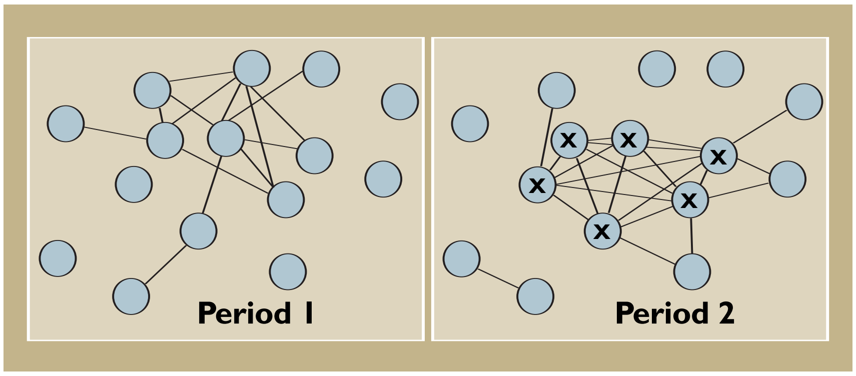
By PATRICK WAGSTROM *and* JAMES HERBSLEB

DEPENDENCY FORECASTING IN THE DISTRIBUTED AGILE ORGANIZATION

Coordination, or the management of dependencies among tasks, can be accomplished in a number of ways in software development projects. Coordination mechanisms include such things as a defined software process, a well-documented software architecture, and detailed project planning. Agile methods tend to eschew these formal coordination mechanisms in favor of frequent, intensive, informal communication among team members and with the customer.

Research has consistently shown, however, that communication across sites in geographically distributed projects is severely attenuated compared to communication in co-located projects. This strongly suggests that in order for agile methods to be effective in distributed projects, great care must be taken to ensure that the necessary communication takes place. Agile methods encourage very short planning horizons, and the dependencies that drive the need to communicate may shift frequently depending on the content of each delivery increment.

One way to identify dependencies that trigger the need to communicate is through extraction from the project code. Traditional dependency methods—identifying common data structures and calls—are labor intensive and may capture only a subset of the important dependencies. We have found that using an automated system that builds links strictly on whether or not developers have modified the same file produces excellent results at little cost to project staff. In this model, a time window, such as the last two weeks, is chosen based on the frequency of team change in the organization. Any two developers who have modified the same file in the time window are assumed to have a need to coordinate their work, and are given a link in the output. As a project progresses, the time window is shifted and new dependency graphs help visualize new dependencies and identify possible communication breakdowns.



The diagram here was automatically built from the version control system of a distributed commercially backed open source desktop application. The project is several years old and has contributions from hundreds of individuals. The two images correspond to two non-overlapping and consecutive two-week intervals of project development. During the first time period, the project focused on plug-in module development, the second time period was a coordinated effort to enhance the core of the program. Each node in the graph represents a developer and lines connect a pair of developers if they worked on the same file. To ease analysis, the position of nodes is kept constant when comparing time periods.

In this organization, as the focus changed from plug-ins to core, the anticipated communication dependencies also increased—as emphasized by the six developer clique, marked with x's, in the second time period. Because this organization is distributed, extra efforts are required to ensure that dependencies are known and communicated. These could include new email policies and teleconferences for the distributed members. In this case, core development team meetings should incorporate all developers in the clique regardless of whether or not they were assigned to the team.

While for this project we computed these networks based on changes made to files during the two-week period, it is also possible to generate networks that predict interaction and coordination needs. For example, one can use prior change history to construct a network where files that were changed in the same modification request are linked together. This is a good proxy of the dependencies between these two files. When combined with a network view of who modified each file, a tool can be used to construct a graph representing who needs to coordinate with whom during the current iteration as soon as developers check out files for editing.

Given the degree to which agile methods rely on frequent communication in order to coordinate project work and the degree to which the coordination requirements change over time, it can be daunting to ensure sufficient communication and coordination across all channels. Taking a network-level view allows automated generation and visualization of these complex dependencies—reducing management overhead while increasing efficiency. We believe that as distributed agile development increases these tools will be essential for understanding communication needs, establishing project teams, and ensuring overall project success. **C**

Patrick Wagstrom (pwagstro@andrew.cmu.edu) is a Ph.D. candidate in the Computation, Organizations, and Society Program and the Department of Engineering and Public Policy at Carnegie Mellon University, Pittsburgh, PA.
James Herbsleb (jdh@cs.cmu.edu) is the Nico A. Haberman Associate Professor of Computer Science and the director of the Software Industry Center at Carnegie Mellon University, Pittsburgh, PA.