analysis is used to provide information to data scientists about the data. Some code is used nearly exclusively by data scientists (e.g., data cleaning), while other code, such as the data analysis, is often shared between clients and data scientists, at least in terms of the results of the code. By looking at these work outputs, experienced data scientists can determine which tasks might be appropriate to do next.

A key point in the analysis of work in terms of genres is that for genres to enable documents to function as models for work they must be part of the conventions of practice shared among members of particular communities. Genres are not naturally occurring: they are rather learned as part of membership of such communities. As new participants are socialized into the communities, they gradually acquire familiarity with the prominent genres. Therefore, training will be needed in the kinds of genres that are suitable for data science and how to work with them. However, a system can provide features that support documents of different genres by making it clear first what genres exist and second what actions are appropriate for documents of different genres.

2.2.4 Combinability. A final important characteristic of work documents for stigmergic coordination is combinability. For the work to be a model for future work, the work must be combinable and improvable in modular increments [41, 48]. Most work tasks are layered and complex: new work contributions can be adjusted and added to existing outcomes. A piece of code might start out as an incomplete frame, a scaffold on which other parts get added in some organized sequence. Later, new functionality can be added to the existing structure.

Combinability in FLOSS development is supported by both cultural norms and the SCCS infrastructure. Atomic commits, mentioned above as important for understandability of code, are also important for combinability. It is easier to combine code with a focused commit than with a commit that does multiple things and touches bits and pieces of dozens of files in the process. It is likewise easier to back out a focused commit if things should go wrong. Developers are also warned: "Don't break the build", which means that the main set of files in the SCCS should always compile and run. This practice ensures that any developer who downloads the code will be able to work with it (i.e., it will be useful as a model for future work). Combinability in FLOSS development is

further supported by the SCCS's capability to merge work from different contributors. For example, developers can try out experimental enhancements on the code in a branch before committing it, or work in parallel and then merge their efforts.

Combinability of FLOSS work is greatly facilitated by the typically high modularity of the developed code. The modularity of a solution is the degree to which the components of a solution can be separated, worked on independently and recombined [63]. Modular code means that changes are often isolated to a few parts of a system, making them more atomic.

Another kind of modularity, task modularity, involves breaking up a problem into discrete chunks [47] and "building a complex product or process from smaller subsystems that can be designed and worked on independently yet function together as a whole" [6]. Task modularity is related to code modularity in the sense that in a highly modular system, developing a single module is more likely to be a separate (i.e., modular) task and developers need understand only a portion of the system to make a change. Task modularity supports complex problem-solving by enabling a team member to focus on smaller challenges, rather than needing to focus on the entire problem [7, 15]. A further benefit of task modularity is that it helps reduce the need to coordinate details of a team member's work with other team members [7]. A developer can run the code with their proposed changes and obtain direct feedback about the success or failure of their changes. This approach allows them to iteratively enhance their understanding of the task and to modify their strategy for managing dependencies between the existing analysis and what they are trying to accomplish.

With respect to data science, the use of R [1] is an example of one aspect of leveraging modularity. Specifically, the Comprehensive R Archive Network (CRAN) contains thousands of "packages" that can be installed and loaded as needed. These packages enable a team to easily combine modules developed by others, such as using an advanced machine learning module via a function call. However, by itself R does little to promote modularity of an analysis script or the task modularity of a team's work.

Contributions to data science analyses are potentially more combinable than suggested by current practice. For example, data-cleaning functionality can be used by various predictive analytic modules, which could be created either in parallel with the development of data cleaning, or at some point in the future. However, to fully encourage combinability will require analysts to develop more modular code to achieve the benefits of decomposing tasks and allowing different team members to work on different aspects of the project. Modularity is primarily a function of the development practices adopted by the team. As noted, data scientists typically think about analyses as a flow of data, and not as modules, but a modular approach can be encouraged, e.g., through training. Further, a system can provide features that promote and support development of more modular approaches to an analysis.

2.2.5 Summary. Figure 1 shows our conceptual framework. We expect that a system that implements features that implement the three sets of affordances described above (specifically affordances related to visibility, genre and combinability) will support stigmergic coordination in the team using it. These affordances work together to support stigmergic coordination. Visibility means that a user will be able to see the work that others are doing or have done in order to guide their own work. Having work products with clear genres will enable the user to comprehend the purpose of those pieces of work, which again allow them to use them to guide their own work. Finally, combinability of work means that users will face fewer issues making others' work fit with theirs and vice versa.

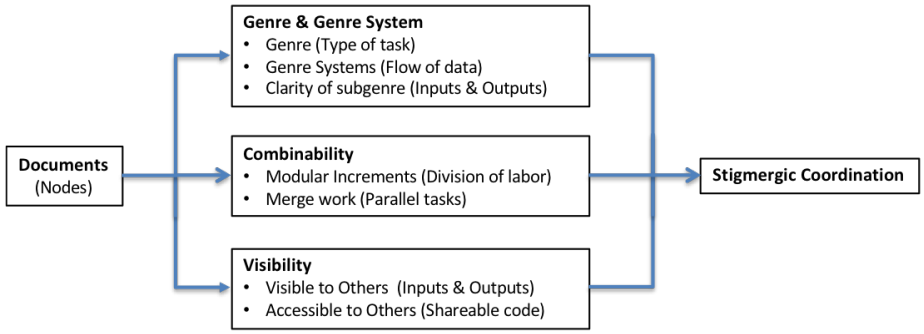

Fig. 1. Conceptual Framework for System Affordances Supporting Stigmergic Coordination

3 SYSTEM DESIGN

The previous section described three sets of socio-technical affordances that we suggest are needed for stigmergic coordination—visibility, genres and combinability—and suggested approaches to implementing each for data-science teams. In this section, we discuss in more detail the design and implementation of a data-science team coordination tool to provide these affordances.

3.1 System Overview

MIDST (Modular Interactive Data Science Tool) is a web-based data-science application that was developed for this project. The tool enables a team of data scientists to collaborate on developing an analysis, using a common data-science programming language R [1]. When using MIDST, analyses are performed by writing what are essentially programs in the R language that take data as input and output analysis results.

MIDST has three integrated views that team members use to create an analysis (or part of an analysis): the network, task and code views. Each are described below.

3.1.1 Network View. The main view of the analysis is as a workflow, in MIDST's network view. As with other data-flow tools, the network view helps users break an analysis into smaller chunks of work (nodes), and then visualize the flow of data through the nodes that comprise the analysis.

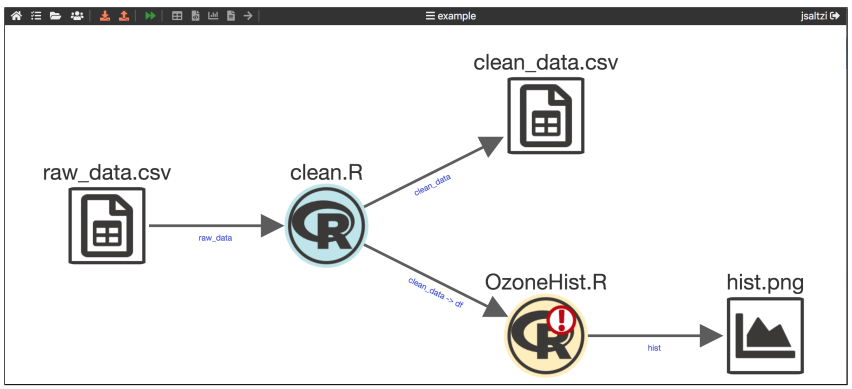

Fig. 2. Network view showing data flowing between nodes and nodes with execution errors (an "!").

There are three kinds of nodes: executable nodes that contain R code (code modules), data nodes that can be connected to an input of a node and visualization nodes that can be connected to an output. For example, Figure 2 shows a simple analysis that reads in a raw data file (the `raw_data.csv` node), cleans and saves the data file (the `clean.R` code node outputting to the `clean_data.csv` data node) and generate a histogram (the `OzoneHist.R` code node reading from the `clean.R` node and outputting to the `hist.png` visualization node).

In the network view, users can add new nodes, define a node's inputs and outputs and connect nodes together, implementing a flow of data between the nodes. As users update the network (e.g., adding nodes or connections), the changes are propagated to other users viewing the network. Users can execute the entire network in the network view by pressing the 'Run' button at the top of the network view window. Any errors that occur during execution of the network are visible as failed nodes, shown by the exclamation mark in Figure 2. Other controls push or pull code changes or change views (discussed below).

3.1.2 Task View. A second view of the node is a task view, shown in Figure 3. Similar to other task boards, such as Trello (www.trello.com), the status of each code node is indicated by the column it appears in. Users can update the status of a node by simply dragging it to a new column. Tasks (i.e., nodes) can also be created in this view, which will add them to the workflow, but without connections. MIDST's task view provides a quick overview of the project status: what is being worked on, who is working on it and the overall balance between completed and uncompleted work. Status is also shown in the network view (Figure 2): each module is colored according to its status.

3.1.3 Code View. Third, by clicking twice on a code node, within either the network or task view, a user drills down to the R code for the node. (Clicking on a data or visualization node gives a preview.) An example is shown in Figure 4, which happens to be the R code for the clean module from Figure 2. In this view the user writes, edits, runs and debugs R code to implement the required functionality for the node, similar to using RStudio, the most common interactive development environment for R. Within the code editor, a user can run the entire node or execute a single line of code (e.g., for debugging). Output from execution is shown in the bottom window pane of the code editor. This output is for the most recent run of the node, whether that was due to the full network being run in the network view, the full node being run in the code-editor view, or a single line being executed.

The node's input ports and output ports, the connections between the node and other nodes, are shown on either side of the code. An automatically-generated R preamble reads the data from the most recent run of the prior nodes and makes them available to the user's R script in variables

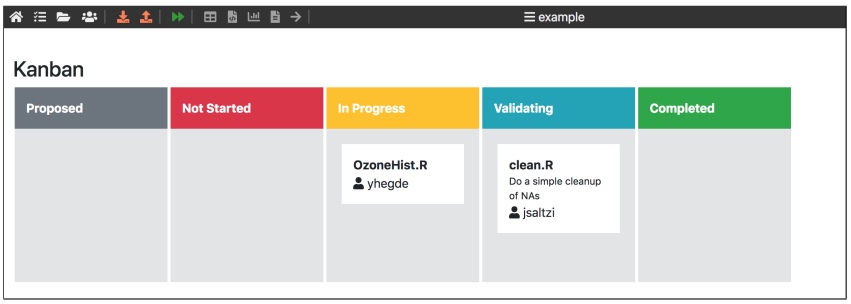

Fig. 3. Task status view

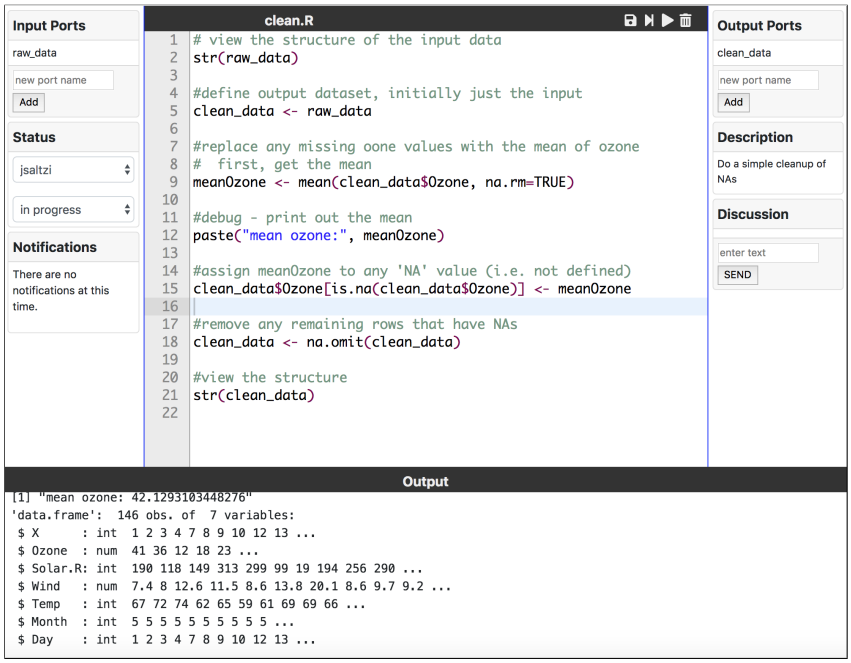

Fig. 4. Edit interface for code in a node, showing code, input and output ports and execution output

with the same name as the input ports; a postamble takes the contents of the named variables and adds them to the output ports to transfer to other nodes. The author of the R code is responsible for making the connection between these input and output variables.

The code in a code node is shared with other users viewing the project. We noted above that the system updates the view of the network dynamically: as users add nodes or connections, these changes are immediately reflected to other users. In contrast, users must explicitly share changes to code, either for one module or the entire network, using a graphical code management system that lets them "push" their updates to and "pull" others' updates from a centralized code repository. The difference between these two approaches is because as a user edits code, it is likely that their code will often be in an intermediate, non-working condition. If such changes were pushed as they are made (e.g., using a shared-editor paradigm such as Google Docs), other users would often find that they could no longer run the entire network, which could block their own work ("breaking the build"). Instead, users can wait until their code is in a usable state before sharing it and defer accepting others' changes until they are ready to incorporate them, since changes could potentially require them to update their own code to fit.

Figure 4 shows other collaboration features within the editor. For example, via the *status* widget on this view, a user can update the status and owner of a node. Team members can post messages about a node in the *discussion* widget. The MIDST system also posts messages here, e.g., to inform a user when the code being viewed is out-of-date (i.e., that another team member has shared a more recent version).

3.2 Other Design Decisions

In order to make MIDST user-friendly and easy-to-use, many user-interface design questions had to be addressed. For example, one can hover over an output item and see a preview of that data or

easily switch between views. Many of these features are to improve functionality or ease of use. However, several novel coordination-related requirements emerged as we implemented the system, which we briefly discuss in this section.

3.2.1 Node Ownership. A common problem in a source-code control system is handling multiple changes to the same document, e.g., by merging non-conflicting changes and providing an interface for a person to resolve conflicting changes. For the current version of the system, we decided to side step this problem. Since a node is assigned to particular a user, in MIDST only the owner of the node is allowed to push changes to the code. If other team members want to make changes, they can change ownership and do it, or discuss the changes with the node owner.

3.2.2 System Hints and Notifications. As noted above, users must explicit decide when to share code. To reduce the need for explicit coordination about changes, MIDST proactively suggests when a team member should push an update to the central repository as well as providing reminders when a team member needs to pull the updates from the team's central repository. MIDST notifies team members, via the discussion widget in the code view or by highlighting the icon in the network view, that another team member has pushed updated code, and also suggests, when viewing the network, that the updated code should be "pulled" from the team's repository (though as noted above, the timing is up to the user).

MIDST also helps a team member keep the status of their nodes up-to-date. For example, when someone starts to edit a node, if that node is not currently owned by another user, the system suggests that the current user own the node and move it to "in progress". If a user tries to edit a node owned by another team member, then the system reminds the user that another team member owns that node. Furthermore, when a user pushes code to share it with other team members, MIDST asks if that node's status should be moved to "validating".

3.2.3 Shared Execution Environment. Our initial implementation of the system was as an application that would run on the user's computer to provide the user interface and R session. The application shared network and code changes via GitHub. We soon discovered that sharing code was not enough to enable easy coordination. Users also need to share all the dependencies that the code relies on, such as data files and libraries (needing particular libraries can be a problem also in FLOSS development). Users not infrequently encountered problems running code provided by their teammates, e.g., file paths that needed to be changed or new libraries that needed to be installed. Also, users with less powerful computers faced performance issues.

To avoid these problems, the current version of MIDST provides access to a common computing infrastructure in which each user has a clone of the same computing environment. More specifically, the current implementation is a web application, meaning that all the R code runs on a common server. This shared execution environment greatly facilitates team collaboration since issues such as what libraries and what versions are installed as well as other details such as location of data files are eliminated. As well, it avoids a dependency on GitHub for sharing code.

3.3 MIDST's Support of Affordances for Stigmergic Coordination

Having reviewed the functionality of the tool, we next discuss how it implements the three sets of socio-technical affordances developed above.

3.3.1 Visibility. There are several ways MIDST supports visibility of code status and activity. First, the breakdown of work that is required within the project is clearly shown within both the network and task views. In addition, how the data flows through the system is visualized via the node connections in the network view. Second, MIDST shows node ownership (in the code and task views) and node status. Changes in these node attributes are visually shared in the network view

(via the colouring and other node decorations) and in the code view (via messages from a MIDST bot in the discussion widget). Third, nodes with errors are clearly shown in the network view (via the '!' icon on a node). Fourth, changes to code made by other users are highlighted as "nodes that need to be pulled" both via a message in the discussion widget (in the code view) as well as in the network view. Finally, the system shares the network structure and code with all users, making their work mobile.

3.3.2 Genres. There are three ways in which MIDST supports genres. First, MIDST supports the notion that each node has a status, which we argue is one form of genre. Second, MIDST supports different node types: input nodes, code nodes and output visualization nodes. Finally, there is an implicit set of genres of code nodes in the data-flow pipeline defined in the network view. The network view shows how data flows from, for example, reading data, to cleaning data, to exploratory analysis to more advanced machine learning to outputs. Thus, where the node is within the network provides context as to the type of work being done within that node. Collectively, these genres helps the data scientist understand the context and goal of the node, independent of the actual details within the node. Further support of genres is a goal for future development, as discussed in Section 5.2.2.

3.3.3 Combinability. Supporting combinability of work was a key goal in the design of the system as our initial investigations of data science had suggested that it was problematic. MIDST supports combinability via the tool's encouragement for users to create modular components, as well as the ability to merge different users' work (while also reducing the potential for duplication of work). Specifically, MIDST supports modularity via the network view, in which nodes of R code have clearly defined input and output ports to connect to other nodes. These input and output ports act as an interface definition, clarifying the work to be accomplished by the node. With respect to merging work, MIDST supports an easy way to push and pull code, but equally important, a way to note that a node is "owned" by a person on the team. When appropriate MIDST reminds that person to push the code to the central repository and then reminds the other team members to pull.

3.3.4 Summary. In summary, MIDST offers a shared workplace where the cooperative work is facilitated, not only by requiring active construction from the participants of a common information space in which users can perceive, access and manipulate the same set of information, but more importantly by providing a shared view. In fact, the core of the notion of a shared view is that multiple actors perceive the same object (the network of node in our case study, containing tasks, description and code or implementation) in the same state and perceive any changes in the state of the object concurrently. Any changes to the object affected by one user will be immediately perceptible to the other users.

3.4 System Implementation

The system architecture for MIDST is shown in Figure 5. The front-end of the system is a web browser. Plotting of the network workflow uses the cytoscape Javascript library. The back-end is a Python web application built using the Flask framework. The web application uses a Mongo database to store data about the network, user projects, code execution and collaboration features, and the server file system to store project files, R scripts and data files. Each user has access to his or her projects and the projects that are shared with him or her by others on the team. The execution of R code in users' R sessions is managed using the RStudio server software. Our initial deployment has been for a data-science class. In this setting, the instructor has privileges to access all projects of his or her class. The entire system is hosted on an Ubuntu server with 16 cores, 32 GB RAM and 500 GB disk space.

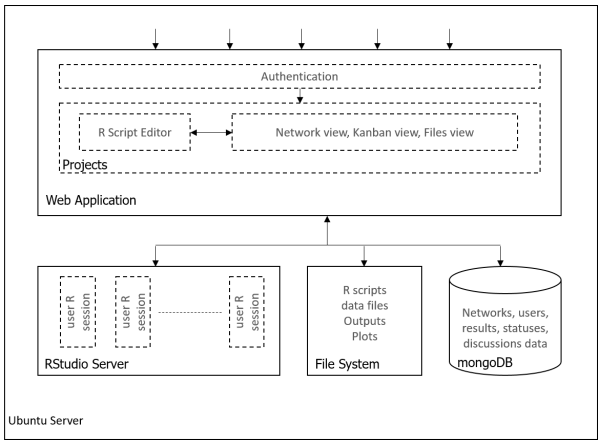

Fig. 5. MIDST System Architecture

3.5 Comparison to Other Systems

MIDST has similarities to and draws features from many existing systems, but the purpose and combination of features is unique due to the design intended to support stigmergic coordination, making the system a novel contribution. MIDST implements the following features: a workflow for data analysis, task-status tracking, R-code editing and execution, code sharing, a shared execution environment and group task support. We compare MIDST to other tools that implement these features to highlight points of similarity and difference.

A number of workflow tools have been developed to support data analysis, e.g., KNIME. Another research system StatWire [70] shares MIDST's goals of using a workflow system to promote code modularity. MIDST differs from these systems in two important respects. First, while existing tools are tools for individuals, MIDST is designed to support a team. Second, the goal of many workflow tools is to enable non-programmers to develop an analysis by combining pre-existing modules. In contrast, in MIDST, we expect users to write their own R code for the code nodes. A more production-oriented version of MIDST could provide a library of nodes, though doing so would break the current one-to-one mapping of code nodes to tasks (discussed below in Section 5.2.3).

Another feature of MIDST is to provide a way to track and make visible the status of development tasks. There are numerous systems that support task tracking, such as Trello. In contrast to those system, in MIDST the task view is directly tied to the work being done, meaning that a user can click on a task and immediately start editing the code. This connection between a task and a node also helps users to create tasks of an appropriate size and scope. And contrariwise, integration of task statuses is not a feature of other code-development tools.

In some respects, MIDST is an integrated development environment (IDE) for R development, making it comparable to IDE tools such as RStudio. Indeed, a frequent complaint from users is that its functionality is lacking compared to RStudio. However, RStudio does not support structuring an analysis in modules, which is the goal of the workflow view of MIDST. Nor is it intended to support team collaboration.

Another feature of MIDST is code sharing. There are many systems for sharing code and providing updates about the status of the code, such as GitHub. Some of these systems also integrate task tracking. Code sharing may also be integrated with an IDE to make it convenient to check in recently-edited code. However, these tools are designed to facilitate sharing of files. We argued

above that these tools do support stigmergic coordination in FLOSS. However, by themselves they are not sufficient to support stigmergy for data science as they do not provide affordances for creating modules that are meaningful for this setting, meaning that the individuals' work lacks combinability.

A final feature of MIDST is the shared execution environment. R Studio Server similarly allows users to run an R session on a server and data analysis notebooks (e.g., Jupyter [45]) can also provide access to a shared environment. Another approach to managing dependencies is a container system such as Docker [11] that bundles together a program with all of its dependencies. However, these tools do not provide support for modularizing an analysis nor for fine-grained collaboration. For example, Rule et al. [59] observe that notebooks often become difficult to navigate and understand, which discourages sharing and reuse. To try and address this challenge, they introduced the concept of annotated cell folding (i.e., the ability to hide/unhide blocks of code), which was somewhat helpful, but also caused someone new, who was trying to read the code, to sometimes overlook components of the analysis that were hidden.

Finally, there are many, many collaborative systems designed to support groups, specifically to support coordination of group work (that is, for managing dependencies among group tasks). However, only a few systems have been explicitly aimed at supporting stigmergic coordination. Musil et al. [50] proposed the concept of a Stigmergic Information System (SIS) architecture metamodel, though their goal is to develop an architectural model that describes many kinds of systems rather than to build one. Most systems described as stigmergic appear so far to simply provide access to the shared work, without specific attention to coordination of the work. For example, Zhang et al. [77] described a system for allowing collective construction of a conceptual model.

4 PRELIMINARY EXPERIENCE

We next report our findings from our preliminary experiences deploying the system. Specifically, to understand how MIDST supports team coordination, we report on an exploratory study that compared teams who carried out a data-science project using MIDST to teams that used RStudio. Observation of and informal interviews with the teams using the two different tools were augmented with data from a survey (i.e., a mixed-method approach). We emphasize though that the main contribution of this paper is the framework development and model-driven system design and development reported above, and that this evaluation is just preliminary, i.e., to indicate if we are on the right track and to guide further system development.

The participants in the study were data-science graduate students. While there has been little written about using students to gain insight into industry teams within the data-science context, experiments with students have been common for decades in the software development domain. In fact, students were used as subjects in 87% of the software development experiments analyzed over a representative ten-year period [66]. It is important to note that when using students as subjects, several factors are typically considered. First, "students vs. professionals" is actually a misrepresentation of the confounding effect of proficiency, and in fact differences in performance are much more important than differences in status [68]. Hence, master-level students, many of whom have several years of industry experience, can often be an appropriate choice for subjects and act as a proxy for junior-level professionals. Second, comparing across experimental conditions, using students may reduce variability because all students have about the same level of education, leading to better statistical characteristics [46]. Finally, while students might not be as experienced as practicing professionals, they can be viewed as the next generation of professionals and hence suitable subjects for studies [44, 60].

Before reporting on these findings about coordination, we note that initial results from a study of MIDST [61] suggest that MIDST improves the modularity of an analysis, as well as the maintainability of an analysis by making it easier to share and to understand the analysis code. Improved modularity and maintainability lead us to believe that MIDST improves combinability (due to improved modular increments) and visibility (since improved maintainability was due in part to improved accessibility and visibility of node inputs and outputs), two of the intermediate variables in our conceptual model (Figure 1).

4.1 Methodology

231 students in eleven sections of a graduate-level introductory data-science class participated in this evaluation. Nine of the eleven sections were face-to-face and two were distance-based. Students from two of the face-to-face sections and one of the distance-based sections used MIDST, a total of 72 students. The students who did not use MIDST used RStudio exclusively and served as a control group (i.e., a quasi-experimental design). Students enrolled in a section of the course without any knowledge of which sections would use MIDST and which would use only RStudio.

Students had similar experience and backgrounds across all sections. Specifically, most of the students were graduate information system students. However, each section also included students from other graduate programs, mainly business administration and public policy. All students received the same data-science and RStudio instruction and had the same project requirements, working in teams of 3 to 6 people on a semester-long data-analysis project. All students received instruction on the importance of modularity in a team project and some feedback on creating a good task division. Students who used MIDST were also given instruction on how to use MIDST prior to the start of the project and used it for some individual assignments.

The study protocol was approved by the University's IRB. Students were not compensated for their participation in the study. Use of MIDST in the MIDST sections was only mandatory for submitting assignments; students could and some did use RStudio for development, as discussed below. Students were informed that MIDST was part of a study and had the option to request that their data not be used; none exercised that option.

4.2 Usage Data

In fall 2018, the class used an earlier version of MIDST. The sections that used MIDST created a total of 10 team projects across 2 sections. The average team size was 5 students. The average number of nodes per team project was 29.4; the average number of R nodes was 10.6. RStudio does not require creating a modular analysis, and hence, there are no comparable counts for the control group.

In spring 2019, an online class used the current version of the tool. The class created a total of 6 team projects. The average team size was 3.2 students. The average number of nodes per team project was 38.7; the average number of R nodes was 11.8.

We expect that MIDST will be much more useful for the distance students, who have fewer opportunities for interaction and coordination. However, it was very useful in debugging the system to be able to observe and interact with students to understand their problems and to provide support, hence our decision to first use the system with students in face-to-face sections.

4.3 Findings from Observations

We first report findings from observations of and informal interviews with student team members while they were working on the team project. The observations were done by one of the authors siting in on classes and observing the students working together in small groups, either face-to-face or in a synchronous video conference, while taking notes. This observer was not an instructor for the class. During the observations, the observer would occasionally ask the students to explain

what they were doing or what problems they perceived. Analysis of the observations are integrated across the two semesters. As appropriate, we note if an example was within the face-to-face or distance-based course.

In reviewing the observations, several themes emerged. First, the project required interaction among the students to build a common understanding of the task. Students in the face-to-face class were able to mobilize all the communicative resources of face-to-face interaction to negotiate a shared understanding of what is to be done. To communicate with each other, students generally met physically to discuss the main point of their project. During their meeting, students were observed trying to make sure that everyone in the group understood all parts of the project. Students tried to help others understand their ideas, and at the same time, felt free to ask for help from other team members. Students in the distance section faced more of a challenge, but did have some more limited opportunities for discussion. In both cases, synchronous discussions were augmented with other interactions, e.g., email or a chat group.

In both cases, it was useful to have shared notes to ensure that all members had a common understanding. This was especially important when some members were absent from a discussion. MIDST complemented such unstructured notes by enabling team members to share their understanding of the work to be done, e.g., by giving a meaningful title and a detailed description for each node. As one student said "we note our comments in a shared document or through a group discussion where we describe our fixed goal and the different tasks. However node title and the brief description give me a quick access to the updated goal".

Beyond easier code sharing, MIDST requires each node to have defined task inputs (which can be the output of another team member's node) and output (which can be the input for another team member's node). Having well-defined inputs and outputs helped team members coordinate their work without discussion. As one student said, "Sometimes we don't ask for resources we just wait for it by looking directly to the needed input from the network." The tool also provides access to the shared code, which also supports coordination. As one student said, "I don't find problems or issues in coordinating the work. I generally go to the code of each resource and look if it is similar to my expectation. Sometime when the code is clear and well commented, I go directly for modifying it (sometime after permission)."

The combination of defined inputs and outputs and node ownership also made explicit coordination easier when it was needed. For example, if students who were waiting for an input from another node's output found that the result was different from what was expected, they knew who to talk to. Ownership suggests that any modification to the node should be discussed with and validated by the owner. As one said, "If they want to improve one task, they should first ask me and then they go to the code and update it."

The notification system was also seen as helpful in supporting coordination activities. Indeed, students expressed a desire for additional notifications indicating the creation of a new node, the deletion or the update of existing one or the presence of another student in the node to help tracking activities related to their own tasks. For instance, one complained, "I'm not notified about other team member working in the same node and delete it".

In contrast, students using RStudio lacked a shared virtual workplace, which limited access to information about the project. They used Trello to track tasks and task status. To avoid problems, they tried usually to finish their work on time and post in Trello the status of the work so they knew when they could ask for the output of another student's task in case of dependencies. However, sharing the code was a challenge for these teams. Some teams were observed using Google Docs. But since the code was not easily shared, the team members typically preferred to divide the project in subtasks and work in subgroups of 2 or 3 people, where members in a subgroup can meet

frequently and work on tasks that are independent from the other groups within the team. This approach minimizes the need for coordination but potentially increases problems integrating code.

Overall, the majority (but not all) of the teams using MIDST succeeded in modularizing the project and creating the network flow according to their understanding. MIDST seemed easy-to-use as a way to understand, explain and share team members' project status. All teams agreed that the tool helped them to understand and track project status as the project progressed. A counter-example is provided by one team in the distance course who encountered significant team issues (indeed, the team actually disbanded just before the end of the project). While there were likely many causes for the team's failure to work well together, one interesting aspect of how they worked was that one of the team members did not do his work within MIDST, but rather used RStudio. When that person tried to integrate their work with the others at the last minute (i.e., a "code dump"), there was significant confusion about what was done and who was doing which analysis. A further complication is that the team member used some advanced techniques that the other team members did not know how to interpret (or even if the results were useful). So even though there was significant effort expended by each team member, there was frustration that the individual work did not contribute to the group effort. We believe that had all the team members used MIDST more proactively during the project, the structure of the nodes (including inputs and outputs) would have helped to structure their discussion about what was being done, perhaps avoiding these problems.

4.4 Survey Analysis

To augment our findings from the observation, we deployed a survey that included quantitative and open-ended questions about visibility and overall coordination when using MIDST or RStudio. The survey was deployed near the end of the class. There were 197 responses to the survey (an 85% response rate). 61 of those students were in the MIDST condition (also an 85% response rate). The data collected served to verify the reliability of the quantitative scales. However, we note that the survey was completed by students who used an earlier, still somewhat buggy version of the system and by only a handful of distance students, those we believe to have a larger need for the system. And even there, not all of the team used MIDST as intended, further compromising the results. We therefore focus here on the qualitative analysis.

The qualitative responses were analyzed by one of the authors, who read through them to look for commonalities in the responses. Four key themes emerged. On the one hand, students appreciated MIDST's ability to help them break up the work and stay coordinated within their team. On the other hand, they also often commented on bugs, usability and performance issues. One student comment was typical "Though a thoughtful idea, it needs polishing".

Below we list examples of student comments for the four key themes identified.

(1) MIDST good for sharing/dividing work:
   - "It is a very good tool for shared projects"
   - "Task assignment was easy"
   - "MIDST allowed us to divide the work and track the progress of the project"
   - "We could all work in it simultaneously without asking each other to share the code with us"
   - "Different versions of code can be made available"
   - "Module wise distribution, easy sharing"
   - "Very good for assigning tasks and group coordination"

(2) MIDST good for visibility/tracking:
   - "We can track team members work"

- "Easily see the progress"
- "It can track each member's progress"

(3) MIDST bugs/issues:
- "The basic functions (like ggplot) stop working abruptly"
- "Running the code takes a lot of time"
- "The session timeouts for big networks"
- "Created a lot of errors on the codes which were working well on R studio"

(4) MIDST ease-of-use issues:
- "Took us a little time to learn the working"
- "Identifying and solving errors became extremely challenging for our team"
- "The time taken to understand it completely"
- "We didn't have any prior experience working with MIDST, so it took some time in getting familiar with MIDST"

In summary, while still preliminary, the evidence from the initial use of the MIDST system suggests that it is supporting better modularity and combinability of code and improving visibility and mobility of work products, which we have argued are necessary preconditions for stigmergic coordination.

5 CONCLUSION

Our goal in this project is to better support coordination in data-science teams by transferring findings about coordination from FLOSS development. Specifically, we sought to enable stigmergic coordination, meaning coordination that is carried out through a shared work product. We noted that simply reusing systems (e.g., using git and GitHub to share R scripts) was not entirely satisfactory due to mismatches due to differences in work products and practices. To understand what support was missing, we developed a conceptual framework for the affordance needed to support stigmergic coordination, specifically, visibility of work, use of clear genres of work products and combinability of contributions. We then sought to implement those affordances for data science. While the success of the final system in supporting stigmergic coordination remains to be proven, our initial experience suggests that it does implement the desired affordances and seems to be useful.

5.1 Future Empirical Studies

At present, the evaluation presented in this paper is only indicative that the tool is useful and that it supports stigmergic coordination. We plan to continue using the tool with future classes to gather a larger evidence base with which to test the conceptual framework. The quantitative study results at least established the reliability of the scales, which is an important precondition for future research using them. We expect to see that teams using MIDST will be able to coordinate at least some of their work stigmergically, thus expending less work on explicit coordination. Failure to achieve easier coordination would falsify the framework proposed here and suggest the need to explore additional affordances. We would also like to use the tool with teams beyond the classroom. For example, we could use the tool with teams that are competing in data-science challenges, such as Kaggle. Such studies could establish how well the system scales to large analyses and teams.

5.2 Future System Features

While the system is usable and seemingly useful in its current state, we have plans for additional features. There are a number of small usability improvements, but also four major changes that touch on the affordances developed above.

5.2.1 Encapsulation. At present, MIDST implements modularity incompletely, since it does not enforce encapsulation. All of the code is run in a common R session, which means that a node can use variables that are created in other nodes. At present, use of global variables is not uncommon, e.g., when a user develops a script in RStudio and then copies the code into MIDST nodes. But relying on global variables is a bad idea, as it creates dependencies between nodes that are not visible in the overview. In the worst case, results might be dependent on the order in which nodes are executed. A second problem caused by the lack of encapsulation is that in the current implementation, port names must be unique in the network, which adds a cognitive overload to creating code nodes. Adding better encapsulation should be straightforward.

5.2.2 Genres. Second, MIDST currently does not implement genres in a deep way, in part because the genre repertoire for data science is still emergent. Future work might further develop different genres of analysis tasks to guide users in structuring their analyses. For example, the system could have a template for a typical data-cleaning node or regression node. Such a template could implement a simple default such that the network would be executable immediately (e.g., for a cleaning node simply copying the input data to the output). It could further suggest what actions are typically part of such a node, e.g., including necessary regression diagnostics as part of a regression node.

5.2.3 Tasks vs. Nodes. Third, a major limitation of the current system is the one-to-one mapping of tasks to code nodes. This approach is suitable for the current application where students are developing a new analysis, meaning that the work needed is to develop a node. However, it would likely not be appropriate for a team maintaining an analysis workflow, where a task to be done might affect multiple nodes. We need to rethink the interface and system functionality to represent tasks that are not connected to a single node.

5.2.4 Code Merging. Finally, as noted above, to avoid conflicting changes that might need to be manually merged the system currently allows only the owner of a node to push code changes. We would like to relax this restriction, especially to support tasks that touch multiple nodes, but also to let users more easily make small changes to others' nodes. To do so will require implementing a mechanism to handle multiple change to the same node, including the ability for a user to manually merge conflicting changes. Such algorithms are already implemented in modern SCCS, so we should be able to adapt the approaches taken by these systems.

5.3 Reflections on Theory-based System Development

To conclude, we reflect on the process of building conceptual frameworks to guide system development. Though CSCW researchers increasingly have the opportunity to study systems that others have developed, systems development remains an important approach to research. We suggest that most if not all system development is based on at least an implicit theory of how the system functionality will affect users. Explicit appeals to theory are less common, though not absent (e.g., [8, 64]). Sometimes the theory is normative, i.e., a belief that if users are made to behave in certain ways their performance will improve. For instance, drawing on speech act theory [4, 65], an early CSCW system, the "Coordinator" [32, 74], sought to improve coordination by requiring that communication be explicit about the coordination requested, eliminating possible misunderstandings about the sender's intent. Alternately, as in our case, the theory may be descriptive, e.g., an observation that certain behaviours are supported by particular kinds of information that a system might provide.

Building a system provides an opportunity to test whether the theoretical understanding is correct. Returning to the Coordinator example, while it might have been true that being more

explicit was more efficient, experience with the system showed that people generally preferred making indirect requests. Our future plans for MIDST include an empirical test of the system that will serve as a test of the generality of conceptual framework presented here (it is already supported by research on FLOSS development).

An advantage of explicitly stating the theory behind a system is that the theory can help clarify what might be missing in supporting the application domain. In other words, the conceptual framework provides a set of design requirements for systems that would support stigmergic coordination. In the data-science case, we noted in particular problems with making the work contributions of data-science team members more easily combinable than they are as simple R scripts. The quest to improve combinability led to development of system support and instruction in creating modular code to make the work more atomic and so easier to combine.

A disadvantage of a theory-based approach is that it can be more time consuming if the theory being drawn on does not already exist. In our case, time was need to study FLOSS and to develop a conceptual framework based on that research, rather than simply investigating problems faced by data-science teams. However, investment in theory building should show benefits in the future because the theory can be applied to other domains.

As a closing example, we are interested in better supporting stigmergic coordination in Wikipedia. We note that the Wikipedia infrastructure provides direct support for stigmergic coordination: a logged-in user can set a watchlist, which provides a page with recent changes made to the pages being watched. Using the watchlist allows an editor to react to changes without any need to be alerted by or to discuss with the other editor.

In a study of Wikipedia editing [57], we found that the majority of edits to two example articles were not associated with discussion on the article's Talk page, suggesting that these changes were coordinated stigmergically. Specifically, minor fixes and vandalism fixing did not seem to require discussion. However, we also found that Talk posts did seem to be related to article quality, suggesting the continued importance of explicit coordination of at least some edits.

Drawing on the theory presented here, we speculate that the issue is that work in Wikipedia is shared as a set of characters changed in an article and as a result, a change does not necessarily have regularities of form that give hints to purpose, i.e., a genre. A few types of changes do seem to constitute genres, e.g., small changes to fix typos or grammar or reversion of vandalism, and as we noted, these kinds of changes seem to be amenable to stigmergic coordination. However, it can be difficult to identify the purpose behind more significant changes simply be examining the characters affected. As a result, the changes by themselves do not always provide a model for future work in the absence of discussion. It may simply be the case that Wikipedia edits are too diverse to be easily categorized, but future research might examine both the genres of Wikipedia edits and ways to embody those genres in tools, which might increase the range of editing that can be coordinated stigmergically.

ACKNOWLEDGMENTS

This work was partially supported by a grant from the US National Science Foundation, IIS 16–18444.

REFERENCES

[1] [n. d.]. The R Project for Statistical Computing. https://www.r-project.org/
[2] David J. Anderson. 2010. *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press.
[3] Oliver Arafat and Dirk Riehle. 2009. The commit size distribution of open source software. In *Hawaii International Conference on System Sciences (HICSS-42).* 1–8.
[4] John Langshaw Austin. 1962. *How to Do Things with Words.* Harvard University, Cambridge.
[5] Mikhail Mikhailovich Bakhtin. 1986. The problem of speech genres. In *Speech Genres and Other Late Essays: M.M. Bakhtin*, Caryl Emerson and Michael Holquist (Eds.). University of Texas Press, Austin, 60–102.

[6] Carliss Y. Baldwin and Kim B. Clark. 1997. Managing in an age of modularity. *Harvard Business Review* (1997), 84–93. Issue September/October.

[7] Carliss Y. Baldwin and Kim B. Clark. 2006. Modularity in the design of complex engineering systems. In *Complex engineered systems*. Springer, 175–205.

[8] Abraham Bernstein. 2000. How can cooperative work tools support dynamic group process? Bridging the specificity frontier. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '00)*. ACM, 279–288. https://doi.org/10.1145/358916.358999

[9] Ethan S Bernstein. 2012. The transparency paradox: A role for privacy in organizational learning and operational control. *Administrative Science Quarterly* 57, 2 (2012), 181–216.

[10] Anant Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron Elmore, Samuel Madden, and Aditya Parameswaran. 2015. Datahub: Collaborative data science and dataset version management at scale. *Biennial Conference on Innovative Data Systems Research (CIDR)* (2015).

[11] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *SIGOPS Operating Systems Review* 49, 1 (Jan. 2015), 71–79. https://doi.org/10.1145/2723872.2723882

[12] Francesco Bolici, James Howison, and Kevin Crowston. 2016. Stigmergic coordination in FLOSS development teams: Integrating explicit and implicit mechanisms. *Cognitive Systems Research* 38 (2016), 14 – 22. https://doi.org/10.1016/j.cogsys.2015.12.003 Special Issue of Cognitive Systems Research – Human-Human Stigmergy.

[13] Geoffry C. Bowker and Susan Leigh Star. 1994. Knowledge and information in international information management: Problems of classification and coding. In *Information Acumen: The Understanding and Use of Knowledge in Modern Business*, L. Bud-Frierman (Ed.). Routledge, London, 187–213.

[14] Geoffry C. Bowker and Susan Leigh Star. 1999. *Sorting Things Out: Classification and Its Consequences*. MIT Press, Cambridge.

[15] Stefano Brusoni, Luigi Marengo, Andrea Prencipe, and Marco Valente. 2007. The value and costs of modularity: A problem-solving perspective. *European Management Review* 4, 2 (2007), 121–132.

[16] John M. Carroll, Dennis C. Neale, Philip L. Isenhour, Mary Beth Rosson, and D. Scott McCrickard. 2003. Notification and awareness: Synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies* 58, 5 (2003), 605–632. https://doi.org/10.1016/S1071-5819(03)00024-7

[17] John M. Carroll, Mary Beth Rosson, Gregorio Convertino, and Craig H. Ganoe. 2006. Awareness and teamwork in computer-supported collaborations. *Interacting with Computers* 18, 1 (2006), 21–46. https://doi.org/10.1016/j.intcom.2005.05.005

[18] Lars Rune Christensen. 2007. Practices of stigmergy in architectural work. In *Proceedings of the 2007 International ACM Conference on Supporting Group Work (GROUP '07)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/1316624.1316627

[19] Lars Rune Christensen. 2013. Stigmergy in human practice: Coordination in construction work. *Cognitive Systems Research* 21 (2013), 40–51.

[20] Lars Rune Christensen. 2014. Practices of stigmergy in the building process. *Computer Supported Cooperative Work (CSCW)* 23, 1 (2014), 1–19. https://doi.org/10.1007/s10606-012-9181-3

[21] L. Colfer and Carliss Baldwin. 2010. *The Mirroring Hypothesis: Theory, Evidence and Exceptions*. Report. Harvard Business School.

[22] Kevin Crowston, Qing Li, Kangning Wei, U. Yeliz Eseryel, and James Howison. 2007. Self-organization of teams for free/libre open source software development. *Information and Software Technology* 49, 6 (2007), 564–575. https://doi.org/10.1016/j.infsof.2007.02.004

[23] Kevin Crowston, Kangning Wei, Qing Li, U. Yeliz Eseryel, and James Howison. 2005. Coordination of free/libre open source software development. In *Proceedings of the International Conference on Information Systems (ICIS)*. Las Vegas, NV, USA. https://doi.org/10.1145/1029997.1030003

[24] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1277–1286. https://doi.org/10.1145/2145204.2145396

[25] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2013. Leveraging transparency. *IEEE Software* 30, 1 (2013), 37–43. https://doi.org/10.1109/MS.2012.172

[26] Laura Dabbish, Colleen Stuart, Jason Tsay, and James Herbsleb. 2014. *Transparency and coordination in peer production*. Report. http://arxiv.org/abs/1407.0377

[27] M. Das, R. Cui, D. R. Campbell, G. Agrawal, and R. Ramnath. 2015. Towards methods for systematic research on big data. In *2015 IEEE International Conference on Big Data (Big Data)*. 2072–2081. https://doi.org/10.1109/BigData.2015.7363989

[28] Paul Dourish and Victoria Bellotti. 1992. Awareness and coordination in shared workspaces. In *Proceedings of the Conference on Computer-supported Cooperative Work (CSCW '92)*. ACM, New York, NY, USA, 107–114. https://doi.org/10.1145/143457.143468

[29] Mark Elliot. 2006. Stigmergic collaboration: The evolution of group work. *m/c journal* 9, 2 (2006). http://journal.media-culture.org.au/0605/03-elliott.php

[30] Thomas Erickson and Wendy A. Kellogg. 2000. Social translucence: An approach to designing systems that support social processes. *ACM Transactions on Computer-Human Interaction* 7, 1 (2000), 59–83. https://doi.org/10.1145/344949.345004

[31] J. A. Espinosa and F. Armour. 2016. The Big Data analytics gold rush: A research framework for coordination and governance. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS)*. 1112–1121. https://doi.org/10.1109/HICSS.2016.141

[32] Fernando Flores, Michael Graves, Brad Hartfield, and Terry Winograd. 1988. Computer systems and the design of organizational interaction. *ACM Transactions on Office Information Systems* 6, 2 (1988), 153–172.

[33] Jing Gao, Andy Koronios, and Sven Selle. 2015. Towards a process view on critical success factors in big data analytics projects. In *Proceedings of the Americas Conference on Information Systems*.

[34] N. W. Grady, M. Underwood, A. Roy, and W. L. Chang. 2014. Big Data: Challenges, practices and technologies: NIST Big Data Public Working Group workshop at IEEE Big Data 2014. In *2014 IEEE International Conference on Big Data (Big Data)*. 11–15. https://doi.org/10.1109/BigData.2014.7004470

[35] Pierre-Paul Grassé. 1959. La reconstrution du nid et les coordinations inter-individuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie: Essai d'interprétation du comportament de termites constructeurs. *Insectes Sociaux* 6, 1 (1959), 41–80. https://doi.org/10.1007/BF02223791

[36] Carl Gutwin and Saul Greenberg. 2002. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)* 11, 3-4 (2002), 411–446.

[37] Robert Heckman, Kevin Crowston, U. Yeliz Eseryel, James Howison, Eileen Allen, and Qing Li. 2007. Emergent decision-making practices in free/libre open source software (FLOSS) development teams. In *IFIP International Conference on Open Source Systems*. Springer, 71–84.

[38] Robert Heckman, Kevin Crowston, Qing Li, Eileen Allen, U. Yeliz Eseryel, James Howison, and Kangning Wei. 2006. Emergent decision-making practices in technology-supported self-organizing distributed teams. In *Proceedings of the International Conference on Information Systems (ICIS)*. 43.

[39] Francis Heylighen. 2007. Why is open access development so successful? Stigmergic organization and the economics of information. In *Open Source Jahrbuch 2007*, Bernd Lutterbeck, Matthias Bärwolff, and Robert A. Gehring (Eds.). Lehmanns Media, Berlin.

[40] James Howison. 2009. *Alone Together: A Socio-Technical Theory of Motivation, Coordination and Collaboration Technologies in Organizing for Free and Open Source Software Development.* Doctoral Dissertation. Syracuse University.

[41] James Howison and Kevin Crowston. 2014. Collaboration through open superposition: A theory of the open source way. *MIS Quarterly* 38, 1 (2014), 29–50.

[42] H V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. 2014. DataHub: Collaborative data science and dataset version management at scale. *Commun. ACM* 57, 7 (2014), 86–94.

[43] Eirini Kalliamvakou, Daniela Damian, Leif Singer, and Daniel M German. 2014. *The code-centric collaboration perspective: Evidence from GitHub.* Report. Technical Report DCS-352-IR, University of Victoria. http://thesegalgroup.org/wp-content/uploads/2014/04/code-centric.pdf

[44] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering* 28, 8 (2002), 721–734. https://doi.org/10.1109/TSE.2002.1027796

[45] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks: A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas: Proceedings of the International Conference on Electronic Publishing*, F. Loizides and B. Schmidt (Eds.). 87–90.

[46] Andrew J. Ko, Thomas D. LaToza, and Margaret M. Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering* 20, 1 (01 Feb 2015), 110–141. https://doi.org/10.1007/s10664-013-9279-3

[47] Richard N. Langlois. 2002. Modularity in technology and organization. *Journal of Economic Behavior & Organization* 49, 1 (2002), 19 – 37. https://doi.org/10.1016/S0167-2681(02)00056-2

[48] Bruno Latour. 1990. Visualisation and cognition: Drawing things together. In *Representation in Scientific Practice*, M. Lynch and S. Woolgar (Eds.). MIT Press, Cambridge.

[49] Julia Lowndes, Benjamin Best, Courtney Scarborough, Jamie Afflerbach, Melanie Frazier, Casey O'Hara, Ning Jiang, and Benjamin Halpern. 2017. Our path to better science in less time using open data science tools. *Nature Ecology and Evolution* 1, 6 (2017).

[50] Juergen Musil, Angelika Musil, and Stefan Biffl. 2014. Towards a coordination-centric architecture metamodel for social web applications. In *European Conference on Software Architecture*. Springer, 106–113.
[51] Wanda J. Orlikowski and JoAnne Yates. 1994. Genre repertoire: The structuring of communicative practices in organizations. *Administrative Science Quarterly* 33 (1994), 541–574.
[52] Carsten Østerlund. 2007. Genre combinations: A window into dynamic communication practices. *Journal of Management Information Systems* 23, 4 (2007), 81–108.
[53] Carsten Østerlund. 2008. Documents in place: Demarcating places for collaboration in healthcare settings. *Computer Supported Cooperative Work (CSCW)* 17, 2–3 (2008), 195–225.
[54] Carsten Østerlund. 2008. The materiality of communicative practice: The boundaries and objects of an emergency room genre. *Scandinavian Journal of Information Systems* 20, 1 (2008), 7–40.
[55] Carsten Østerlund, Steve Sawyer, and Elizabeth Kazianus. 2010. Documenting work: A methodological window into coordination in action. In *26th Conference of the European Group for Organizational Studies (EGOS)*.
[56] H. V. Parunak. 2006. A survey of environments and mechanisms for human-human stigmergy. In *Environments for Multi-Agent Systems II*, D. Weyns, H. V. D. Parunak, and F. Michel (Eds.). Lecture Notes in Artificial Intelligence, Vol. 3830. 163–186. https://doi.org/10.1007/11678809_10
[57] Amira Rezgui and Kevin Crowston. 2018. Stigmergic coordination in Wikipedia. http://www.opensym.org/wp-content/uploads/2018/07/OpenSym2018_paper_34.pdf
[58] Alessandro Ricci, Andrea Omiciniand Mirko Viroli, Luca Gardelli, and Enrico Oliva. 2007. Cognitive stigmergy: Towards a framework based on agents and artifacts. In *Environments for Multi-Agent Systems III*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.). Lecture Notes in Computer Science, Vol. 4389. Springer, 124–140. https://doi.org/10.1007/978-3-540-71103-2_7
[59] Adam Rule, Ian Drosos, Aurélien Tabard, and James Hollan. 2018. Aiding collaborative reuse of computational notebooks with annotated cell folding. *Proceedings of the ACM on Human-Computer Interaction* 2(CSCW), 150 (Nov. 2018).
[60] I. Salman, A. T. Misirli, and N. Juristo. 2015. Are Students Representatives of Professionals in Software Engineering Experiments?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 666–676. https://doi.org/10.1109/ICSE.2015.82
[61] Jeff Saltz, Robert Heckman, Kevin Crowston, Sangseok You, and Yatish Hegde. 2019. Helping data science students develop task modularity. In *Proceedings of the 52nd Hawai'i International Conference on System Sciences (HICSS-52)*. http://hdl.handle.net/10125/59549
[62] J. S. Saltz and I. Shamshurin. 2016. Big data team process methodologies: A literature review and the identification of key factors for a project's success. In *2016 IEEE International Conference on Big Data (Big Data)*. 2872–2879. https://doi.org/10.1109/BigData.2016.7840936
[63] Melissa A Schilling. 2000. Toward a general modular systems theory and its application to interfirm product modularity. *Academy of Management Review* 25, 2 (2000), 312–334.
[64] Kjeld Schmidt and Carla Simonee. 1996. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work: The Journal of Collaborative Computing* 5 (1996), 155–200.
[65] John R. Searle. 1969. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University, Cambridge.
[66] D. I. K. Sjoeberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N. . Liborg, and A. C. Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering* 31, 9 (Sep. 2005), 733–753. https://doi.org/10.1109/TSE.2005.97
[67] Dorothy E. Smith. 2005. *Institutional Ethnography: A Sociology for People*. AltaMira Press, Oxford.
[68] Z. Soh, Z. Sharafi, B. Van den Plas, G. C. Porras, Y. Guéhéneuc, and G. Antoniol. 2012. Professional status and expertise for UML class diagram comprehension: An empirical study. In *20th IEEE International Conference on Program Comprehension (ICPC)*. 163–172. https://doi.org/10.1109/ICPC.2012.6240484
[69] H. Colleen Stuart, Laura Dabbish, Sara Kiesler, Peter Kinnaird, and Ruogu Kang. 2012. Social transparency in networked information exchange: A theoretical framework. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 451–460. https://doi.org/10.1145/2145204.2145275
[70] Krishna Subramanian, Johannes Maas, Michael Ellers, Chat Wacharamanotham, Simon Voelker, and Jan Borchers. 2018. StatWire: Visual Flow-based Statistical Programming. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18)*. ACM, New York, NY, USA, Article LBW104, 6 pages. https://doi.org/10.1145/3170427.3188528
[71] Lucy A. Suchman. 1993. Technologies of accountability: Of lizards and aeroplanes. In *Technology in Working Order*, G. Button (Ed.). Routledge, London.
[72] Lucy A. Suchman. 1995. Making work visible. *Commun. ACM* 38, 9 (1995), 56–65.
[73] Luca Tummolini and Cristiano Castelfranchi. 2007. Trace signals: The meanings of stigmergy. In *Environments for multi-agent systems III*, Danny Weyns, H. Van Dyke Parunak, and Fabien Michel (Eds.). Springer, 141–156. https:

//doi.org/10.1007/978-3-540-71103-2_8
[74] Terry Winograd. 1987. A language/action perspective on the design of cooperative work. *Human–Computer Interaction* 3 (1987), 3–30.
[75] Joanne Yates and Wanda J. Orlikowski. 1992. Genres of organizational communication: A structurational approach to studying communication and media. *Academy of Management Review* 17, 2 (1992), 299–327.
[76] Manuel Zacklad. 2006. Documentarisation processes in documents for action (DofA): The status of annotations and associated cooperation technologies. *Computer Supported Cooperative Work (CSCW)* 15, 2-3 (2006), 205–228.
[77] Wei Zhang, Haiyan Zhao, Yi Jiang, and Zhi Jin. 2015. Stigmergy-based construction of internetware artifacts. *IEEE Software* 32, 1 (2015), 58–66. https://doi.org/10.1109/ms.2014.133

Received April 2019; revised June 2019; accepted August 2019