Facilitating Enterprise Software Developer Communication with CARES

Anja Guzzi
Delft University of Technology
Delft, The Netherlands
a.guzzi@tudelft.nl

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

Jessica K. Miller
Microsoft Research
Redmond, WA USA
jessica.miller@microsoft.com

Krishna Nareddy
Microsoft Research
Redmond, WA USA
knareddy@microsoft.com

***Abstract*—Enterprise software developers must regularly communicate with one another to obtain information and coordinate changes to legacy code, but find it cumbersome and complicated to determine the most relevant and expedient person to contact. This becomes especially difficult when the relevant person has transferred teams or changed their personal contact information since contributing to the project. We conducted a year-long series of surveys and interviews to help us learn how, why, and how often software developers discover and communicate with one another. In response to what we saw, we designed, deployed, and evaluated a domain-specific, IDE-embedded, photo-oriented, communication tool. We overcame a significant challenge found in long-lived projects: uniquely identifying individuals years after their contributions to the project. After deploying our tool, iteratively refining it, and deploying it again on a company-wide scale, most users reported that it simplified the process of finding and reaching out to other developers and offered them a sense of community with their colleagues, even if those colleagues did not currently work on their team. The lessons learned from our study and tool development should apply to other large, multi-team, legacy software projects.**

***Keywords*-communication, software engineering, longitudinal empirical study, coordination**

I. Introduction

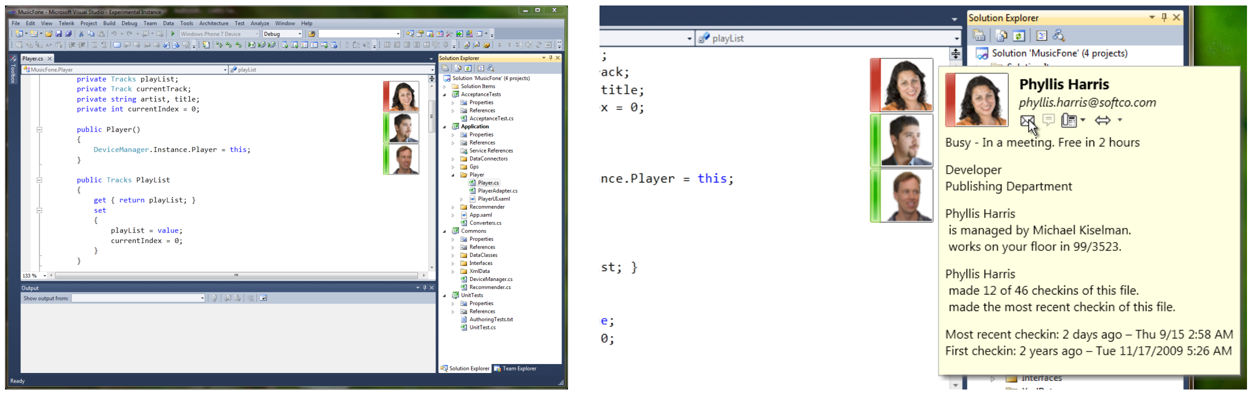
Succesfully developing and maintaining software products requires effective communication between dependent engineers. Our research explores the challenges of communication at Microsoft, where software products are extremely large, long-lived, and developed by non-collocated product teams. Software engineers must often maintain software written by developers who have left Microsoft, or moved to other teams. They try to rely on specifications, documentation, and source code to answer their questions, but in the end, engineers prefer to speak with knowledgeable experts or people with the authority to coordinate actions they need to get their work done [1], [2], [3], [4]. Unfortunately, when there is too little information shared between dependent engineers about the project's status and changes, the work relationships required for fluid collaboration suffer and threaten the project's success [5].

Grubb and Begel studied the causes of the paucity of inter-team communication and found that software developers felt inhibited from sharing information about their work [6] with people in other teams, products, and divisions. When asked why, engineers reported an aversion to "spam" other engineers with work notifications they might not be interested in. The asymmetry of dependencies on software teams and the modular boundaries induced by their software architecture [7] sometimes prevent engineers from noticing that others depend on their work. If an engineer thought no one cared about their status or changes, he would not even think to communicate at all.

We decided to tackle this problem from the reverse perspective, looking at the software developer who needs to communicate with a code owner who can explain some aspect of the software, the rationale behind it, or who had the authority to coordinate joint action to improve it. Over the course of a year, we conducted surveys and interviews to better understand how and why Microsoft software developers communicate with one another, and how often they do so. We also discovered the criteria developers use to identify and choose a set of *relevant* people, how they select the most *expedient* person to contact, the means by which they contact that person, and how often their conversations led to positive working relationships.

We then designed, developed, deployed and evaluated a new communications tool to encourage them to communicate with one another and simplify the process of doing so. Our tool, CARES: Colleagues and Relevant Engineers' Support (shown in Figure 1), is a Visual Studio extension designed specifically for software engineers who want to communicate with others about source code. CARES displays a context-sensitive array of photos of the engineers who are most tightly connected to the code in each file currently being edited in the IDE. To help developers select the best person with whom to communicate, each photo has a tooltip that reveals the person's code history, organization, physical location, and current availability. Developers can then choose to meet the person face-to-face (F2F), or initiate contact using email, instant messaging (IM), A/V chat, application sharing, or screen sharing buttons right in the tooltip.

To build the tool, we found that we had to address a major pragmatic issue in enterprise communication; corporate longevity implies organizational change. Over many years of product development, many people have joined


```
private Tracks playList;
private Track currentTrack;
private string artist, title;
private int currentIndex = 0;

public Player()
{
    DeviceManager.Instance.Player = this;
}

public Tracks PlayList
{
    get { return playList; }
    set
    {
        playList = value;
        currentIndex = 0;
    }
}
```

Phyllis Harris
phyllis.harris@softco.com

Busy - In a meeting. Free in 2 hours

Developer
Publishing Department

Phyllis Harris
is managed by Michael Kiselman.
works on your floor in 99/3523.

Phyllis Harris
made 12 of 46 checkins of this file.
made the most recent checkin of this file.

Most recent checkin: 2 days ago – Thu 9/15 2:58 AM
First checkin: 2 years ago – Tue 11/17/2009 5:26 AM

MusicFone - Microsoft Visual Studio - Experimental Instance
File Edit View Telerik Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Windows Phone 7 Device
Debug
Player.cs
MusicFone.Player
playList
Solution Explorer
Solution 'MusicFone' (4 projects)
Output
Show output from:
Ready
(a)
(b)

Figure 1. (a) Screenshot of the CARES tool running in Visual Studio 2010. (b) A tooltip next to Phyllis' picture displays information about her. Phyllis made the most recent checkin, but we cannot ask her about it right now because she is in a meeting. We shall email her instead.

Microsoft, left Microsoft, and transferred between teams. Email addresses are recycled when employees leave; re-joining Microsoft does not necessarily result in receiving one's original email address, especially if hired in a different employee category than before (e.g. vendor vs. full time). The challenge for us was that Microsoft's software repositories identify individuals using email addresses. In order to support communication with people who had made contributions to the source code at any time in the past, we had to develop an algorithm that could map an email address and a checkin date to a unique individual, in spite of any changes in that individual's status or identifying "labels." CARES can properly label any employee and enable communication with them (as long as they work for Microsoft at the time you want to find them).

In this paper, we describe this year-long study and what we found about developer communication behaviors. We then reflect on the lessons we learned from the processing of designing and implementing the tool, and from measuring its suitability and impact in the enterprise. Next, we show how our studies and tool were inspired and influenced by the research literature, and conclude with our suggestions for researchers interested in developing usable software enterprise-oriented communication tools.

II. Methodology

In the work described here, we wished to explore in more detail how software developers communicate about their source code, in what situations they are willing to do so, and how often this occurrs. From our previous studies, we expected to find that ad hoc, asynchronous, intermittent communication was unsupported by tooling and would suffer the most from communication inhibitions.

Over a two week period at Microsoft in July 2011, we conducted a 50 question web-based survey (Survey 1) divided into four sections: demographics and three communication scenarios (described below). The questions were drawn from Begel *et al.*'s previous study of inter-team coordination [5] and were piloted with several developers before being deployed.[1]

Out of 500 randomly sampled Microsoft developers (5% of all developers) invited to take Survey 1, we received 94 valid responses (19% response rate). Invitees were incented to respond by a raffle for US$100. Demographically, the 94 respondents of Survey 1 had spent an average of 11.3 years (SD=7.5) in the software industry and 6.9 years (SD=5.2) at Microsoft. Most (68%) reported that they had previously worked at other companies. 97% of respondents were developers, and 90% of those were individual contributors (ICs).

After the responses were received, we realized that we should have asked some additional questions on communication frequency. We sent a supplemental survey (Survey 2) to respondents of the first survey who indicated they were willing to speak further on our topic. Survey 2 was sent to 32 respondents of the first survey, and we received 18 valid responses (56% response rate).

From the research literature, our own prior research, and what we learned from these surveys, we decided to build a tool to help with the problems that developers reported. We designed the CARES tool to help developers find and select the most relevant person to communicate with for their needs. To make that selection actionable, CARES supports initiating communication with the selected party.

We deployed CARES to the 32 people who received Survey 2 and gave them instructions on how to install and use it. After a few weeks, we emailed these pilot users and asked them if they would be willing to be interviewed about it. Eight users agreed after having used CARES from between one and three weeks. They were all developers (including two developer managers) and worked in six different Microsoft departments. Each worked on a team with four to nine people and regularly collaborated with

[1]To request a copy of the survey, please email Andrew Begel.

three to thirty engineers working on other teams. The first two authors of this paper interviewed each of those eight CARES users for an hour; one author asked questions and engaged the interviewee while the other recorded the interview and took copious notes. The interviews were transcribed verbatim when it improved the accuracy of the notes. In each interview, the interviewee was asked to demonstrate his or her use of CARES in his or her own workspace. The actions taken by the interviewee in the demonstration were written down by the note-taking author.

After the first deployment, the second, third and fourth authors improved CARES to take advantage of what was learned. Feature and user interface enhancements were added to improve CARES' compatibility and robustness with the Microsoft's development environments. In March 2012, CARES was publicized at an internal Microsoft research conference and made available to all employees at the company. To monitor CARES usage for our study, the tool included a logging facility to report feature usage.[2] As we write this paper in June 2012, CARES has been used by a total of 106 employees (excluding the original 30 pilot users and anyone associated with the development of the CARES tool). CARES continues to be used (defined as at least twice in the two weeks prior to the submission deadline) by 36 of those 106 users. We recorded a total of 4,943 log sessions. The most prolific user used CARES 411 times since installation.

In April 2012, we conducted a third survey (Survey 3) of the 87 known CARES users. In addition to demographic questions, we asked about their usage of the CARES tool, their understanding of the visualization, their perception of its utility and impact on their daily work, their subjective assessment of its features, and their suggestions for improvement. We offered no incentive to answer this survey. We received 24 responses (28% response rate), of which 19 were developers (79%) and five were testers (21%). Respondents spent 9.1 years (SD=6.9) in the software industry and 4.8 years (SD=4.1) at Microsoft. The log data show that ten of these survey respondents still regularly use CARES.

III. Developer Communication

One of our first research questions was to identify, in detail, why and how developers communicate with one another about source code. Our previous studies indicated that asymmetry of dependency was an important factor in mediating the quantity of communication, so we asked study participants questions that concerned both a forwards and backwards version of communication scenarios that we included in our surveys.

[2]File paths opened in the editor and employee names and email addresses visible in the UI of the tool are hashed before being written to the log. This identifying information is not necessary for our analyses of the data. Logs are collected automatically, but users can opt out in a Settings dialog.

Reasons for communication	**Resp. [%]**	**S1**	**S2**	**S3**
Coordination:				
Discuss a change I want to make to the code		**54**	36	26
Know if my use case was supported by the code		37	**42**	29
File a bug on the code		**33**	11	12
Know if a bug on the code was fixed		**22**	14	9
Propose a collaboration on a topic related to the code		12	**23**	10
Take ownership of the code		**9**	8	N/A
Ask them to make a change to the code		N/A	**20**	14
Seeking Information:				
Ask how the code worked		51	**65**	**65**
Ask why the code was written that way		**47**	33	29
Find out who wrote the code		**15**	4	12
Ask if the code had test cases		**8**	2	4
Learn more about the code because I used to work on it		**7**	1	4
Courtesy:				
Ask permission to make a change to the code		**20**	13	10
Let them know I filed a bug on the code		**16**	11	7

Table I
WHY RESPONDENTS COMMUNICATE ABOUT SOURCE CODE, DIVIDED BY CATEGORY, WITH RESPONSE RATE FOR EACH SURVEY SCENARIO: S1 (N=91), S2 (N=84), S3 (N=69). BOLDED REASONS ARE IN THE TOP FOUR OVER ALL. BOLDED NUMBERS INDICATE FOR WHICH SCENARIO THE REASON WAS CHOSEN MOST.

The first survey scenario (Sc1) asked respondents to consider the *most recent time* they needed to communicate with someone on another team about source code they saw in their IDE.[3] The second scenario (Sc2) asked about the most recent time someone on another team asked them about code they wrote. The third scenario (Sc3) asked about the most recent time someone on another team had asked them about code they did not own at the time. Table I lists the reasons (drawn from our previous studies and the research literature) respondents could check off to explain why they needed to communicate. In this report, we divide them into three categories: *coordination* (*i.e.,* communication requiring negotiation or extended interaction), *seeking information*, and *courtesy* (*e.g.,* notifying someone that you are about to change their code).

The most frequent reasons, marked bold, are discussing a code change, inquiring about support of a particular use case, and to learn how the code worked and why it was written that way. The top three reasons were highly correlated with one another — about half of respondents reported two of the three, while thirteen reported all three. These reasons are similar to those found in previous studies [8], [2]. The diversity of responses is interesting as well, as more than 80% (Sc1: 80% (N=90),Sc2: 91% (N=82), Sc3: 81% (N=67)) of respondents indicated that the specific conversation they referred to in their responses was typical for them.

[3]We ask about a specific event to avoid memory and generalization biases by respondents.

After deploying CARES, we expected that users would be inspired to ask us to extend its functionality. Survey 3 respondents asked for a way to easily see and communicate with other sets of people than just those who had made checkins. The most popular sets were developers who had code dependencies (i.e. called methods, used classes, etc) (36% N=22), and testers who wrote tests for code in the file (36% N=22). Some also wanted to see anyone who had ever made changes to any file in the same Visual Studio project or solution (18% N=22), and anyone who ever reviewed a checkin to the file (14% N=22). This indicates the need to investigate additional communication reasons and usage scenarios in the future.

A. *Selecting a Person*

We anticipated that when a developer wanted to talk about code, he would have a difficult time finding someone relevant to talk to. First, he must discover possible choices of appropriate people, and then pick the one he believes is most relevant and expedient for his needs. However, the majority of respondents (66%) said "it was easy to find someone relevant to communicate with." Upon reflection, we realized that most of their communication must be with people they already knew, so finding that person again would not be difficult. We believe this is the case because in Sc2 and Sc3, developers specifically indicated that they already knew most of the people that contacted them (Sc2: 73% N=82, Sc3: 58% N=69). However, when developers did *not* already know anyone relevant to their issue, the ones we interviewed complained that finding the right person was tedious.

Why pick this person? **Resp. [%]**	**S1**	**S2**	**S3**
I thought they owned the code	**56**	N/A	N/A
They thought I knew the owner	N/A	**63**	48
They contributed to the code	45	**60**	29
One of their teammates wrote the code	13	N/A	**30**
Their team owned the code	52	**54**	45
I already knew them, and was more comfortable talking to them first	26	N/A	**28**
I could not get in contact with the current owner	3	2	**12**

Table II
REASONS FOR CHOOSING A PARTICULAR A PERSON TO TALK TO, WITH RESPONSE RATE FOR EACH SURVEY SCENARIO: S1 (N=91), S2 (N=83) S3 (N=69). BOLDED NUMBERS INDICATE FOR WHICH SCENARIO THE REASONS WAS CHOSEN MOST.

We asked Survey 1 respondents how they choose a relevant person. In all three scenarios (see Table II), they indicated that code ownership and contribution are the most important criteria. Echoing the reverse sense from the previous paragraph, 26% (Sc1: N=91) said, that if possible, they would pick a person they already knew.

In our interviews, we explored the process further. All eight interviewees said that they search the source code checkin history first. From each checkin, they find the author's (committer's) email address, the explanation for the checkin, and the diff showing what changed. One interviewee explained that the owner is the person to contact because "ownership is important for understanding design rationale." On some teams, the owner is the committer with the most checkins. On others, "there is no single owner because a lot of people touch every file." A third group of teams contacts the author of the most recent change, instead of the owner: "I look at the [source code] history and find out who last modified the file...[The] last person makes more sense than number of times." Each developer's rules appeared to be team-specific, even among those working on different teams for the same product. This fits in with Microsoft's grass-roots software process culture, in which each team is encouraged to use whatever process works best for them, so it "would be hard to get the real owner based on what [my] team['s] practice is." Of course, if they see someone they already know, developers may pick that person over everyone else. "The closer to me the dev is, the easier it is [to talk to him]." We found it surprising that no interviewee ever spoke about searching for a subject matter expert about the code. Perhaps they expected that the information they sought could always be provided by anyone with knowledge about the file.

B. *Contacting a Person*

After finding a person, the next step is to contact her. Many survey respondents indicated that they initiated contact using multiple communication channels at the same time. Over all three scenarios, email was most common, followed by face-to-face contact, and IM (Sc1 (N=91): Email 86%, F2F 23%, IM 20%. Sc2 (N=83): Email 76%, F2F 31%, IM 22%. Sc3 (N=69): Email 72%, F2F 30%, IM 20%). Other channels were rarely used.

The choice of communication channel often depends on how well the developer knows a person, whether the person is currently available online, and how far away he is. One interviewee explained, "I mostly use email and IM. But, if they are on the same floor, sometimes I'll visit them in person. If I know him, I'll IM him. [But] if he's far away, I will ping or email. If I don't know him, I will email him." Another said, "I would IM or email. If [they are] not online, then I would email. Even if [they are] in the building, I will IM anyway. When they get back (if [they were] away) I will IM them back. If [they are] out of office, then I will send email." The person's job level also has an impact: "If [they are] low on the management chain, I talk to them in person. If high up in the chain, I send an email. It doesn't matter if I know them." Another interviewee who felt similarly explained. "Usually managers are busy... I'd send an email before [I would] knock on their door." When contacting a manager, they said they would ask for a more appropriate IC contact.

Thus, although developers and their teams have distinct "algorithms" for finding and selecting the most relevant and expedient person to speak with, overall, they seem to consider at least six criteria: ownership, checkins (most recent or most numerous), a preexisting relationship, physical distance, job level, and online availability.

The interviewees communicated most often with their own teams, all of whom were collocated on the same floor of their building, and frequently in the same hallway. However, conversations with developers on other teams occurred often as well; in Survey 2, respondents indicated that they contact other developers two to five times per work task, and that most contact someone at least once a day when doing coding tasks. Most responses to emails or IMs occurred immediately (31% N=87) or while the asker was still working on his task (56% N=87). Though we thought it would be more difficult to get a response from a file owner on another team, 73% (N=91) of Sc1 respondents said that they were happy with the timeliness of the response they received.

These kinds of conversations resulted in mainly positive impressions. Survey 1 respondents reported being satisfied with conversations they had for all of the scenarios (Sc1: 84% (N=91), Sc2: 94% (N=84), Sc3: 90% (N=69)). In Scenarios Sc1 and Sc2, most respondents agreed with the statement, "the outcome of the conversation we had is still relevant for me" (Sc1: 81% (N=90), Sc2: 83% (N=82)). Such frequent, positive, material communication is likely to help engineers build and maintain positive working relationships with one another.

IV. Lessons Learned: Design and Implementation

Over the course of the study, we learned a lot about tool-mediated developer communication at Microsoft. Continuous change in the organizational hierarchy and in the employees' personally identifying information (PII) stymied all of our naive attempts to map email addresses to people. This challenged us to develop a history-sensitive identity mapping algorithm. After our CARES tool was deployed (twice), we monitored its usage via surveys, interviews, and log analysis, and found how it fit (or did not fit) into a diversity range of developer communication workflows, and saw the impact it had on helping developers forge positive working relationships. In the following sections, we describe the most important lessons we learned from our experience.

A. Tool Design

In this section, we describe the design of the CARES tool. CARES is a Visual Studio 2010 extension which displays a vertical array of photos of engineers in the whitespace in the upper-right corner[4] of the current editor window (see Figure 1). This is done using a view-relative, embedded editor adornment (*i.e.,* a graphic effect layered on top of the text view that always remains at the same position relative to the window borders).

[4]Source code typically has a lot of whitespace on the right margin.

The photos shown are context-sensitive; in each editor, CARES shows the faces of just those engineers who contributed to the file (*i.e.,* checked in changes on any branch). This minimizes the set of people developers have to consider speaking with about the file's source code. This reduction can be significant, as product teams often employ hundreds of developers, and anyone, on any team, may have worked on the code in the past. Some of the interviewees worried that a few files in their product were edited by many colleagues, and they would not be able to see them all. During Deployment 1, we asked the interviewees to show us one of their own files that had many committers, but none could find any with more than five people. When some Deployment 2 users also complained about too many photos, we added the ability to switch to a more vertically compact, name-only view. Incidentally, this name-only view addresses a potential problem among those developers prone to making stereotypical judgments based on seeing someone's race, ethnicity, age, or gender in a photo. CARES respects employee's choice over how their photos are used by observing a Microsoft IT-standard opt-out option.

When the developer hovers over a person, a tooltip displays her name, email address, title, department, manager (because managers are often more widely-known than ICs), office location relative to the user (developers are more likely to walk down the hall to meet with someone than climb stairs), and a colored bar indicating her availability (taken from her IM and work calendar status). The tooltip also shows her historical contribution to the code: whether she made the most recent checkin, made the most checkins of all of the people shown, or added the file to the repository, and how many commits relative to the others she made to the file. Finally, the tooltip reveals the dates of her first and most recent checkins, enabling the user to figure out if she is currently working on the code, or has moved on.

Once the developer chooses a person, he can click on email, IM, A/V chat, Visual Studio application sharing, or screen sharing buttons in the tooltip to initiate contact. CARES helps contextualize the ensuing conversation by filling in the current file path, class, and method (if applicable) as the subject of the message.[5]

B. Identity

While building the tool, we found that linking the email address of a committer of a checkin in long-lived codebases to data describing the employee was a lot more complex than we thought. To the source code repository, from which CARES fetches checkin and shelveset metadata, the author is just an email address. In order to identify the name, contact info, and organizational information of the person

[5]For more information on CARES, please see *http://research.microsoft.com/cares*.

represented by that email address, we must look it up in our Active Directory employee database. This does not always succeed, however. The source code repository records past events, but Active Directory only has records for **current** employees. If the email address cannot be found, it may mean the employee has left the company, or the employee has changed his email address, or the email does not belong to an employee at all, but in fact, is a machine account that made the checkin on behalf of an employee or tool.

Consider this four-step example:

1. John Doe made a checkin in 1999 with the email address: jdoe@microsoft.com. He then left Microsoft for a startup in February 2000. Active Directory today has no record of a jdoe@microsoft.com, thus CARES would have no way to find out who jdoe@microsoft.com actually is.
2. In 2005, Jane Doe joined Microsoft and was assigned the unused email address jdoe@microsoft.com. When we now look up jdoe@microsoft.com in Active Directory, we get Jane Doe's contact information. This would cause CARES to show that Jane Doe contributed to the file six years before she joined Microsoft!
3. John Doe (the original) rejoins Microsoft in 2008 as a vendor. He receives the email address v-jdoe@microsoft.com — the "v" prefix identifies him as a vendor. An Active Directory lookup of jdoe@microsoft.com still returns Jane's contact info. CARES still believes the committer is Jane, even though John is a current employee.
4. Jane got married in 2011 and changed her name to Jane Public. She also changed her email address to jpublic@microsoft.com. Now, Active Directory will again have no record of jdoe@microsoft.com, thwarting CARES' lookup. Yet, John Doe does work at Microsoft, but is stuck with his vendor email address v-jdoe@microsoft.com. This leaves Microsoft with the right John Doe, but no way to link him to his former email address.

Implementing an Identity Map

To address these scenarios, CARES makes use of the Codebook [5] web service which maintains a graph of software process information mined from the software repositories used by product teams at Microsoft. This information includes people, checkins, bugs, documents, tests, etc. Crucially for CARES, Codebook retains and make available the entire history of all of the repositories. Every graph node in Codebook has the potential for recording a start date and end date when it was valid. Person nodes can store multiple sets of start and end dates, since people can leave and rejoin Microsoft many times. Each property of a graph node may be declared to be "revisable," which means it too retains a record of every value it ever had along with the date range for which the property had that value.

Codebook mines its employee data from Human Resources, rather than Active Directory. Human Resources uses a much more complex employee database that contains historical information about current, former, and contingent (vendors, interns, contractors, etc.) employees. They distinguish employees not by name or email address, but by a personnel number, which is unique to each individual and never reused. Even when a person leaves and rejoins Microsoft, they are assigned the same personnel number.

Unfortunately, the Human Resources database contains a lot of "dirty" data. Some data is missing when we expected it to be there (*e.g.,* the identity of the second author's manager is missing from the database for his first year of his employment). Some data is not applicable for a given date range (*e.g.,* a salesman working out of his home has no "office" phone number). And many date ranges themselves are inconsistent across different tables in the same database. We spent six months after building CARES learning how to clean the data into a coherent, consistent form. This cleaned data is what is stored in the Codebook graph.

CARES invokes a Codebook web service API that provides the personnel number for any email address when also supplied with a single date. On any given day at Microsoft, only one person has a particular email address, so the combination of email address plus date is unique. CARES looks up each committer (or shelver) using the date of the checkin (or shelveset), and then retrieves (using another Codebook web service API) the complete historical record of that person's employment at Microsoft.

Codebook does not contain all of the information about people. Active Directory alone contains the person's photo and their instant messenger email address, both needed by CARES. Another employee database contains an important opt-out bit that indicates whether the employee wishes to make their photo public to people inside Microsoft. We must combine information from all three data sources in order to properly display the CARES UI. Codebook contains every historical email address for a person and their personnel number, while Active Directory contains every employee's current email address and personnel number. We can use Codebook's information to obtain the personnel number, and use that to look up the photo and IM address in Active Directory. We can then use the current employee email address to look up the opt-out bit in the third database.

Building such a pair-wise data lookup model made it difficult to design CARES with an extensible architecture that can incorporate additional employee information databases [9].[6] However, if we could sufficiently modularize the employee lookup extensions, we could speed up CARES by parallelizing the lookups. Our solution is to use an AggregatePerson facade object which can dynamically aggregate and cache information from individual Person objects returned by each employee lookup extension. Each extension is requested to

[6]The CARES data model supports non-employees as well, enabling it to merge in information from third-party user authentication systems.

lookup an employee given a pair of an email address and a date. If found, the extension creates a Person object with whatever data it has available (including historical) and adds it into a CARES global Person Repository. Several of a person's properties can be used to uniquely identify them, either the personnel number, or a pair of a date range plus an email address, Exchange name (a person's name with qualifier, when necessary to distinguish the person from others with the same name), GUID, Windows Security ID, IM address. Whenever a Person is added into the Person Repository, its unique identifiers are intersected against those already in the repository. Whenever there is a match, those Person objects are aggregated into a single AggregatePerson facade.

For example, let us say that John Doe made a checkin. TFS identifies him as "jdoe@microsoft.com @ 1999-04-20." Active Directory identifies John Doe as "v-jdoe@microsoft.com @ current," as "John Doe @ current," and as "Personnel Number 1234567." Codebook identifies John Doe as "jdoe@microsoft.com @ 1997-01-03 to 2001-02-10," as "v-jdoe@microsoft.com @ 2008-04-20 - current," as "Personnel Number 1234567," "John Doe @ 1997-01-03 to 2001-02-10," and as "John Doe @ 2008-04-20 - current." The third database identifies John Doe as "v-jdoe@microsoft.com opt-in @ current." The Active Directory Person and Codebook Person overlap in the personnel number, so they are merged together. The email address of the third database's John Doe overlaps with the email address of the Active Directory Person, allowing it to be merged into the AggregatePerson object as well. Finally, the email address and date of the TFS Person overlaps with the date range of the Codebook person's first email address. This last match lets CARES associate its checkin with the John Doe AggregatePerson object and display his correct, current contact and organizational information in its user interface.

So, what if the person who made the checkin is found in Active Directory but not in Codebook? This can happen when employee checkins are gated by quality testing — all employee checkins in the same time period (e.g. hour) are grouped together and tested. If the tests succeed, a machine account then commits all of the checkins to the repository. Machine accounts are Windows principals, and thus exist in Active Directory, but since they are not employees, they do not exist in the Human Resources database that supplies data to Codebook. We noticed that a secondary way to confirm the email address belongs to a machine account is that its Active Directory record contains no manager.

Sometimes an email address from a checkin cannot be found. If the Human Resources employee database were infallible, we would be able to conclude it is a machine account that is no longer used. However, the HR database has some missing rows in its email address to personnel number table, preventing us from linking his contact information to his email address. To reduce the incidence of these "missing" employees, we have manually curated 1,500 former employee's table entries, using manual inspection. The names and email addresses are often very similar to one another making them easy for a person to identify.

V. Lessons Learned: Deployment

Many developers at Microsoft use Visual Studio, so it was a natural choice to host the CARES extension. CARES uses Visual Studio's Managed Extensibility Framework (MEF), which enables plugins to easily extend Visual Studio (similar to what is possible with Eclipse) and allows plugins to be extended as well. CARES supports two kinds of extensions: source control repository access and employee metadata crawling. When a file is opened, CARES asks each source control extension to check if the file is managed by that source control system. If yes, the extension asynchronously populates a list of email addresses and associated dates for each checkin and "shelveset" recorded for the file.[7]

We knew from the CSCW literature and our own past experience deploying tools at Microsoft that requiring any configuration steps, server setup, or any kind of waiting time between installation and use would hurt CARES' adoptability [10]. We took advantage of Visual Studio's built-in source control connections to obtain information about committers and their contributions to the code. We use Windows' connection to Active Directory (available at Microsoft and many other enterprises) to retrieve employee information. We piggyback onto the user's active Microsoft Lync unified communications session to provide our various communication modes. When the user is disconnected from the corporate network, or disconnected from Lync, CARES gracefully degrades the user interface by disabling the features that rely on those servers without affecting the others. For example, if the user's connection to the source code control server is severed, CARES cannot fetch the checkins and committers at all. If a new file is opened, CARES will display an error message to the user for five seconds and then fade out. If the connection comes back, CARES will reactivate and attempt to initialize itself again.

A. CARES Usage Case Studies

We deployed CARES twice, first in a pilot to 30 developers, and then in a Microsoft-wide internal release. To this date, it has been used by 106 additional employees. The reaction to CARES by most ICs has been primarily positive, however, a few people felt that it did not fit with the ways they preferred to communicate with others. In this section, we describe both the positive and negative reactions to CARES, illustrating the diversity of communication styles used by employees sharing the same role.

Eight pilot users were interviewed. All of them liked the CARES tool itself. "CARES is pretty cool" and "awesome."

[7]A shelveset is like a checkin, but is intended only to be temporarily held while it is code reviewed by others.

One developer liked it enough to show it to his manager, who said, "[it] puts a face to the code. Now I know who to talk to." That manager went on to ask his entire team to start using CARES. All of the Survey 3 respondents said it was clear why the people who showed up in CARES were there (100% N=23).

They told us that CARES simplified and sped up their process for finding relevant engineers. One interviewee said, I "would use the CARES tool to get their name, email and contact card with their office address. [It] saves me time from running [a code history tool] and [an address book tool]." Another said "The more I can just stay here [in the IDE] where I'm doing my work, the better it is." A third said he "looks at [CARES'] availability indicator to confirm [that someone is] free" "before walking over. Green: yes, Red: message, Yellow: wait or email." 56% (N=18) of Survey 3 respondents reported that seeing the person's availability helped them to decide who to contact at that moment.

Developers spoke of situations where CARES had helped them. One said, "The add-in is helpful for me... There's lots of people who have implemented [code] in the past, and I have to understand them all." Another said it was "handy to know who worked on that particular code, especially when it was developed by someone on a different team years and years ago." Two others predicted that people who used CARES would end up asking *them* questions, one because he worked on the product's core which everyone else used, and the other because he was in the same team for many years, and had contributed to almost every file.

In the CARES design, we chose to show photos instead of names because we wanted to *encourage* communication. The photos are always visible, giving the developer the feeling that someone out there *cares* about his work, and is keen to be contacted about it. The CSCW literature backs up our intuition, suggesting that the visual cues provided by photos can help people identify individuals' relevant social categories and promote shared social identity among colleagues [11]. 48% (N=21) of Survey 3 respondents strongly agreed or agreed with the statement, "Seeing the faces of the contributors to the code helps me to feel like I am part of a community." 43% were neutral and only one respondent (5%) disagreed.

CARES' photos enable group members to easily recognize one another, which increases their sense of belonging. One interviewee said it was "definitely easier to figure out who the people are with the pictures." Another showed us how he investigated pointers to Visual Studio solutions that he received in his email. Having never looked at the solution's code before, and then opening it into his CARES-enabled IDE, he exclaimed, "that's [name omitted]! He's actually on my new team. It'd be real easy for me to talk to him. [Pause] That would have definitely taken me longer without CARES. I would be trying to hunt people down." His feeling was shared by many in Deployment 2. 57% (N=21) of Survey 3 respondents strongly agreed or agreed with the statement "For those people whom I had met before, seeing their faces in CARES helped me to recognize them."

Not everyone found that CARES was the right tool for them or their team. While 81% (N=21) of Survey 3 respondents reported that CARES showed enough information to understand the recency of a person's contribution to the file, only 39% (N=23) felt there was enough information to see the *magnitude* of a person's contribution to the file. Five Survey 3 respondents requested that CARES show line-by-line attribution each file, rather than aggregate all contributors to the file. We have been reluctant to add this feature since it would duplicate the functionality of the "annotate" function (*i.e.,* blame) in Visual Studio with TFS. Showing relevant people per line also presents a user interface challenge because you do not want your tool to attract the user's attention with UI changes when it is irrelevant to the user's main task. [12].

In our interviews, we spoke with one developer who said that while it was useful to know which developer in India wrote some code, he still preferred to contact the developer's team *liaison* in India to ask questions. He explained that his way enabled the liaison to delegate his question to anyone who was relevant and available to answer the question. With our tool, if the chosen developer was unavailable, or out of office, the answer would have been delayed at least 24 hours.

Another developer, the most senior member of his team, said that people came to him to find out who to talk to. "I could gauge whether their question was appropriate, or in my estimation, they should have done more homework." He worried that people on his team would use CARES to speak directly to more relevant colleagues without "demonstrating that they had any mental model at all." He noted further that CARES was of no use to him because "it's almost always clear who the author is, or who the right person is to contact... I know, based on my years of experience... who the author is."

In Deployment 2, CARES employed logging to record developers' interactions with the tool. We have not yet analyzed the detailed data, but have processed basic usage data (shown in Figure 2). In its second deployment (ongoing), CARES has been used by 106 software engineers all across Microsoft (not including Deployment 1 users). More importantly, CARES was not just used once or twice, but continues to be used on a regular basis by 36 of those engineers. Most small utilities deployed in an enterprise lose users quickly, due to serious bugs, overcomplexity, or user boredom; that CARES has remained installed and used by developers bodes well for its future.

VI. Related Work

Coordination studies have a long history in software engineering and CSCW research. Sarma [13] presents a comprehensive review of coordination tools, with features

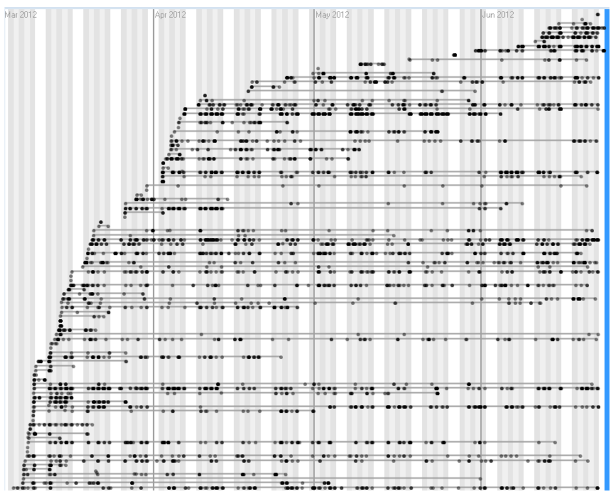


Figure 2. Usage logs of CARES users. The X axis is the date and the Y axis shows each unique user. A dot on a row shows when the person used CARES in his or her Visual Studio session. The timeline begins on March 8, 2012 and ends on June 24, 2012.

ranging from live team awareness (Palantir (ASE 2007), TUKAN, FastDash, Jazz, CollabVS), to top-down socio-technical exploration (STC) (Expertise Browser, Palantir (ICSE 2003), FastDash, Ariadne and Tesseract (both use Cataldo's STC), BeeHive, Codebook), to IDE-based tools (IBM Rational Team Concert (née Jazz), CollabVS, Palantir, TUKAN [14], Deep Intellisense, TeamTracks), to context-sensitive tools (TUKAN, TeamTracks, Deep Intellisense).

Unlike the live awareness tools, CARES aims at developers on large enterprise software teams working independently on components of a software product [2], [8], [15], [16], [17]. Each developer may care little of the daily work of people on other teams (if they know them at all), except when a coordination request or information need arises. As in Costa *et al.*'s study [17], the products we studied are also long-lived; relevant contributors may have moved on to unrelated projects. This reduces the need to see others' current activities and allows a simpler display of availability in line with the literature's recommendations [12]. While some of CARES' design elements overlap with Schummer's awareness design patterns [14], other aspects harmonize with Nakakoji's guidelines for communication with experts: being personalized, contextualized, and socially aware (though we do not limit users' communication modes) [4].

CARES is context-sensitive and contextualized to the IDE, only showing the people relevant to the user's focus. This design (also used by TUKAN and Deep Intellisense [18]) requires a simple glance and mouse hover to choose with whom to communicate. Expertise Browser, Palantir, FastDash, Ariadne, and Tesseract, however, use search, browse, or information-pivoting operations to move from a global view to the desired context. Jazz, CollabVS, and Palantir always show everyone on the team, while TeamTracks anonymizes the people and displays their aggregated IDE actions. Only Ariadne and CARES show how people are related to one another and to the tool user. Ariadne shows a similarity-metric-based person graph, while CARES shows concrete relations, such as relative office and organizational location, and relative code contributions.

There are many possible methods to select relevant people to show (e.g. organizational charts, program analysis, whole-system STC graph analysis [19], email [20], degree of interest functions [21], [22], and newsfeeds [23]). Our CARES prototype uses committers as a simply-computed, ecologically valid proxy for ownership and knowledge. This choice drastically improves CARES' deployability by avoiding the need for offline analysis and custom servers (which are required by many of the other tools). CARES is auto-configured from the project's source control context and works immediately after installation.

While many coordination studies have looked at why software engineers communicate and pondered the implications of too little or too much communication, ours is the first to discover the criteria engineers use to select the right person with whom to speak. In addition, very little of the communication and coordination tool literature offers longitudinal data about tool use over the long term, whereas our report describes two deployments over a total of five months of tool use at a large software enterprise.

VII. Threats to Validity

External Validity: CARES was designed for individual software developers working within large, long-lived software teams. However, the diversity of inter-team communication behaviors we saw makes us optimistic that its design will apply to large software teams at other organizations. Many Microsoft teams consist of collocated groups collaborating together from different locations across the globe. CARES may require additional study to adapt it to fully distributed software teams, such as those in some open-source projects. Identity information for some kinds of developers may be difficult, inappropriate, illegal, and/or impossible to obtain, preventing the CARES UI from offering a view of user's place in an organization. We have tried to make our design principles and deployment context clear in the exposition to enable others to adapt them to their particular situations.

Internal Validity: Participation in all surveys and in the second deployment was voluntary and non-anonymous. Some may have answered Survey 1 solely for its US$100 draw. Survey 2 and Deployment 1 participants were drawn from Survey 1 respondents who identified interest in participating in the study, used Visual Studio and Lync, and were currently developing code. Survey 3 respondents were drawn solely from CARES users who began using it during Deployment 3. No pilot users were included.

VIII. Conclusion

Enterprise software development is notable for large numbers of engineers working for long periods of time on projects with significant amounts of legacy code. At Microsoft, we noticed that ad hoc, asynchronous, intermittent communication between software developers about the source code was common, yet poorly supported by the general communication tools in daily use: email and IM. We found, after several months of study, the methods and "algorithms" that software engineers use to discover and select relevant people to speak with about their code. We learned that although their communication was intermittent, it was a key factor in establishing long-lasting work relationships that help make future collaborations less difficult.

By building an IDE-based tool to specifically support the person discovery and selection process, we realized that modeling employee identification in a large long-lived enterprise was very complex, and was often made so by the multitude of complementary, yet inconsistent employee metadata databases maintained by various corporate departments. Correlating information and dates across the databases was essential to uniquely identifying individuals who had long ago contributed to source code repositories but had since changed their organizational affiliation or PII. Our techniques can be used to disambiguate or "unify" individuals across many different kinds of PII metadatabases in an efficient manner.

After distributing our tool twice (once for three weeks and once for four months (ongoing)) we confirmed that deployability is strongly influenced by ease of installation, simplicity of use, and effectiveness at a single task. We plan to continue studying the impact CARES has on developer to developer communication at Microsoft. Our long-term goal is to learn about and support communication scenarios between developers and non-developers. With communication comes cooperation and trust, and with both comes more effective and successful collaboration.

References

[1] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *ICSE'10*, Cape Town, South Africa, 2010.

[2] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *ICSE '07*, Minneapolis, MN, USA, 2007, pp. 344–353.

[3] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *SIGSOFT '06/FSE-14*. New York, NY, USA: ACM, 2006, pp. 23–34.

[4] K. Nakakoji, Y. Ye, and Y. Yamamoto, "Comparison of coordination communication and expertise communication in software development: motives, characteristics, and needs," in *JSAI-isAI '09*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 147–155.

[5] A. Begel, Y. P. Khoo, and T. Zimmerman, "Codebook: Discovering and Exploiting Relationships in Software Repositories," in *ICSE'10*, Cape Town, South Africa, 2010.

[6] A. M. Grubb and A. Begel, "On the perceived interdependence and information sharing inhibitions of enterprise software engineers," in *CSCW*, Bellevue, WA, USA, 2012, pp. 1337–1346.

[7] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "How a good software practice thwarts collaboration: the multiple roles of apis in software development," in *FSE*. Newport Beach, CA: ACM Press, 2004, pp. 221–230.

[8] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *ICSE*, 2006, pp. 492–501.

[9] A. Guzzi and A. Begel, "Facilitating communication between engineers with cares," in *Proceedings of ICSE, Tool Demo Track*, Zurich, CH, 2012, pp. 1367–1370.

[10] J. Grudin, "Groupware and social dynamics: eight challenges for developers," *Commun. ACM*, vol. 37, pp. 92–105, January 1994.

[11] M. K. Rabby and J. B. Walther, *Maintaining relationships through communication*. Mahwah, NJ: Lawrence Erlbaum, 2003, ch. Computer-mediated communication effects in relationship formation and maintenence, pp. 141–162.

[12] L. Dabbish and R. Kraut, "Research note—Awareness Displays and Social Motivation for Coordinating Communication," *Info. Sys. Research*, vol. 19, pp. 221–238, June 2008.

[13] A. Sarma, D. Redmiles, and A. van der Hoek, "Categorizing the spectrum of coordination technology," *Computer*, vol. 43, pp. 61–67, 2010.

[14] T. Schümmer and J. M. Haake, "Supporting distributed software development by modes of collaboration," in *ECSCW*. Bonn, Germany: Kluwer Academic Publishers, 2001, pp. 79–98.

[15] J. DiMicco, D. R. Millen, W. Geyer, C. Dugan, B. Brownholtz, and M. Muller, "Motivations for social networking at work," in *CSCW*, 2008, p. 711.

[16] C. R. B. de Souza and D. F. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *ICSE*, Leipzig, Germany, 2008, pp. 241–250.

[17] J. M. Costa, M. Cataldo, and C. R. de Souza, "The scale and evolution of coordination needs in large-scale distributed projects: implications for the future generation of collaborative tools," in *CHI*. Vancouver, BC, Canada: ACM, 2011, pp. 3151–3160.

[18] R. Holmes and A. Begel, "Deep intellisense: a tool for rehydrating evaporated information," in *MSR '08*, Leipzig, Germany, 2008, pp. 23–26.

[19] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," in *CSCW*, Banff, Alberta, Canada, 2006, pp. 353–362.

[20] R. Holmes and R. J. Walker, "Customized awareness: recommending relevant external change events," in *ICSE '10*, 2010, pp. 465–474.

[21] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *FSE*. Portland, OR: ACM Press, 2006, pp. 1–11.

[22] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *ICSE*, Cape Town, South Africa, 2010, pp. 385–394.

[23] T. Fritz, "Determining Relevancy: How Software Developers Determine Relevant Information in Feeds," in *CHI*, Vancouver, BC, Canada, 2011.