On the Perceived Interdependence and Information Sharing Inhibitions of Enterprise Software Engineers

Alicia M. Grubb
Department of Computer Science
University of Toronto
Toronto, ON, Canada
amgrubb@cs.toronto.edu

Andrew Begel
Microsoft Research
Redmond, WA, USA
andrew.begel@microsoft.com

ABSTRACT
Software teams often have trouble coordinating shared work due to poor communication practices. We surveyed software engineers (N=989) at Microsoft to investigate three rarely explored aspects of coordination: (1) how an engineer's perception of dependence is predicted by his organizational characteristics, (2) how this perception differs when the dependence varies by the kinds of shared work artifacts, and (3) how the work group range affects the likelihood that an engineer will share information about work artifacts with another. Our results indicate that engineers tailor their communications about shared work for each group of intended recipients. This suggests that many existing coordination tools that rely on automatic mining and visualization of engineering activities have prevented senders from controlling the distribution of information about their work, and may have overestimated the receivers' abilities to comprehend it.

Author Keywords
Coordination; Enterprise software development; Awareness; Communication

ACM Classification Keywords
K.4.3 Organizational Impacts: Computer-supported cooperative work

General Terms
Human Factors

INTRODUCTION
Large-scale software engineering requires coordination between engineers who build and ship a software product. Effective coordination relies on communication between individuals who are dependent on the status and content of the shared work artifacts (e.g., source code, bugs, tests, documentation, and project schedules). Prior research has shown that necessary communication is inhibited by physical, technical, cognitive, social, and organizational factors, which leads to coordination breakdowns that can threaten the viability of a project [14, 18, 34].

Quantifying the mismatch between actual communication and suggested communication (by relating people to their shared work [12]) can help indicate possible coordination problems (e.g., build failures [35]) which may be fixed by increased (or decreased) communication [17, 26, 28, 41, 49]. If employed successfully, better *communication strategies* should improve team performance [1].

We tried to employ this extensive body of research to make sense of observations of inter-team coordination in large-scale software development at Microsoft, and noticed three gaps. First, studies suggest that if engineers communicated properly with one another along dependency lines, it would avoid problems [12, 35]. But, we find that this picture to be too simplistic; not all engineers are the same. The activities associated with some job functions involve many more dependencies than others. Likewise, an engineer's departmental affiliation and tenure in the company influences the number and depth of dependencies [9, 25]. Differences in dependencies may have significant effects on associated coordination and communication needs.

Second, software modularity at all levels (e.g., methods, classes, libraries, components, applications, etc.) isolates teams of engineers from one another [19], and creates an *asymmetry* in many engineers' perceptions of their dependencies. To an engineer using a software library, it is obvious that he depends on the library's team. However, the library team may be completely unaware of his existence unless he chooses to contact them. One will not (and cannot) communicate with a dependent that one does not perceive to exist.

Third, social biases and organizational culture can interfere with an engineer's willingness to share information about his work with outsiders, even direct dependents. The amount of information an engineer is willing to share is inversely related to his *work group range* to the other person. People on the same team trust one another much more, and consequently share much more information, than people who work in different corporate departments. Ingroup bias and confidentiality requirements may be the source of these inhibitions. A belief that outsiders would not even be able to comprehend your information (i.e. the illusion of transparency) may also play a role.

CSCW12, February 11–15, 2012, Seattle, Washington, USA.

In our study, we investigated three research questions:

> RQ_1: Does an engineer's organizational characteristics (measured by job function (developer, tester, project manager (PM)), management role (manager or individual contributor (IC)), department, years of experience in the software industry, and years of experience in the company) predict whether an engineer perceives that he depends on or is dependent upon shared artifacts from engineers on other teams?
>
> RQ_2: How does an engineer's perception of his dependencies differ with the kinds of shared artifacts, and how much?
>
> RQ_3: How does *work group range* (measured on a scale from no one, to my team, and to the entire company) affect the likelihood that an engineer will share information about work artifacts with another? Does this likelihood vary by artifact type?

We surveyed software engineers (N=989) at Microsoft, a large U.S.-based software company, on their perceptions of interdependence and communication between engineers. Our findings on interdependence mainly agree with previous research at Microsoft [7, 36], but reveal several significant differences. As we thought, our data indicates that engineers' perception of their dependencies varies significantly by their organizational characteristics and the direction of the dependency. In addition, among many work artifact types, we see significant signs of an asymmetry in dependency perception.

We found that the most widely shared work artifacts are specifications and release schedules. Among developers, code changes (checkins) are surprisingly the most *narrowly* shared work artifacts. Our analysis revealed that with every type of work artifact we asked about, engineers who perceive that they depend on others are both permitted, and willing, to share information more widely than those who do not. We expected to find the reverse—if you reported that others depend on you, you would be more likely to share—but did not. Finally, irrespective of dependencies, developers and requirements and project engineers felt more comfortable sharing information than they were permitted, but testers felt less comfortable.

Our findings and analyses suggest engineers customize the information they share according to the organizational characteristics and work group range of their chosen recipients. In the Implications section, we explore how the automation inherent to many existing coordination support tools may in fact thwart this customization ability, and places undue burden on receivers to comprehend the import of a generic, untargeted message. Many of these tools do not allow senders to control the recipients of their messages at all, which further inhibits communication. We believe that a new generation of communication and coordination support tools operating with a mixed-initiative approach to message customization and recipient targeting could enable senders to increase their information sharing with a comfortable sense of control.

RELATED WORK

There is inherent risk in depending on others to develop software; they may not deliver on their promises. Managing the risk and ensuring a successful outcome, then, requires discovering, maintaining awareness of, and communicating effectively with one's dependencies.

Dependence Discovery

Since Conway, we have known that software teams organize the architecture of their software similarly to their own organization [13]. Thus, software analysis methods are often used to derive technical dependency networks and relate them to their authors [16]. However, more complete socio-technical dependency network discovery derives additionally from an understanding of software process, organizational culture, and workflow [11]. Here, empirical methods that mine software repositories are used to uncover how team members coordinate their work around various kinds of work artifacts, such as bugs [12], checkins [38], and emails [4].

While the dependence network shows where communication should occur, it does not show where it *does* occur. Cataldo quantified this difference using a measure called socio-technical congruence [12], and along with others, showed that the magnitude of the congruence difference correlates with coordination problems and project failures [11, 35, 51].

The existence of a so-called *socio-technical congruence gap* illustrates that many engineers have difficulty identifying (and communicating with) their dependencies. Our work seeks to discover how organizational factors can help predict whether engineers perceive that they have dependencies on one another.

Dependence Awareness

Engineers gain awareness of their dependencies's actions during daily activities. At a high level, daily work varies by job function, management role, departmental affiliation, and tenure within the company. Developers communicate with colleagues and dependents about software code quality through code reviews [43], about software code changes through push-based notification mechanisms [32, 22], and learn about code ownership and responsibility through the exploration of revision control systems and bug databases [6, 23]. Testers wait for project managers (PMs) to send them specifications for their features before they can write testing plans [5, 36]. Then, they must wait for developers to finish writing the code and notify them before they can run their tests and report bugs. PMs depend on one another to coordinate work between dependent teams by maintaining awareness of the status, scheduling, and any changes to work artifacts on which their own team depends. They conduct regular meetings with other PMs and developers to prioritize bugs and to coordinate component completion schedules [2].

Despite much communication, engineers' awareness of their dependencies remains incomplete. For example, both developers and PMs may stay unaware of testers' work until the tester finds and reports a bug. Testers reporting bugs in a build may have difficulty discovering which developers among a

large set should be made aware of a problem, and resort to inefficient email broadcasts to notify them [3]. Requirements gathering activities challenge many PMs, due to the difficulties in discovering and communicating with *all* relevant stakeholders [37] and the dynamic and unpredictable dependencies between projects with changing requirements [15]. Creating and enforcing strict APIs between component modules hinders developers from observing and communicating about shared work that spans the boundary [19]. If developers are not contacted by users of the modules they own, they may lack awareness of the existence or needs of their dependent developers.

Zooming in, some software team practices such as daily standup meetings (common in Agile software methods), can improve awareness of imminent dependencies (e.g. when one engineer is blocking another from getting work done). Coordination tools, such as configuration management systems, dependency and conflict awareness visualizations, and collaborative IDEs, support communication, work artifact management, and task management [46]. Recommender systems can help engineers discover others who are working on similar projects to them, so that they might discover and stay aware of one another's progress [40, 53]. These tools are often intended for individual developers, leaving other job functions, management levels, and experience levels understudied and under-supported.

Our study identifies how engineers with various organizational characteristics perceive their dependencies on work artifacts beyond software code and APIs. The prevalent asymmetry of this dependence, along with an engineer's organizational characteristics, affects his visibility into and understanding of those dependencies [44, 42]. Our survey uncovers these work artifact dependence asymmetries and explores how these might affect the perception of dependence.

Communication Inhibitions

Even when engineers are aware of their dependents, they may not wish to share information about their work with them. Inhibitors to communication include geographic separation [30, 31, 39], information overload [32], low social capital [33] and lack of trust [5, 10, 31, 52], organizational separation [13, 19], and personality [48]. Communication difficulties in software teams resemble those in many other kinds of product development organizations [9], therefore advice on improving coordination through increased cross-functional communication may be applicable to software organizations as well [1, 21].

Prior work rarely discusses communication inhibitors in *enterprise* software development. The perceived confidentiality of one's work artifacts will inhibit the distribution of information about the artifact to recipients. The specific sensitivity of the work artifact will inversely affect the acceptable communication range, i.e. a more sensitive work artifact will be shared with fewer people than a less sensitive or public one. A prevalent desire for a team to speak with one voice and control its message to outsiders may inhibit sharing information that is unfinished, or not yet approved by management. Likewise, individual team members may not feel empowered to speak on behalf of the team, preferring that an official team liaison mediate communication with outsiders [36]. As teams isolate themselves from others, they may easily demonstrate social biases where they mistrust others — something which is equally likely in open source software teams [47] — or perceive that their dependents do not know enough to understand the information that they requested. Our survey uncovers the diversity and range of these enterprise-oriented communication-inhibiting factors on engineers with varying organizational characteristics.

RESEARCH CONTEXT

At Microsoft, engineers fall mainly into one of three job functions: developer, tester, and project manager (PM) [36]. Developers are responsible for writing the software code and fixing bugs. Testers create test harnesses, write and execute blackbox and end-to-end scenario-based test suites, and report bugs. PMs elicit requirements from customers to create high-level design specifications and coordinate work between the developers, testers, and PMs of other teams according to the product's release schedule.

The smallest workgroup at Microsoft is the *feature crew*, in which a developer, a tester, and a PM create, develop, and maintain a feature in the product. *Feature crews* may be organized hierarchically or grouped according to the coupling of their features, but ultimately report to a *team* consisting of development, test, and PM leads and managers. The entire set of *teams*, along with general managers, architects, and liaisons to sales and marketing form a *product group*, ranging in size from 100 to 5,000 people. *Product groups* are then organized into *departments*, each of which housing a suite of products.

Engineers at Microsoft coordinate their work with one another using a variety of communication methods, including face-to-face and computer-mediated team meetings, emails, instant messages, phone calls, and via entries in work item (bug) databases [7].

METHODOLOGY

Our research was conducted using a web-based survey over a period of 8 days in the summer of 2010. An invitation was emailed to 3,000 software engineers (with at least one year of tenure at Microsoft), randomly selected from a much larger population of around 28,000 engineering employees from around the world. Responses were anonymous, but respondents could identify themselves (separate from their survey submissions) to enter a raffle for a prize of US$250. The survey contained 40 questions, though in this paper, we only analyze questions relating to interdependence and information sharing.[1]

In preparing the survey, we prototyped our questions through a series of pilot surveys and followup interviews (in which we discussed the wording of the questions). We adapted lists of work artifacts from the previous work at Microsoft [7]. All questions with lists allowed the respondents to select

[1]We encourage other researchers to replicate our study. Please contact us for a copy of the questionnaire.

Table 1. Response Rate and Sample Distribution by Job Function[2]

	Total	PMs	Devs	Testers	Other
Total Sampled	3000	1000	1000	1000	0
Responses	989	247	327	376	39
Response Rate	33%	24.7%	32.7%	37.6%	–
Percent of Sample	–	25%	33%	38%	4%

"Other," and include a qualitative response. We analyzed these "Other" responses for completeness, but they were not frequently used by the respondents.

Demographics

The survey began with a set of demographic questions to determine the respondent's organizational characteristics. Respondents were asked to indicate their job function (PM, Developer, Tester, Other), their management role (Individual Contributor (IC), Lead, Manager, Other), department within the company, the number of years of professional software engineering experience they had, and the number of years they had worked at Microsoft.

We received 1052 responses, of which 63 were invalid (empty or duplicate). This resulted in 989 valid responses, for an overall response rate of 33% (see Table for response rates by job function). Overall, respondents reported that they worked at Microsoft for an average of 6.01 years (SD = 4.3), and as professionals in the software industry for 10.03 years (SD = 6.59). Although both means are positively skewed, the individuals in our sample report a fair amount of experience, and have spent several years working in the software industry before joining Microsoft. Years of experience at Microsoft is strongly correlated with years experience in the software industry ($r = 0.52$, $p < 0.001$). For the remainder of our analysis, we will only use years of experience at Microsoft.

PMs were the most experienced, followed by developers and then testers, both at Microsoft and as software engineers. From our sample of 989 participants, 774 respondents reported working as individual contributors, 158 were either leads or managers, 39 reported working as both an individual contributor and a lead/manager, and 18 reported working in another role (e.g. architect or executive). Managers have four years, on average, more experience at Microsoft than individual contributors, but a similar number of years as software engineers. The distribution of respondents' departments within our sample is consistent with the distribution of software engineers across the organization.

Statistical Methodology

Throughout the Results section, we examine various relationships within the dataset. The dataset is quite large and the majority of the variables are ordered or dichotomous variables. We used *t* tests to evaluate whether two sub-populations of our sample had significantly different means, and whether the slope of our regression lines differed significantly from zero. We used ANOVA (Analysis of Variance) to test for significantly different means in more than two subgroups. We minimally used CHI-Square (χ^2) because its sensitivity is problematic with large samples. Throughout the analysis, we used three forms of regression: linear regression for continuous dependent variables, binary logistic regression for dichotomous dependent variables, and ordered logistic regression for categorical dependent variables.

[2] *Other* responses were excluded from analysis when split across job function.

Terminology

We introduce two terms: *iDepend* – the respondent perceives that he depends on work done by others outside his team, and *oDepend* – the respondent perceives that individuals from outside his team depend on him for his work. We also list the 20 work artifact types for which at least one engineer in Microsoft reported a dependency: API Definitions, Binaries, Bug Fixes, Bug Reports, Build Information, Checkin Messages and Source Code Diffs, Source Code, Delivered features, Documentation, Schedules (Feature, Release, and Product/Shipping), Specifications (Design, Development, Implementation, Product, Test, and Other), Team-specific Engineering Standards, Team Meeting Notes, Tests, Test Results, Work Items, Work Item Priorities, and Work Status.

In our understanding, the concepts of *work artifact* and *information about a work artifact* are indistinguishable. When an engineer sends a message to another, sometimes it is a simple artifact (e.g. source code in a repository, binaries in a build, specifications on a web site), and sometimes it is a deliberate act of communication (e.g. an email sharing status meeting notes, an IM checking up on a task request). But, it is often both at the same time. For example, writing a checkin message might be considered sharing information about code, but the source code diff contained in the checkin is code. The reduction of a buggy program to a simple test case to include in a bug report is simultaneously an act of communication and an artifact. Bug reports contain structured fields describing the problem, while unstructured fields record acts of communication between the engineers who worked on it. Thus, in this paper, we use the terms *sharing artifacts* and *sharing information about artifacts* interchangeably.

We operationalize the notion that individuals share their work information (the status and content of the shared work artifacts) to specific audiences within Microsoft. We measure sharing through a set of proxies corresponding to various work information types (including artifacts and status) using a scale called *work group range*. This scale ranges from (0) no one, (1) their feature crew, (2) their team, (3) their dependents and those that depend on them, (4) their whole product team, (5) their department, or (6) the entire company. We combine these proxies to create a *sharing metric* instrument to understand with what work groups engineers share confidential and non-confidential information. We also use the same *sharing metric* to evaluate the difference between the information that engineers are permitted to share vs. what they report feeling comfortable sharing.

Table 2. Logistical Regression Analysis of Perception of Inter-team Dependencies

(Q1) I depend on the work of people outside my team.					
iDepend	Coeff	OddRatio	Std Coeff	z	$P > \|z\|$
Developer	-1.09	0.34	0.60	-3.83	0.000
Tester	-1.30	0.27	0.53	-4.70	0.000
Manager	0.45	1.57	1.20	1.73	0.083
Years-Exp.	0.08	1.08	1.40	3.09	0.002
Intercept	1.88			6.52	0.000
Number of Observations=899				Log likelihood=-420.1809	
				Null Model Log likelihood=-456.0344	

(Q2) People outside my team depend on work that I personally do.					
oDepend	Coeff	OddRatio	Std Coeff	z	$P > \|z\|$
Developer	-1.01	0.37	0.62	-3.52	0.000
Tester	-1.32	0.27	0.53	-4.75	0.000
Manager	0.49	1.64	1.22	1.85	0.064
Years-Exp.	0.08	1.09	1.40	3.07	0.002
Intercept	1.87			6.47	0.000
Number of Observations=899				Log likelihood=-415.4851	
				Null Model Log likelihood=-448.9960	

RESULTS

Interdependence

We asked respondents about their perception of interdependence with two Yes/No questions: (Q1) "I personally depend on the work of people outside my team." (Q2) "People outside my team depend on the work that I personally do." 78.8% of respondents said they were dependent on others outside their team, and 79.7% said that others depended on their work. Only 67.4% of respondents said yes to both questions (bi-directional dependencies). Q1 and Q2 are moderately correlated ($r(911) = 0.37, p < 0.0001$).

We were interested in learning whether our respondents' organizational characteristics could predict whether they perceived having a dependency. The null hypothesis is that the likelihood of having inter-team dependencies does not vary by organizational characteristics. We explored this relationship using logistic regression. Q1 and Q2 were coded as dichotomous dependent variables. Any respondent who indicated that the question was not applicable ("N/A") was coded as blank.

The organizational characteristics likely to correlate with perceived interdependence are job function (PMs, Developers, Testers, Other),[3] management role (Yes (Lead or Manager) or No (IC)), department, and years of experience (minimum 1 year). We removed department from the analysis because there was no significant variation.

In Table 2, we report significant variables and their significance levels for the regressions of Q1 and Q2. The log odds of years of experience and job function were significant in both models; this allows us to reject our null hypothesis. PMs have a significantly higher likelihood than Developers ($\chi^2(1) = 14.69, p = 0.0001$) and Testers ($\chi^2(1) = 22.08, p < 0.0001$) of reporting that they are dependent on engineers outside their team. Perceiving an external dependence also varied by years of experience ($\chi^2(1) = 9.53, p = 0.002$); greater years of experience increase a respondent's likelihood of depending on others. PMs have a significantly higher likelihood than Developers ($\chi^2(1) = 12.41, p = 0.0004$) and Testers ($\chi^2(1) = 22.53, p < 0.0001$) of perceiving that others depend on their work. Engineers with more experience have an increased likelihood as well ($\chi^2(1) = 9.44, p = 0.002$). We did not find any significant interaction effect between years of experience and job function.

[3] *PM* was selected as the omitted category. *Other* was removed due to extremely low sample size.

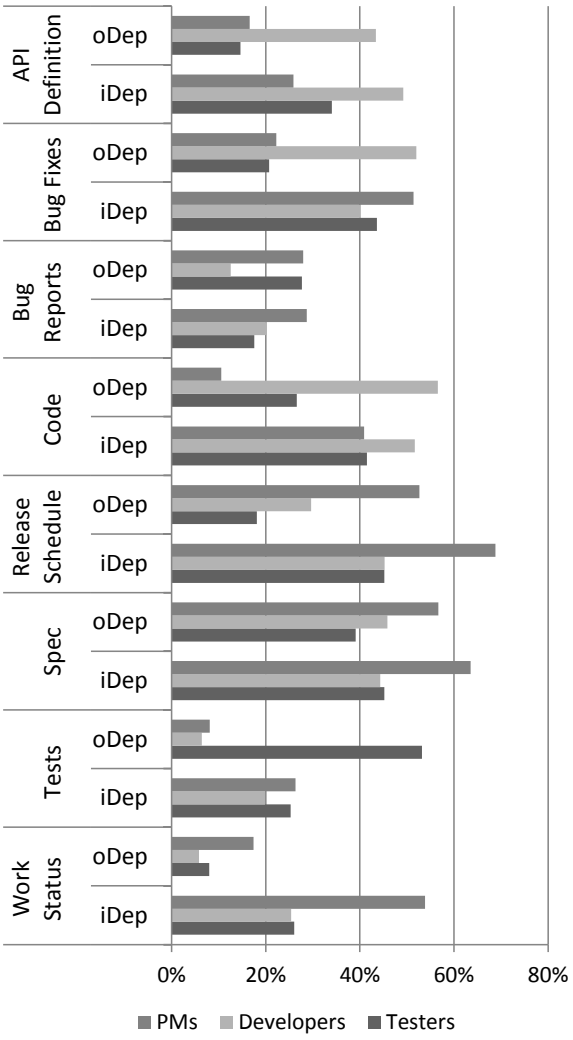

Figure 1. Artifact dependency as a percentage of respondents with dependencies divided by job function.[5]

Shared Work Artifacts

Next, we investigated how an engineer's perception of his external dependences on different types of shared work artifacts varied by his organizational characteristics. We asked respondents (Q3) "Which kind(s) of work artifacts from engineers outside your team do you depend on?" and (Q4) "For which

[5] Some work artifact types listed in the question (selected by very few respondents) were omitted to make the plot easier to read.

kind(s) of work artifacts do engineers outside your team depend on you?" Respondents could choose any number of artifacts from a list we supplied (see Terminology section), and/or choose to write in their own artifact type. Our null hypothesis is that the distribution of perception of dependence between different shared work artifact types does not vary by a respondent's organizational characteristics.

We first explored which shared work artifacts were reported as the most common dependencies. For each work artifact type, we calculated the percentage of respondents who reported being dependent upon an engineer outside their team, and the percentage of those who reported that an engineer outside their team was dependent on them. Respondents reported that they depended most on others for release schedules and specifications, while others were most dependent on them for specifications and source code.

Figure 1 shows these shared work artifacts split by job function. Each row consists of two sets of three bars representing the percentage of the respondents who report being dependent on that artifact (iDepend) and the percentage of respondents who report that others depend on them for the artifact (oDepend). Each of the three bars represents the results for respondents with one of the job functions. When divided by job function, we observe striking variations in most of the dependencies. These variations are significant, and enable us to reject our null hypothesis.

In this figure, one can additionally see distinct asymmetries in the direction of each work artifact type's dependencies. 53.85% of PMs report that they depend on other engineer's work status, but only 17.41% report that others depend on their work status. These differences are larger for developers (iDepend: 25.38%, oDepend: 5.81%) and testers (iDepend: 26.06%, oDepend: 7.98%). For almost every work artifact type and job function, oDepend is less than iDepend. The exceptions for developers are bug fixes, and code, which makes sense, since writing code and fixing bugs are their main job tasks. Testers report that more people are dependent on them for bug reports and tests, matching their job tasks.

Information Sharing

In this section, we look at the factors that influence how widely engineers share information about their work artifacts. In an enterprise work environment, there are rules about sharing information with other teams. Some features are considered *sensitive* (i.e. secret), and should not be talked about with people who are not privy to the secret. Most features are *non-sensitive*, which Microsoft often advertises to the public long before a product ships.

We asked respondents four questions that covered whether they were permitted to share and comfortable with sharing information about *sensitive* and *non-sensitive* work artifacts. One instance of the question read, "What are the work groups to which you are *allowed* to share each of the following work artifacts that are related to this *sensitive* feature?"[6] Three variants substituted *comfortable with* for *allowed to* and *non-sensitive* for *sensitive.* We label these (Q5) allowed-sensitive (AS), (Q6) comfortable-sensitive (CS), (Q7) allowed-non-sensitive (AN), and (Q8) comfortable-non-sensitive (CN). Each of the representative work artifacts were rated by respondents on our *work group range* scale and combined into a single *sharing metric* value for each of the four questions (see Terminology section). We accommodated missing data in the *sharing metric* to ensure that missing responses would not positively skew our results. The resulting distributions approximate to being continuous and normal.

[6]Italics were in the original question.

Table 3. Mean & Standard Deviation of the work group range (0-6) to which engineers were allowed to share and comfortable with sharing information about sensitive and non-sensitive features. Higher values indicate wider sharing. Each of the 8 sections reports the statistics for a different subsample. The four questions are labeled (Q5) AS: allowed-sensitive, (Q6) CS: comfortable-sensitive, (Q7) AN: allowed-non-sensitive, and (Q8) CN: comfortable-non-sensitive.

	Mean	Std. Dev.		Mean	Std. Dev.
Entire Sample			**iDepend = Yes**		
(Q5) AS	3.017	1.279	(Q5) AS	3.091	1.264
(Q6) CS	3.089	1.283	(Q6) CS	3.155	1.252
(Q7) AN	3.865	1.442	(Q7) AN	3.942	1.404
(Q8) CN	3.850	1.472	(Q8) CN	3.924	1.431
PMs			**iDepend = No**		
(Q5) AS	3.098	1.337	(Q5) AS	2.767	1.310
(Q6) CS	3.190	1.308	(Q6) CS	2.893	1.396
(Q7) AN	3.969	1.401	(Q7) AN	3.653	1.543
(Q8) CN	3.947	1.453	(Q8) CN	3.642	1.583
Developers			**oDepend = Yes**		
(Q5) AS	3.088	1.315	(Q5) AS	3.059	1.289
(Q6) CS	3.278	1.325	(Q6) CS	3.150	1.272
(Q7) AN	4.054	1.414	(Q7) AN	3.935	1.424
(Q8) CN	4.059	1.406	(Q8) CN	3.914	1.459
Testers			**oDepend = No**		
(Q5) AS	2.916	1.202	(Q5) AS	2.894	1.233
(Q6) CS	2.877	1.202	(Q6) CS	2.916	1.322
(Q7) AN	3.645	1.459	(Q7) AN	3.669	1.473
(Q8) CN	3.613	1.509	(Q8) CN	3.670	1.487

The mean and standard deviations of the *sharing metric* and subpopulations are reported in the 8 sections of Table 3. Notice that respondents indicated that they were allowed ($t(750) = 17.94$, $p < 0.0001$) and comfortable ($t(739) = 16.73, p < 0.0001$) sharing non-sensitive features much more widely than sensitive ones. Additionally, there was a significant effect for perceiving that one had external dependents in all cases (AS: $t(802) = 2.87$, $p < 0.01$, CS: $t(770) = 2.25$, $p < 0.05$, AN: $t(765) = 2.23, p < 0.05$, CN: $t(769) = 2.14$, $p < 0.05$). Those who say they depend on the work of people outside their team (iDepend) share to a wider audience than those who say they do not have outside dependents. However, when the respondent perceives that others depend on his work, there is a significant effect only in the comfort-sensitive and allowed-non-sensitive cases (CS: $t(770) = 2.01$, $p < 0.05$, AN: $t(765) = 2.02$, $p < 0.05$). Overall, respondents who perceive that others depend on their work (oDepend) share more widely than those who do not.

Splitting the data by job function shows that PMs and developers reported being allowed to share work artifacts with a wider audience than testers. There was a significant effect

Table 4. Artifacts (and their means) that are shared the most widely and most narrowly by role. Increased values indicate wider sharing.

	Most Widely Shared		Most Narrowly Shared	
PM	Release Schedule	4.26	Source Code	2.77
	Specifications	4.24	Checkin Msg.	2.81
	Feature Schedule	4.18	Test Suites	2.95
Dev	Release Schedule	4.24	Meeting Notes	2.71
	Specifications	4.24	Test Suites	2.98
	Binaries	4.18	Source Code	3.00
Test	Release Schedule	3.79	Meeting Notes	2.53
	Bugs Fixes	3.77	Source Code	2.69
	Bug Reports	3.69	Checkin Msg.	2.71

for job function in all but the allowed-sensitive case: (CS: $F(2, 790) = 8.07$, $p = 0.000$, AN: $F(2, 785) = 6.65$, $p = 0.002$, CN: $F(2, 788) = 7.43$, $p = 0.001$).

To further understand the relationship between organizational characteristics, the perception of interdependence, and our *sharing metric*, we performed linear regressions for each of the four questions (AS, CS, AN, and CN). AS is not significant. In both the CS and AN models, there is a statistically significant difference between PMs and testers, and between developers and testers. In the CN model, there is a statistically significant difference between developers and testers. For the CS, AN, and CN models, a management role and years of experience are also statistically significant, albeit with smaller standardized effect sizes than job function. Years of experience has the largest standardized effect size. The descriptive statistics indicate that comfort levels contribute to the significant effect between PMs and testers, and between developers and testers. Thus, developers and PMs are more comfortable sharing both sensitive and non-sensitive artifacts than they are permitted. Testers, however, are allowed to share more widely than they feel comfortable with. There may be an interaction effect between job function and the perception of interdependence that requires further data to validate.

Next, we analyzed which artifacts were shared to the widest and most narrow work group range. The results are shown in Table 4. Respondents of any job role were willing to share release schedules with the widest range of recipients. PMs shared code and checkin information with the most narrow audience. Developers shared team meeting notes and test suites with the most narrow audience, and testers minimally shared team meeting notes and code.

After asking about these hypothetical information sharing situations in our survey, we asked, using Yes/No questions, whether respondents had ever not been allowed to or felt uncomfortable sharing information about a sensitive or non-sensitive feature in response to an explicit information request. If they had, we asked them to explain why. 33.2% of respondents report having been asked for work-related information that they were not *allowed* to share. Fewer (25.1%) reported being asked for information that they did not feel *comfortable* sharing. We triangulated their answers by analyzing the long-form "why" follow-up question. Using an iterative card-sorting technique, we categorized the respondents' reasons into 5 groups: (A) the sensitivity of the request (e.g. confidentiality, legal issues, preventing leaks to the press), (B) uncertainty as to whether the information should be shared (e.g. response should come from management), (C) the information was too preliminary (e.g. draft or incomplete work), (D) mistrust of the requestor (e.g. request is too vague, requestor has no "business need," requestor may have "malicious intent."), and (E) would not understand (i.e. the requestor lacks the background or technical expertise/depth to understand the information).

DISCUSSION

The results and analysis of the survey support several conclusions. First, answering RQ_1, PMs are more likely than developers or testers to perceive that they have dependency relationships with outside teams. This is an expected result, given that PMs function as project engineers and are responsible for coordinating their team's work with others. Second, engineers with more work experience are more likely to perceive that they have dependency relationships with outside teams. This is consistent with an earlier finding [36] that junior engineers are insulated from outside teams by a team lead who handles much of the inter-team communication. We did not find significant correlations of dependence perception with a management role or departmental affiliation.

The third result, answering RQ_2, showed that overall, engineers of all job functions report that they depend the most on specifications and release schedules. This result initially surprised us, but when the dependencies were split by job function, more familiar rankings appear for developers — they feel most dependent on code and API definitions. The split does not change the ranking for testers and PMs. This makes sense for PMs, who coordinate work between teams, however for testers, it confused us. After we interviewed a few testers, however, we found that they must wait for specifications from PMs before they can begin designing their test suites, and are extremely sensitive to the release schedule because much of their work cannot proceed until the developers have finished programming new features. Indeed, the fourth most dependent work artifact for testers is source code. Continuing with the opposite dependence direction, engineers of all job functions feel that others depend on them mostly for specifications and code. When broken down by job function, PMs provide specifications and schedules, developers provide code and bug fixes, and testers provide tests and specifications. This result also agrees with the literature.

We noticed that some of the dependency numbers were quite low. For example, only 6.42% of developers report that others depend on them for tests. Since at Microsoft, testers write the integration and acceptance tests, while developers write unit tests, this indicates that either very few developers perceive that anyone depends on them to write unit tests, or that very few engineers depend on the results. A followup interview revealed that developers are expected to be responsible for their own unit tests, and rarely expect others to read, write or run them.

Another set of low numbers for developers was for bug reports. Only 20.18% say they depend on others' bug reports,

and only 12.54% say that others depend on theirs. Since developers are the only engineers knowledgeable enough to file bugs against the source code, this may indicate they do not find bug reports to be useful for telling one another about buggy code. Developers told us in followup interviews that when they may suspect a bug in another team's code, they feel that they are likely missing something important. It is better to leave "buggy" code alone, than risk causing a new problem by changing it. They would also rather visit another developer in person to tell them about a bug than file a bug report.

Our last results answer RQ_3, and focus on the willingness to share information about shared work artifacts with a variety of audiences. When the feature to be shared was sensitive (for which respondents were told that they should be concerned about confidentiality), engineers mainly refrained from sharing information further than their immediate dependencies. When the feature was not sensitive, they could share with their entire product team.

A surprising result is that engineers who perceive that they depend on others feel comfortable sharing information about work artifacts with a *larger* range of recipients than those who say they do not depend on anyone. Might this suggest that engineers at this company are altruistic? Perhaps they share with others to set an example for, and thus encourage, the people they depend on to share more information. We expected the inverse, that engineers would only share more widely when they knew that people on other teams were depending on them for information. This will require further exploration at Microsoft (and at other companies) to understand the engineers' sharing motivations.

Considering job function, we found that developers and PMs are more comfortable sharing information about their work artifacts than they are permitted to by their organization. Testers, however, reported the opposite; they are permitted to share with a wider audience than with whom they feel comfortable. Two possible explanations came to mind. (1) If testers are reporting news of problems with the product, sharing that news widely might harm its reputation. (2) Testers may lack enough confidence in their own abilities and activities to reveal their ideas about work artifacts, and fear they might receive a poor response from their more accomplished and experienced PM and developer co-workers. Followup interviews revealed that testers' lack of confidence was the most likely explanation.

Finally, more engineers than we expected reported being apprehensive or not allowed to share information when asked. One-third of all respondents reported they were not permitted to share information after being explicitly asked, and one quarter felt uncomfortable doing so. Our long answer followup question indicated that confidentiality, uncertainty in one's authority, and unfinished information were the primary reasons to avoid sharing work artifacts with outsiders. Particular product features may be embargoed to small teams during early phases of development. This helps to prevent early designs from being revealed and possibly misconstrued as final product guidance by others at the company, or by outsiders. In addition, we had anticipated that social biases like ingroup bias and the illusion of transparency might affect how much sharing with which an individual would be comfortable, and we saw that in this question's other two categories of responses as well.

Implications

We hypothesize that engineers are tailoring their information sharing behavior to the organizational characteristics and work group range of the possible recipients of their communication. We ask the question, then, is this behavioral mediation deliberate, and how does it change in response to evolving organizational contexts? Studies of product innovation have found correlations of product success to increased cross-functional communication and violation of established barriers to communication because they allowed team members to combine their perspectives in a highly interactive and iterative way [21]. Researchers have hoped that communication behaviors could be shaped with appropriate communication practices and new tools. Prior work in communication tools aimed at software engineers incentivizes changes in communication behavior through tool use. For instance, Palantir [45], FastDash [8], and Syde [29] promote early, source code change conflict resolution communication by notifying engineers when they are working on the same file. Common repository-based notification systems attempt to keep engineers up-to-date on other engineers' changes through frequent emails, IMs or even new Web 2.0 technologies. Passive notification systems, such as Codebook [6] or GitHub [24], record repository changes in a persistent context-dependent newsfeed that may be accessed by interested engineers at their leisure.

When these tools are viewed from an enterprise perspective, we notice that their displays commonly require the recipients of information to make sense of automatically mined information presented in a form familiar to participants in the project, but often unknown, overwhelming, and/or irrelevant to outsiders. In addition, all of these systems presuppose that any and all engineers with an interest in the information will already have appropriate security credentials to see it. Tools such as TagSEA [50] and James [27] integrate tagging and microblogging into the IDE, enabling developers to broadcast their information to all, but do not give the sender control over who receives his message. Our study's results indicate that awareness tools should enable engineers to customize the message about their work to be more salient to those whom they *choose* to send it. Enabling the sender to tailor his message to be more salient for a particular audience, however, requires that the sender do more work than current practice typically presumes.

While engineers already use non-automated information sharing tools such as email and IMs, a semi-automated approach could help in a number of ways. To help engineers identify potential interested recipients, a system could use Socio-Technical Congruence [12] to discover probable, and potentially unknown, external dependents and enable the engineer to investigate their organizational characteristics, including whether they have access to the original software repository

in which the artifact originated. To help engineers craft an appropriate message, a system might indicate which recipients are most strongly tied to the sender. These recipients would be permitted to receive more detailed information on the rationale behind a change, or a frank assessment of a project's likelihood for shipping on time (which our probe surveys show would be closely guarded secrets). The system could group recipients by various organizational characteristics, since they often share particular information needs. For example, when a developer informs engineers of his checkins, other developers will read the source code diffs to review the effects on the code, but testers may simply care how much closer this checkin brings the project to the time where they may begin black box testing the software component. In addition, since some interested recipients have more seniority, more power, or tighter deadlines than the others, a tool that visually represented dependents' relative organizational characteristics could also help the engineer to craft an appropriate message to each.

Threats to Validity

Construct Validity: The dependency relationships reported by the respondents are the respondents' *perceptions* of their dependencies, not their actual dependencies. Dependency relationships were not verified, as respondents were not asked to identify them. Respondents selected work artifact types from a list (with an "Other" option) generated through past studies at Microsoft and verified through preliminary surveys and interviews. We operationalized engineers' sharing through measuring the *sharing metric* on the *work group range* scale. We believe this is the best approximation, thus far, of this concept, and encourage other researchers verify its validity.

Internal Validity: Our results about organizational characteristics may be confounded with demographic factors that we did not collect (e.g. age, gender, native language, or country of origin). Years of experience and the ratio of interdependent work involved per job role may be confounded, thus we required participants to have at least one year of employment. Survey participation was voluntary and anonymous, though some may have answered solely for the US$250 draw. No invited participant shared a common manager with the research team. We believe that participants answered sensibly to the intermittent long answer questions in the survey. All questions were optional, allowing respondents to complete only part of the questionnaire. Although the number of respondents answering each long answer question did drop slightly towards the end of survey, those who completed the survey appear to have given their answers some thought. Thus, we believe respondents either answered truthfully, or did not answer at all. We triangulated our results by correlating the results of oDepend with a rephrased version of oDepend at the end of the survey.

External Validity: This study is based on a single large US-based software company, so the results may not a priori generalize to other large software companies. However, studies of other large enterprise software companies [52, 20] show that they share a similar work environment, and would likely have similar results to ours. Microsoft's communication culture maintains a significant reliance on email, even as it enables contextualized communication within software repositories (e.g. source code and bugs [6]) and begins to experiment with Web 2.0 communication technologies. Since Microsoft is an enterprise-level, profit-oriented organization, observations and analyses of open-source community information sharing and altruism may have limited explanatory power here.

FUTURE WORK AND CONCLUSIONS

In this paper, we reported on a survey we conducted to understand three gaps in inter-team coordination. First, we learned that an engineer's job function and years of experience are significant determinants of his perception of dependency. Second, we found that these dependencies differed significantly by work artifact, and were always asymmetric. Third, we found that organizational rules, along with the type and sensitivity of particular shared work artifacts mediate the work group range to whom an engineer is willing to communicate. We also discovered a surprising result that begs further study. Engineers at this company appear to be altrustic because they report that they share more widely when they depend on information from others. Why? As Grudin asked about calendars [25], what are the social factors that incentivize or inhibit engineers at this enterprise company from sharing information with colleagues?

The results have important implications for tool designers, guiding them to explore ways in which information sharing tools can enable engineers to share their work artifacts automatically, yet be able to select the intended recipients and make the message salient for each. We hope that as new tools are introduced with increased understanding of the effects of organizational characteristics, dependency perception, and attitudes around information sharing, they can help make inter-team coordination function better.

ACKNOWLEDGEMENTS

We thank the HIP team at Microsoft Research, Steve Easterbrook, and our anonymous reviewers for the contributions to our study. We thank our study participants for their time.

REFERENCES

1. Ancona, D. G., and Caldwell, D. F. Bridging the boundary: External activity and performance in organizational teams. *Administrative Science Quarterly 37*, 4 (1992), pp. 634–665.
2. Anvik, J., and Murphy, G. C. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *IEEE TSEM 20*, 3 (2011).
3. Aranda, J., and Venolia, G. The secret life of bugs: Going past the errors and omissions in software repositories. *Proc. ICSE '09*, 298–308.
4. Bacchelli, A., Lanza, M., and Robbes, R. Linking e-mails and source code artifacts. *Proc. ICSE '10*, 375.
5. Begel, A. Effecting change: coordination in large-scale software development. *Proc. CHASE '08*, 17–20.
6. Begel, A., Khoo, Y. P., and Zimmerman, T. Codebook: Discovering and Exploiting Relationships in Software Repositories. *Proc. ICSE '10*.
7. Begel, A., Nagappan, N., Poile, C., and Layman, L. Coordination in large-scale software teams. *Proc. CHASE '09*, 1–7.

8. Biehl, J. T., Czerwinski, M., Smith, G., and Robertson, G. G. Fastdash: a visual dashboard for fostering awareness in software teams. *Proc. CHI '07*, 1313–1322.
9. Brown, S. L., and Eisenhardt, K. M. Product development: Past research, present findings, and future directions. *The Academy of Management Review 20*, 2 (1995), pp. 343–378.
10. Casey, V. Developing trust in virtual software development teams. *J. Theor. Appl. Electron. Commer. Res. 5* (August 2010), 41–58.
11. Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D. Software Dependencies, Work Dependencies, and Their Impact on Failures. *IEEE TSE 35*, 6 (2009).
12. Cataldo, M., Wagstrom, P. A., Herbsleb, J. D., and Carley, K. M. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. *Proc. CSCW '06*, 353–362.
13. Conway, M. E. How Do Committees Invent? *Datamation 14*, 4 (1968), 28–31.
14. Curtis, B., Krasner, H., and Iscoe, N. A Field Study of the Software Design Process for Large Systems. *Comm. ACM 31*, 11 (1988), 1268–1287.
15. Damian, D., Kwan, I., and Marczak, S. *Collaborative Software Engieneering*. Computer Science Editorial Series. Springer-Verlag, May 2010, ch. Requirements-Driven Collaboration: Leveraging the Invisible Relationships between Requirements and People.
16. de Souza, C. R., Quirk, S., Trainer, E., and Redmiles, D. F. Supporting collaborative software development through the visualization of socio-technical dependencies. *Proc. GROUP '07*, 147–156.
17. de Souza, C. R., and Redmiles, D. F. The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor? *IEEE TSE 37* (2011), 325–340.
18. de Souza, C. R. B., and Redmiles, D. An Empirical Study of Software Developers' Management of Dependencies and Changes. *Proc. ICSE '08*.
19. de Souza, C. R. B., Redmiles, D., Cheng, L.-T., Millen, D., and Patterson, J. How a good software practice thwarts collaboration: the multiple roles of apis in software development. *Proc. FSE '04*, 221–230.
20. DiMicco, J., Millen, D. R., Geyer, W., Dugan, C., Brownholtz, B., and Muller, M. Motivations for social networking at work. *Proc. CSCW '08*, 711.
21. Dougherty, D. Interpretive barriers to successful product innovation in large firms. *Organization Science 3*, 2 (1992), 179–202.
22. Fritz, T. Determining Relevancy: How Software Developers Determine Relevant Information in Feeds. *Proc. CHI '11*.
23. Fritz, T., and Murphy, G. C. Using information fragments to answer the questions developers ask. *Proc. ICSE '10*, 175–184.
24. Github, I. Github social coding, 2011. Available at: http://www.github.com.
25. Grudin, J. Groupware and social dynamics: eight challenges for developers. *Comm. ACM 37* (January 1994), 92–105.
26. Gutwin, C., Greenberg, S., and Roseman, M. Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. *Proc. HCI on People and Computers XI '96*, 281–298.
27. Guzzi, A., Pinzger, M., and van Deursen, A. Combining micro-blogging and ide interactions to support developers in their quests. *ICSM, Early Research Achievements Track* (2010), 1–5.
28. Halverson, C. A., Ellis, J. B., Danis, C., and Kellogg, W. A. Designing task visualizations to support the coordination of work in software development. *Proc. CSCW '06*, 39–48.
29. Hattori, L., and Lanza, M. Syde : A Tool for Collaborative Software Development. *Proc. ICSE '08*, 235–238.
30. Herbsleb, J. D., and Grinter, R. E. Splitting the Organization and Integrating the Code: Conway's Law Revisited. *Proc. ICSE '99*, 85–95.
31. Hinds, P., and McGrath, C. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. *Proc. CSCW '06*, 343–352.
32. Holmes, R., and Walker, R. J. Customized awareness: recommending relevant external change events. *Proc. ICSE '10*, 465–474.
33. Jiang, H., and Carroll, J. M. Social capital, social network and identity bonds: a reconceptualization. *Proc. C&T '09*, 51–60.
34. Kraut, R. E., and Streeter, L. A. Coordination in Software Development. *Comm. ACM 38*, 3 (1995), 69–81.
35. Kwan, I., Schroter, A., and Damian, D. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE TSE 37* (May 2011), 307–324.
36. LaToza, T. D., Venolia, G., and DeLine, R. Maintaining mental models: a study of developer work habits. *Proc. ICSE '06*, 492–501.
37. Lim, S., Quercia, D., and Finkelstein, A. StakeNet: using social networks to analyse the stakeholders of large-scale software projects. *Proc. ICSE '10*, 295–304.
38. Maalej, W., and Happel, H.-J. From work to word: How do software developers describe their work? *Proc. MSR '09*, 121–130.
39. Mockus, A., Fielding, R. T., and Herbsleb, J. D. Two case studies of open source software development: Apache and mozilla. *ACM TOSEM 11*, 3 (2002), 309–346.
40. Mockus, A., and Herbsleb, J. D. Expertise browser: a quantitative approach to identifying expertise. *Proc. ICSE '02*, 503–512.
41. Omoronyia, I., Ferguson, J., Roper, M., and Wood, M. Using developer activity data to enhance awareness during collaborative software development. *Journal of CSCW 18* (2009), 509–558.
42. Poile, C. *Asymmetric dependence and its effect on helping behaviour in work groups*. PhD thesis, University of Waterloo, 2010.
43. Rigby, P. C., and Storey, M.-A. Understanding broadcast based peer review on open source software projects. *Proc. ICSE '11*, 541–550.
44. Safayeni, F., Duimering, P. R., Zheng, K., Derbentseva, N., Poile, C., and Ran, B. Requirements engineering in new product development. *Comm. ACM 51* (March 2008), 77–82.
45. Sarma, A., Noroozi, Z., and van der Hoek, A. Palantír: raising awareness among configuration management workspaces. *Proc. ICSE '03*, 444–454.
46. Sarma, A., Redmiles, D., and van der Hoek, A. Categorizing the spectrum of coordination technology. *IEEE Computer 43* (2010), 61–67.
47. Sinha, V. S., Mani, S., and Sinha, S. Entering the circle of trust: developer initiation as committers in open-source projects. *Proc. MSR '11*, 133–142.
48. Sodan, A. C. How much do technical scientists really cooperate? *ACM SIGCAS 36*, 2 (June 2006).
49. Teasley, S. D., Covi, L. A., Krishnan, M. S., and Olson, J. S. Rapid Software Development through Team Collocation. *IEEE TSE 28*, 7 (2002), 671–683.
50. Treude, C., and Storey, M.-A. How Tagging Helps Bridge the Gap Between Social and Technical Aspects in Software Development. *Proc. ICSE '09*.
51. Wolf, T., Schroter, A., Damian, D., and Nguyen, T. Predicting build failures using social network analysis on developer communication. *Proc. ICSE '09*, 1–11.
52. Wu, A., DiMicco, J. M., and Millen, D. R. Detecting professional versus personal closeness using an enterprise social network site. *Proc. CHI '10*.
53. Xiang, P. F., Ying, A. T. T., Cheng, P., Dang, Y. B., Ehrlich, K., Helander, M. E., Matchen, P. M., Empere, A., Tarr, P. L., Williams, C., and Yang, S. X. Ensemble: a recommendation tool for promoting communication in software teams. *Proc. RSSE '08*, 2:1–2:1.