Exploring Interrelationships among Project Entities to Support Coordination in Distributed Teams

Anita Sarma and Jim Herbsleb

November 03, 2008
CMU-ISR-08-138

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper is geared to start a discussion about what data to preserve and analyze to facilitate collaborative production tasks. We do so by representing interrelationships among different project entities as networks and combining these networks using the concept of Meta Matrix – a methodology for combining individual networks to create derived networks to investigate. For illustration, we present how socio-technical dependencies, task dependencies, and knowledge networks can be constructed using the Meta Matrix. We conclude by showing how we used validated the feasibility of Meta Matrix by presenting Tesseract, a socio-technical browser that models a subset of individual and derived networks for a software project. It specifically, captures relations between developers, artifacts, and issues/bugs which is then displayed via a set of four juxtaposed, cross-linked displays.

This effort is partially funded by the National Science Foundation under grant number IIS-0534775, IIS-0329090, and the Software Industry Center and its sponsors, particularly the Alfred P. Sloan Foundation. Effort also supported by a 2007 Jazz Faculty Grant. The views and conclusions are those of the authors and do not reflect the opinions of any sponsoring organizations/agencies.

Keywords: Software development, Congruence, Tesseract

1 INTRODUCTION

Collaboration frequently occurs around production tasks, i.e., where individuals, groups, or organizations work together to pro-duce an artifact of value. Data generated during collaborative production can be employed in a variety of ways to create a better understanding of the development process and the team, and to provide useful collaborative functionality. Two examples are (1) automatic generation of social networks based on archived communication records and (2) automatic interpretation of which artifacts are associated or dependent on which other artifacts so as to create an understanding of the subtleties of artifact structures that are otherwise invisible. Further, these social and technical relationships can be linked together to create a rich socio-technical information space that can provide deeper insights into the production process. For example, determining which developers are tied to which other developers based on the underlying dependencies in the artifacts that they edit; or, locating experts based on their past experiences in the project; or, determining whether developers who work on interdependent artifacts communicate with each other.

Mining archived data to gain such understandings of the team and its work has been especially popular in software development. This is so, because software engineering tools archive sufficient data to serve as a rich source for studying and supporting collaboration. For example, communication patterns that are logged through discussion forums may be used to determine emergent teams and communities [7]. As another example, dependencies among artifacts and concurrent edits that are logged by change management systems can be used to create contextual awareness. Similarly, interdependencies among artifacts can be used to determine which developer needs to communicate with whom [2]. Several software engineering tools employ such data for locating experts [7, 8], finding relevant artifacts [3], providing workspace awareness, and managing tasks [6].

Here, we attempt to take the lessons learned about tool-generated data in software engineering and apply them more broadly. It is important that we do so now, as software delivery modes are be-ginning to fundamentally change how people perform many kinds of collaborative work. Suites of tools such as Google docs and apps, customer relationship management tools such as Sales-force.com, and many other kinds of functionality are being redesigned to be accessible through a browser (often enhanced with new forms of programming such as AJAX). This way of delivering functionality is often called “software as a service” in contrast to software that is delivered as applications installed on individual machines, which is currently the dominant delivery mode.

Delivering software as a service is an impending sea change not just in how software vendors do business, but in the opportunities for studying and supporting collaboration. When users use individual applications on their personal machines, files are stored and accessed in many locations. Further, communication, awareness, editing, and computational tools are all typically used independently of each other, with no links among the artifacts created, nor links among the work done with different applications. Contributions of different people to an artifact are not recorded and preserved in useful forms. So, for example, a team collaborating on a report may use word processing, spreadsheet, and graphics editing applications, while communicating by e-mail, IM, IP phone, and application sharing sessions. The traces left behind are, at best multiple versions of different types of files on the team members’ machines, along with mail stored in various accounts potentially on many servers and on local machines, and perhaps some manual

archiving of IMs. This disconnected jumble of in-formation spread across the team is not particularly useful.

Contrast this with what could be available if all data is maintained centrally, and all work is done with an integrated tool suite, as is likely with software delivered as a service. The complete history of every artifact – every change made, along with information about who made what change when – is available. Links among artifacts can be maintained as when, for example, numbers copied from a spreadsheet are pasted into a report. Links between arti-facts and communications can often be created automatically when messages contain links to artifacts, or explicitly refer them.

As a way of beginning to think through the question of what data to preserve, and how to capture it, we present a novel way of investigating interrelationships between different entities in a development project to better understand the kinds of data that is useful to support coordination needs. The interrelationships among entities such as developers, artifacts, and tasks (e.g., developer-to-developer, developer-to-artifact, or developer-to-task) or a combination of these relationships can provide answers to many coordination questions (see Table 1). Each relationship can be represented as a network and different networks can be combined to create derived networks that provide deeper insights into coordination needs. Here, we describe the kinds of data that need to be collected to create these networks and describe the Meta Matrix, a methodology for storing and combining networks.

In the rest of the paper, we first describe the Meta Matrix and the data needed to create it. We then discuss how individual networks can be combined to create derived networks (e.g., socio-technical dependency, task dependency, and knowledge networks). Finally, we conclude by showing how we used the concept of Meta Matrix to create Tesseract, a socio-technical browser for visually exploring socio-technical relations in a software project.

2 COORDINATION META MATRIX

Identification of interrelationships among different project entities that drive coordination needs is a first step towards understanding which data needs to be collected to facilitate coordination. In a collaborative project, coordination needs arise because of interdependencies among people, artifacts, and tasks. In order to under-stand the relationships between these entities, we adapt Carley and Reminga's [1] concept of Meta Matrix – a methodology for storing and combining networks – for studying relationships among an organization's entities. The original Meta Matrix modeled the relationships among personnel, knowledge, resources, and tasks entities in an organization, which are then analyzed using network analysis tools to investigate the structural proper-ties of an organization for potential risks.

We adapted the Meta Matrix, specifically, for software development where interdependencies among different entities (e.g., artifact-artifact, developer-developer, or developer-artifact) can be represented as individual networks. These individual networks can then be combined in different ways via the Meta Matrix. Table 1 presents such a Meta-Matrix that consists of individual net-works, each representing relationships among the three main entities, namely person, artifacts, and tasks.

Currently, some data is already collected in a useful form that can be directly used to create networks in the Meta Matrix (e.g., which developer has checked in which artifact? which feature

has been implemented by which developer?); other kinds of data exist, but not in a form easy to use (e.g., who has worked closely with whom in the past? This question can be answered by identifying which developers had worked on the same Modification Request or on the same file at nearly the same time); while still other kinds of data are not typically collected at all (e.g., activity levels in a particular project or a document, views of documents).

Each cell in Table 1 represents a network and contains questions that are answered by that network. Researchers may use these questions as a guide to indentify which data is needed to create these networks, which can then support their collaboration needs

2.1 Individual Networks

Communication Network: Communication patterns that are answered by questions such as who talks to, works with, or reports to whom can reveal internal (often informal) organization structures such as which developers work closely together or which developers are sought out for advice. Studies have also shown that the lack of communication among developers is a leading cause for conflicts or defects in software [2]. It is therefore important to track and analyze communication patterns in a software project.

Data Collection: The fact that the bulk of communication in today's software project is via electronic media makes it relatively easy to archive communication history to answer "who talks to whom?" The real challenge is the ability to cross reference different communication channels (e.g., email, chat, discussion forums) when users employ different names in each channel or to query a particular topic which may be worded differently in different channels. Answering the question "who works with whom" is less straightforward and can be obtained from records of who had worked together on a common set of artifacts or same tasks, or based on discussions around a particular task or issue. Details about "who reports to whom" are not currently logged and it may prove beneficial to complement conventional org charts with data collected from surveys or interviews.

Expertise Network: Programmers, both experienced and novice, frequently need to ask "who has expertise in this [artifact, skill set, domain]?" Finding the correct answer is nontrivial and has been studied by researchers at length [3, 7]. However, these research tools are not in wide use and developers have to mostly rely on their colleagues' possibly incorrect memory.

Data Collection: Recommender systems analyze Change Management (CM) histories to identify which developer has edited which artifact and use that information, typically lines of code changed, to determine expertise. While extremely useful, this information could be complemented by logging information about: (1) the reason for the change, (2) the criticality of the change (e.g., bug severity, critical artifact in the design), and (3) skill sets or domain knowledge required per artifact (e.g., from design or requirement documents).

Artifact Network*:* Any non-trivial software project comprises of complex interdependencies among its constituent artifacts. Despite industry best practices of separation of concerns (using well defined Application Programming Interfaces) and modularity (separating specific functionality into specific modules) artifact dependencies can rarely be cleanly separated or completely contained. Moreover, the intangible nature of software makes it difficult to conceptualize these interdependencies and keep track of changes to them. Frequently,

information about changes to critical artifacts is not appropriately communicated across teams leading to conflicts and/or defects [5].

Data Collection: Interdependencies among artifacts in code can be identified using different methodologies. Code analysis techniques reveal programmatic dependencies among artifacts. However, in situations where the call statements are separated, associating artifacts that are frequently changed together can be a determinant of artifact dependencies. For example, in a remote procedure call (RPC) the called and calling functions may have many dependencies. However, they are separated by numerous links in a call graph because of the code implementing the RPC. Typically, code analyses focus only on the implementation phase, with data primarily extracted from CM repositories or issue trackers. It is equally important to consider dependencies

$$PP = PA \times AA \times PA^{'} \qquad \text{(Eq. 1)}$$

across life cycle phases. Researchers have started investigating such dependencies (e.g., Light House bridges the design and the implementation phase [4]; Chianti bridges the testing and implementation phase [9]).

Task Networks: As mentioned earlier, artifact dependencies create dependencies among developers and tasks. Conflicts can occur when multiple developers change the same artifact or when developers change interdependent artifacts [2, 5]. Here, we combine discussions about the three task networks mentioned in Table 1, namely *person-task* (which person has been allocated to which task), *artifact-task* (which artifacts belong to which tasks), and *task-task* (which task is precedent to which task). These networks provide a task centric view of the project that can serve as a quick coordination lookup for developers as well as managers. Such up-to-date task networks can also allow one to identify conflicting situations by determining which developer is working on which task, whether tasks might require concurrent changes to the same artifact or conflicting changes to interdependent artifacts.

$$TT = AT^{'} \times AA \times AT \qquad \text{(Eq. 2)}$$

Data Collection: Currently, there is a disconnect between project management and development tools. The above mentioned task networks are typically considered to be a project management chore rather than a coordination mechanism. As a result of which task networks often are not kept up-to-date with the latest development efforts. Such disconnects also occur because: (1) new artifact dependencies are created during development, (2) such changes occur frequently, and (3) no automatic logging facilities exist to identify such changes. Ideally, when developers commit files automatic light-weight logging mechanisms should enable them to link the artifacts with task details (which can be extracted from either issue tracker or project management tools). Clear Quest and Jazz, both commercial development environments, are working towards this direction, but they require developers to manually link each artifact changed with a particular task name. A more light weight and a tighter coupling between the committed artifacts and tasks is desirable.

2.2 Network Combinations

While the individual networks by themselves provide important information, the interesting feature of the Meta Matrix is that individual networks can be combined to provide deeper insights. Here we present three such examples:

Socio-Technical Dependency: It is well established that social dependencies are create because of artifact dependencies [2, 5]. One way of identifying such a socio-technical dependency is to combine the person-artifact and artifact-artifact networks. We can do so, by first specifying the relationship networks in matrix form. The person-artifact network can be represented as a person by artifact matrix (PA[m][n], m is number of developers and n the number of artifacts) where a one in cell *ij* indicates developer *i* is working with artifact *j*. Similarly, the artifact-artifact network can be represented as an artifact by artifact square matrix (AA[n][n], n is the number of artifacts), where a cell *ij* is one when artifacts *i* and *j* are interdependent.

Multiplying the two matrices (person-artifact and artifact-artifact) gives a person by artifact matrix that represents the set of artifacts a particular developer should be aware of, given the artifacts the particular developer is editing and the dependencies of those artifacts with other artifacts. Finally, a representation of socio-technical dependency, or the extent to which each pair of developer needs to coordinate their work is obtained by multiplying the product of person-artifact and artifact-artifact matrix with the transpose of person-artifact matrix (see Eq. 1). This product results in a person by person matrix where a cell *ij* (or cell *ji*) indicates the extent to which person *i* works on artifacts that share dependencies with the artifacts worked on by person *j*.

Task Dependency: We can employ similar matrix manipulation to create a set of task dependencies based on the underlying artifact dependencies. We can convert the artifact-task network into an artifact by task matrix (AT[m][n]), where a one in cell *ij* indicates that artifact *i* is associated with task *j* (such an association can be extracted from task centric CM systems like Clear Quest). By multiplying the transpose of this matrix with the artifact-artifact and the artifact-task matrices, we obtain a task by task matrix (see Eq. 2). Such task dependency relationships can be used, for example, to better gauge project completion schedules.

Knowledge Networks: Knowledge networks can represent different kinds of information. For example, knowledge can be regarding a particular software component (e.g., User Interface, database, networking, etc.) or regarding different technologies or programming languages (e.g., J2EE, Java, C++, SOA framework). Networks of *knowledge areas by development tasks* or of *knowledge areas by person* can be created by investigating which artifacts a developer has implemented or edited in the past, along with additional information about (1) the component nature of the artifact (e.g., UI component as derived from design documents) or (2) the skill set required to edit that artifact (e.g., a log of the technology required for a particular software).

One can also create a "system knowledge" network that shows which developers are knowledgeable about critical components of a given software system. Commercial software development often involves teams working on separate projects that are tied together with predefined APIs [5]. In many situations, the subprojects may even be in different domains and the teams geographically separated. In such cases, it is imperative for the teams to have a common understanding of their interdependencies which arise because of shared artifacts (e.g., APIs or interfaces) and possess some level of shared mental model of how each team operates. System knowledge network can be created following a similar approach as the social-technical dependency network, but in this case the artifacts to be considered must be shared artifacts across teams (e.g., interfaces, libraries, third party software).

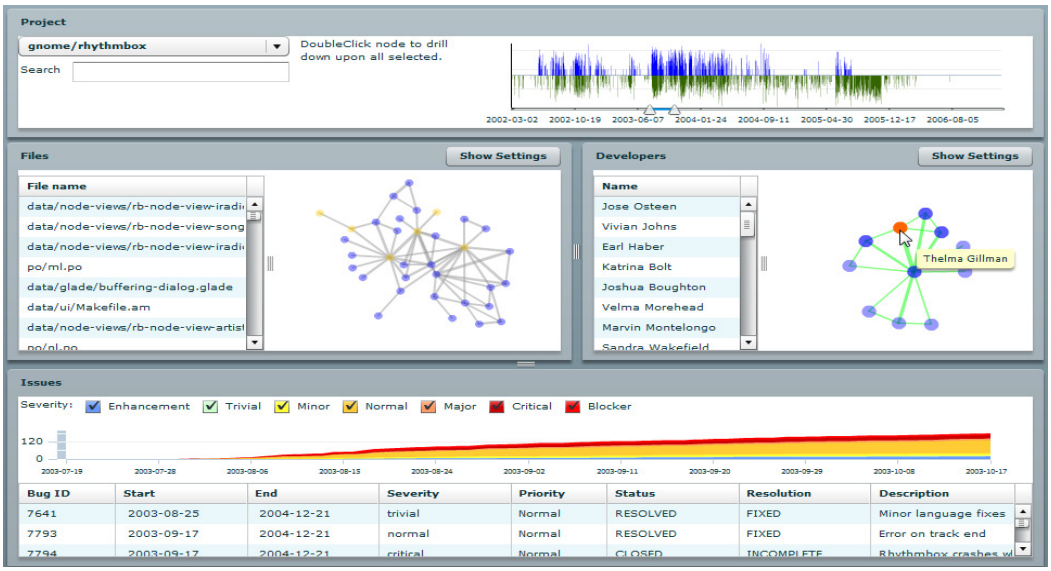

Figure 1. User Interface of Tesseract.

3 Tesseract: A Socio-Technical Browser

We present Tesseract, a socio-technical browser that analyzes code archives, communication records, and the project's issue database to capture a subset of socio-technical relations listed in the Meta Matrix. Tesseract uses four juxtaposed displays (see Fig. 1): (1) a time series display presents overall project activity – code commits on top and communications at the bottom – which can also be used to investigate a particular period in the project, (2) artifact networks as created by associating artifacts that are frequently changed together, (2) developer network based on their communication patterns, and (3) open issues or enhancements as a stacked area chart. Further, Tesseract uses Eq. 1 to create the coordination requirements matrix which is then matched with the communication patterns in the team [2]. In the developer network an edge between two developers is colored green when they are interdependent and communicate; otherwise the edge is colored red. These displays are cross-linked allowing interactive exploration of underlying relations tying these entities. For example, developers are linked to files that they have modified and bugs/issues on which they commented or fixed/contributed. Similarly, files that were edited to address a particular issue/bug are linked with that particular bug. We use juxtaposition and cross-linking of project entities to allow a quick exploration of the socio-technical relations in a large information space.

4 CONCLUSIONS

We have shown that interdependencies among entities (e.g., developers, artifacts, and tasks) that create coordination needs in software development can be represented as a set of networks. These individual networks can then be combined via the Meta Matrix to create derived networks. Articulating these relationships as networks allows informed investigation to provide deeper insights into the development process and team and for creating collaborative tools. We then present Tesseract, a socio-technical browser that analyzes data from various sources to capture individual networks of relationships between artifacts-artifacts, artifacts-developers, artifacts-bugs, developers-developers, developers-bugs, and a derived network – socio-technical dependencies.

Tesseract draws on the concept of Meta-Matrix to capture *communication*, *expertise*, *artifact*, and *task* networks by analyzing archived data from versioning systems, email archives, and issue trackers. Any production project involves such data – versions of artifacts, communication activities, and task lists, which implies that Tesseract can be easily extended for such domains outside the realm of software development. While the data that is already being archived is useful and can be employed to create interesting tools such as Tesseract, our intention is to start the discussion about the kinds of data that should be ideally collected and analyzed. We believe that the Meta Matrix can help us envision the different kinds of interrelations (among project entities) that are useful and how these interrelations can be combined so as to provide deeper insights that can guide us in creating better collaborative functionalities. This is particularly relevant now, because the advent of software being delivered as a service is drastically changing the way many collaborative tasks are performed and how data is captured and stored. We are at the moment in time and place where we can impact the data collection and archival by these providers.

5 REFERENCES

[1] Carley, K.M. and J. Reminga. 2004. ORA: Organization Risk Analyzer. Carnegie Mellon University, Institute for Software Research International.Tech Report: ADA460034 p. 50.

[2] Cataldo, M., et al. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. ACM Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada. p. 353-362.

[3] Cubranic, D., et al.,2005. Hipikat: A Project Memory for Software Development IEEE Transactions on Software Engineering, 31(6): p. 446-465.

[4] da Silva, I., et al. 2006. Lighthouse: Coordination through Emerging Design. OOPSLA workshop on Eclipse Technology eXchange. Portland, Oregon. p. 11-15.

[5] de Souza, C.R.B., et al. 2004. How a good software practice thwarts collaboration: the multiple roles of APIs in software development. International Symposium on Foundations of Software Engineering. Newport Beach, CA, USA. p. 22-230.

[6] Kersten, M. and G.C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. Fourteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. Portland, Oregon, USA. p. 1-11.

[7] Minto, S. and G.C. Murphy, *Recommending Emergent Teams*, in *Fourth International Workshop on Mining Software Repositories*. 2007. p. 5.

[8] Mockus, A. and J. Herbsleb. 2002. Expertise Browser: A Quantitative Approach to Identifying Expertise. International Conference on Software Engineering. Orlando, FL. p. 503-512.

[9] Ren, X., et al. 2004. Chianti: A Tool for Change Impact Analysis of Java Programs. Conference on Object-Oriented Programming, Systems, Languages, and Applications. Vancouver, BC, Canada. p. 432-448.