Expertise Browser: A Quantitative Approach to Identifying Expertise

Audris Mockus
Bell Laboratories
263 Shuman Boulevard
Naperville, IL, USA 60566
+1 630 713 4070
audris@mockus.org

James D. Herbsleb
Bell Laboratories
263 Shuman Boulevard
Naperville, IL, USA 60566
+1 630 713-1689
jherbsleb@bell-labs.com

ABSTRACT
Finding relevant expertise is a critical need in collaborative software engineering, particularly in geographically distributed developments. We introduce a tool that uses data from change management systems to locate people with desired expertise. It uses a quantification of experience, and presents evidence to validate this quantification as a measure of expertise. The tool enables developers, for example, easily to distinguish someone who has worked only briefly in a particular area of the code from someone who has more extensive experience, and to locate people with broad expertise throughout large parts of the product, such as module or even subsystems. In addition, it allows a user to discover expertise profiles for individuals or organizations. Data from a deployment of the tool in a large software development organization shows that newer, remote sites tend to use the tool for expertise location more frequently. Larger, more established sites used the tool to find expertise profiles for people or organizations. We conclude by describing extensions that provide continuous awareness of ongoing work and an interactive, quantitative resume.

Keywords
Collaborative software engineering, measures of expertise, expertise visualization

1 INTRODUCTION

Bringing the right expertise to bear on design and implementation issues is critical to the success of any engineering project. Communication cannot be effective unless engineers can identify the person with whom they need to communicate. Previous research has helped to clarify the amount of engineering effort devoted to communication. In particular, engineers in one classic study spent around 16% of their time in communicating with experts [2]. Interestingly, Allen [2] reported a tendency for higher-performing engineers to consult much more with experts *outside* their own discipline than did lower-performing engineers, although both groups spent about the same proportion of time overall communicating. Finding experts is critical, especially those outside one's group, whom one probably knows least well.

Research specifically with software engineers has also shown many times that engineers spend a large proportion of their time communicating. Herbsleb [12] indicates that early in a software project, engineers spend about half their time communicating, while DeMarco and Lister [5] estimate at least 70% of developer time is spent communicating with others. A substantial amount of this time [20] is spent in informal communication.

The task of finding needed expertise is especially difficult in geographically distributed teams, where barriers to effective communication exist due to geographic distance, cultural and time differences, and lack of face-to-face contact [3,6,10,11]. In fact, the difficulty of finding needed experts quickly has been identified as one of the major reasons that development work split across sites tends to take much longer than similar co-located work [9]. Studies indicate that engineers [2] and scientists [16] spontaneously communicate much less frequently with colleagues whose offices are distant from theirs, so there are many fewer opportunities to find out who has expertise in various areas when teams are distributed.

The problem of finding experts is not limited to widely-distributed teams, however. In fact, people whose offices are separated by 30 meters communicate about as infrequently as people who are located on different continents [2]. So one might expect that projects whose members are spread across a campus, or multiple floors of a single building, or even a long hallway, will experience much-reduced communication among the more widely-separated members.

One way to test the expectation that finding the right person is an important practical problem in development, is to look at the timing of work done on closely-related

changes to the software. Such changes, although assigned to a single person, often require that this person recruit one or more others, with complementary expertise, to complete the work. If finding such people is a problem, one would expect substantial delays in initially getting these additional people to work on the problem. We present results relevant to this question of the importance of finding experts.

Previous work suggests an approach to solving the expertise-finding problem. In an empirical study of finding experts in a software development company, Ackerman and Halverson [1] observed that experience was the primary criterion engineers ordinarily used to determine expertise. In fact, developers often used change history to identify who had experience with a particular file, generally assuming that the last person to change it was most likely to be "the" expert. This strategy had several shortcomings, including the inability to determine if the person who carried out the change had made a large or small change, and whether the person had made many or only a few changes in the relevant code. Additionally, when someone with breadth was needed, it was quite difficult to identify such people from the change information stored in individual files.

McDonald and Ackerman [17] report on a system that uses various heuristics to select an expert, based on who has touched various files, who is organizationally closest to the requester, and how well the requester knows the expert (based on a previous analysis of the social network in the organization). The idea is to produce a very short list of recommended experts based on heuristics specified by the user. If no satisfactory expert is identified, the user can "escalate," and the system will produce a larger list of potential experts, e.g., by changing threshold values in the heuristics.

The current project is part of a larger study of geographically distributed software development teams. In the interview data (reported on elsewhere, see, .e.g., [8,10]) we discovered that the techniques currently in use for finding experts in geographically distributed development groups are highly uncertain, time consuming and often impose an undue burden on certain individuals. Project architects, who were known to have a broad knowledge of the product and people, hence were already burdened with fielding lots of questions, had become clearinghouses, in effect, for information about expertise. These individuals' time is quite valuable, and spending substantial amounts of time helping others find experts took them away from their technical duties. Additionally, when such individuals were not available, there was no mechanism other than escalation through the management chain, often a slow and burdensome process, for finding experts.

Finally, we wanted to minimize the demands on users. In particular, we wanted to eliminate the need for people to write and maintain some description of their expertise. Our experience has been that such descriptions are generally difficult to use because different people often describe similar expertise differently, people have very different standards for judging the degree of their expertise, and such descriptions go out of date very quickly and are difficult to maintain. We also wanted to allow users to select experts whom they know, whenever possible, but we didn't want to impose the burden of collecting and maintain information about social networks and incorporating that in the tool. As previous research has shown, social networks in software development are highly volatile [13].

We wanted a tool that would meet the following requirements:

- identify experts quickly and easily,
- do not overburden a few individuals,
- allow the user to find alternatives when some experts were not available, and
- users need not create or maintain descriptions of their expertise, nor report information about their social networks.

Additionally, given previous findings, we decided that using change history seemed likely to be a sound approach, but we needed some way to quantify expertise so that

- potential experts could be compared with one another in terms of their "degree" of relevant expertise, and
- experts with the needed distribution of expertise, i.e., broad, as well as narrowly specialized experts could be identified very quickly.

Finally, we wished to explore the use of visualization techniques to allow users to browse through the available experts, based on their relevant characteristics, rather than pre-specifying heuristics and having the system make the selection.

In the next section, we define quantitative measures of expertise and describe how to obtain them from a software project's change management systems. Then we describe methods to identify experts for any part of the software. In a similar fashion we also define qualities of software based on the previous experience of people creating and maintaining it. In section 3, we introduce Expertise Browser (ExB) — a web-based tool to assist developers, testers, and managers in identifying experts for a number of software development tasks. The tool displays the relationship between parts of software and domain experts, using the experience measures. We briefly report on

deployment and usage of the tool in section 4. In section 5, we conclude with extensions of this approach, including 1) a variation of ExB designed to keep developers and managers aware of activity related to their own work, 2) a quantitative resume that summarizes a person's development experience in an interactive visualization, and 3) a refinement of our measure of expertise. We conclude in section 6.

2 MEASURES OF EXPERTISE

Expertise is difficult to measure or observe directly – the most direct measure, perhaps, is a test such as ones proposed for professional licensing. Moreover, there are many types of expertise, and possible taxonomies of expertise. For pragmatic reasons, we wanted to focus on the areas of expertise most often needed in the course of a large software projects. Based on extensive interviews we have conducted with members of distributed project teams, we found that people sometimes wanted to locate an expert in a particular technology or tool, e.g., a database person, or an ObjectTime person. We also found a frequent need for an expert in a particular part of a product, e.g., someone who knows the OA&M interface for a particular network element. Our approach needed to accommodate both of these sorts of experts.

In the remainder of this section, we define experience atoms (EAs) and identify several types of expertise in a software project that can be measured by EAs, by means of data extracted from software change management systems. Finally, we argue that EAs provide a reasonable, although not infallible, way of measuring expertise. We discuss several empirical results, one previously published and two derived from previously unpublished data, that tend to validate EAs as a measure of expertise.

2.1 Experience atoms

Experience atoms (EAs), are elementary units of experience. Experience, we assume, is the direct result of a person's activity with respect to a work product, enhancing it or fixing a problem. The smallest meaningful unit of such changes is an EA. We define the experience of an object as a collection of all such elementary units pertaining to that object. According to our definition, experience may pertain to a person, organization, or a work product such as a piece of code. The simplest unit of experience that could be observed in projects using change management systems is the atomic change (delta) made to the source code or to documentation. The person (and the person's organization) implementing the change gained a certain amount of experience by doing work required to change the particular part of a file. The changed work product provided a particular type of experience to a specific person.

Although we are mostly concerned with finding people with relevant experience, we might also pose other questions, e.g., What experience has a particular person or organization had? How do I find the work that would bring me the experience I want? As we will point out later, many other interesting kinds of questions might be asked and answered as well.

2.2 Software changes

The purpose of the typical work item in a software organization is to make a change to a software entity. Work items range in size from very large work items, such as releases, to very small changes, such as a single modification to a file. The source code of large software products is typically organized into subsystems according to major functionality (database, user interface, etc.). Each subsystem contains a number of source code files and documentation.

The versions of the source code and documentation are maintained using a version control system (VCS), such as Concurrent Versioning System [4] commonly used for Open Source Software projects, or a commercial system, such as ClearCase. Version control systems operate over a set of source code files. An *atomic* change, or *delta*, to the program text consists of the lines that were deleted and those that were added in order to make the change. Deltas are usually computed by a file differencing algorithm (such as Unix *diff*), invoked by the VCS, which compares an older version of a file with the current version. Included with every delta is information such as the time the change was made, the person making the change, and a short comment describing the change.

In addition to a VCS, most projects employ a change request management system (CMS) that keeps track of individual requests for changes, which we call Modification Requests (MRs). Whereas a delta is used to keep track of lines of code that are changed, an MR is intended to be a change made for a single purpose. In most projects, new functionality, corrections, adaptations, and perfective maintenance are all initiated by opening MRs. Each MR is assigned to one developer, and may have many deltas associated with it. The MR owner will often have to find other developers, with complementary expertise in other parts of the product, to make additional changes to complete the work needed for the MR. Some commonly used problem tracking systems include ClearDDTS from Rational and the Extended Change Management System (ECMS) [18]. Usually such systems associate a list of deltas with each MR.

2.3 Timing of changes – finding experts a critical problem?

As we mentioned in the previous section, we believe that the timing of changes in an MR can help to determine if in fact expertise-finding is a serious problem. When multiple people are needed on an MR, it is up to the MR owner to find the other people with the correct expertise. If this is

relatively easy to do, we would expect that the second and subsequent people to work on the MRs would be found quickly, and would begin making their contributions relatively early in the overall MR interval. If, on the other hand, finding experts is difficult and time-consuming, we would expect contributions from the second and subsequent contributors to an MR to begin work late in the MR interval.

To test this, we looked at all multi-person MRs and calculated where in the MR interval the second person begins working. Specifically, we examined two large projects, and counted how frequently the second person's first delta falls in the last ten percent of the overall MR interval. In project A, that happened 77% of the time and in project B, that happened 76% of the time. In both projects, the second and subsequent contributors began making their contributions very late in the overall MR interval. These results are what one would expect if finding experts is difficult and time-consuming. It appears that this is an area where there is enormous potential for improvement.

2.4 Domains, types, and measures of experience

There are multiple domains of software experience. A particular EA may or may not belong to a particular domain. The experience domain may, for example, be represented by one or more of the following properties of the delta:

- Software delivery where the delta belongs. That may be a new release of software, a patch, a bug fix, or other delivery.
- Functionality of the product part (file, module, subsystem) touched by the delta. Different parts may define different functional areas, e.g., GUI, database, etc.
- Technology used to do the work. It may involve programming language, development process, editors and other development tools and techniques.
- Purpose or type of change. Corrective changes fix problems, adaptive changes add new functionality, and perfective changes improve the structure and other properties of a software code.

Experience in any of these domains can be approximated by counting the number of relevant changes, i.e., the changes in a particular delivery; in a file, module, or area of functionality; made using a particular language or tool; or changes with a specific purpose, such as fixing problems. One can also identify current experience, by restricting domain experience to a specified (recent) period of time. One can also use the change management system to identify testing experience, e.g., to find a tester with experience in a particular domain, as one who has raised a substantial number of problem-reports on the relevant code.

2.5 Relationship between experience and expertise

Expertise is defined as the skill of an expert, and, if interpreted quantitatively, reflects the degree of the ability of a person to perform certain tasks. We would argue that being able to perform a task quickly, with minimal effort, and producing a high quality result are the key qualities of a software expert. We discuss one published study and two small, previously unpublished studies, that all tend to show that experience, as measured by EAs, tends to have the relationship one would expect with quality, productivity, and the judgments of other developers about who the experts are in particular domains.

A previous study [19] examined the relationship between EAs and the probability that a software delivery would cause a failure after deployment. One would expect that if EAs are a reasonable measure of expertise, that deliveries constructed by developers with a larger quantity of relevant EAs performed in the past should have a lower probability of causing a failure than a delivery constructed by a less experienced staff. The results showed this was in fact the case.

One would also expect that when developers start working on a new project, there should be evidence that their expertise increases as they accumulate EAs. This increased expertise should be measurable, for example, as greater productivity over time. An empirical estimate of such a curve is presented in Figure 1. The productivity over 50 developers who started working on the project within a three year period from 1995 to 1998 is measured by the average number of deltas completed by a developer in a month. The time is shifted for each developer to show their first delta occurring in month one. This allows us to calculate productivity based on the duration of developer experience with the new code.

The horizontal axis shows the length of a developer's experience on the project in months and the vertical axis shows the average number of deltas per month. The jagged curve represents monthly averages, while the smooth curve illustrates the trend by smoothing the monthly data. The figure shows that the time to reach full productivity (the learning curve flattens) is approximately 15 months. The data show the pattern one would expect if the accumulation of EAs does in fact provide a measure of expertise.

Finally, we did a qualitative study in order to see if experts identified on the basis of accumulated EAs are also identified as experts by their peers. In two development groups of about 50 developers, where we were conducting a joint project unrelated to Expertise Browser, we identified the three developers with the most deltas in the area of the code their group worked in. We were interested

in finding candidates for interviews to help us understand software development process in both groups. We then showed this list to two developers and two managers from each of the groups during individual conversations. We asked whether they thought that these three developers would be the best (i.e., the most expert) candidates for our interviews. In all cases we got complete agreement.

While these three studies do not provide definitive evidence that EAs are a valid measure of expertise, they do provide a substantial degree of support for this measure. In particular, we regard the support as adequate justification for using EAs as a basis for a tool that uses a quantification of expertise to overcome many of the limitations of other expert location techniques. We describe such a tool in the next section. An algorithm for generating a more sophisticated model of expertise, based on a statistical analysis of change data, is presented in the final section.

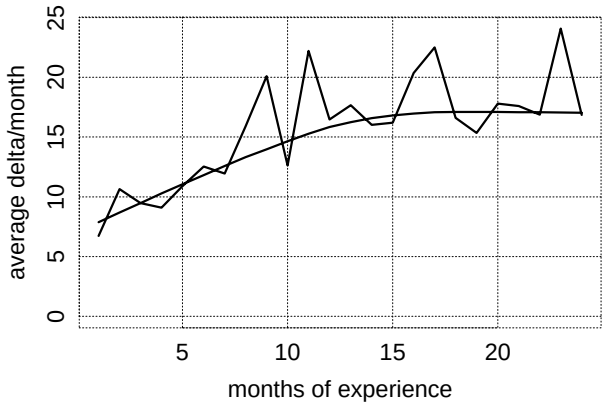

Figure 1: Learning curve.

3 EXPERTISE BROWSER

In this section we discuss ways to disseminate and present the expertise information obtained using the methods described in the previous section. The expertise information must be readily and easily accessible by developers, testers, and managers to help them rapidly locate the best experts for every task. Consequently the access and presentation methods are crucial in practice. To simplify deployment of expertise information we chose the Web as delivery mechanism. We designed the presentations in the form of HTML pages and embedded Java Applets, enhanced with a more traditional forms-based interface with search engine and hierarchical navigation. For more details see the LiveDocs framework in [14].

To visualize expertise information we created Expertise Browser (ExB) implemented as a Java Applet. The main idea behind ExB is visually to query and present relationships between product (code, documentation, design, functionality) and the people or organizations that have a desired type of experience with respect to these artifacts. There are two basic questions the ExB is designed to answer:

1. Who has appropriate expertise for a particular part of code, documentation, functionality, or delivery. We refer to that part as the product unit.
2. What is the expertise profile of a particular part of organization: a person, a group of people, or a group of organizations. We refer to that part as organization unit.

To answer the first question ExB provides display of, navigation among, and selection of an arbitrary product unit. Once the user has selected the desired product unit, all the EAs for that unit are summed for each organization unit involved. The participating organization units are displayed to the user with an indication of their relative expertise.

To answer the second question ExB provides display of, navigation among, and selection of an arbitrary organization unit. Once the user has selected the desired organization unit all the EAs for the selected organization unit are summed over each product unit the organization has worked on. Each product unit then displays the contribution of that organization unit relative to the total count of EAs related to the product unit.

To implement these basic ideas we need to divide the product and the organization into units and to provide display and navigation among these units. The product can be divided into hierarchical units in several ways, including the two we used as actual bases for deployed versions of the tool:

- a directory-induced product hierarchy, based on the source code directory structure, and
- a delivery-item induced product hierarchy, such as a release, update, or patch.

There are several possible ways to obtain the organizational division. We chose to use the existing department structure that could be obtained from organization charts, since that is the type of information most desired by users.

3.1 The views and relationships among views

To answer the basic expertise questions the ExB displays aggregations of EAs into product and organization units and displays the relationships among them. It also provides details on product and organization units including contact information for individuals. This structure leads to three main types of views:

- Product unit views show a hierarchy or a list of product units. Several views may be created to represent different interesting divisions of the product,

as described above.

- Organization views show a hierarchy or a list of organizations or individuals. Several views may be created to represent different organizational hierarchies.
- Detail views show information on a user-selected product unit, organization, or individual. In addition to information display, the detail views implement basic search capabilities to find and select a desired individual, organization, or a part of the product. Detail views also may show the time trends of the EAs of interest to the user.

Additional views may be added to represent the work units, including releases, features, and Modification Requests (MRs), or the different branches of the product.

Organization and product views contain a number of possibly hierarchically organized or otherwise related elements. Consequently, the view may be decomposed into a layout that determines the position of a particular element within a view and an element renderer that displays the element (product or organization unit). Section 3.2 describes basic user interactions. We consider effective ways to layout the elements in Section 3.3 and ways to display each element in Section 3.4 . Finally, we describe detail views in Section 3.5

3.2 User Interactions

The user poses one of the basic questions by selecting the unit or units of interest, for example, by clicking the mouse over the visual representation of the desired units. If the goal is to find experts on a particular piece of code, the user selects the product units in question. If the goal is to find the expertise profile of a particular group of individuals, the user selects these organization units. As a result of the selection action all EAs pertaining to the selected units are selected. If the unit is a piece of code, then all deltas on that piece of code are selected. If the organization unit is selected, all EAs pertaining to that unit are selected in the same manner (all deltas done by developers who belong to that unit).

The result of user selection automatically propagates to all other views (see, e.g., [21]), providing a visual answer to the user. The units in other views will in general be only partially selected, because only a subset of EAs pertaining to the each unit will be selected. For example, if a user selects one source code file (effectively selecting all deltas for that file), there might be several developers who contributed deltas to that file, and it is likely that some of them contributed deltas to other files as well. Consequently, developers in the organization view will be unselected (did not contribute to the file), partially selected (contributed some of their deltas to the file), or fully selected (all the deltas they made were in that file). This requires the units to be able to represent partial selection graphically.

Figure 2 illustrates a typical, brief interaction with ExB. The user wishes to find an expert on the "main" module, hopefully someone who has some breadth of expertise across the various files in the module. The code view on the left shows part of the source code tree. After expanding the "main" module node to display the individual files, the user clicks on the box labeled "main." This user action populated two organization views on the right: "Bug Reports" and "Code Commits" representing two types of expertise. The vertical size of a unit (a person or a code module) represents the number of EAs gained by the person or embedded within the module. The horizontal size represents the number of people who contributed deltas to the module or file.

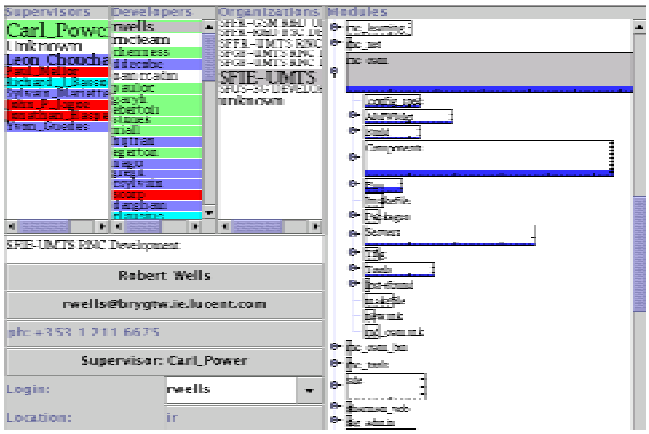

Figure 2: ExB user interface.

The user then selected a potential expert, the person in the "Code Commits" view at the top of the list. This is the person with the most EAs in the "main" module. This selection highlighted the proportion of EAs gained by the selected person on each code unit. The proportion is shown in yellow. From this highlighting, it is clear that the selected person has accumulated EAs from many of the files in "main," so appears to be a good candidate, i.e., broad expertise in the various parts of the "main" module. The relevant code unit and organization unit details are displayed at the bottom right.

In addition to posing the expertise questions, a user might desire to focus on a particular time period, a release of the code, or an organization unit. To facilitate such tasks, all undesired EAs may be hidden, together with all undesired product and organization units.

3.3 The layout

The role of the layout is to help the user easily locate the units of interest. Consequently, the layout should represent some aspect of the product's or organization's structure. The source code has an inherent tree-like directory

structure that breaks a system down into subsystems, modules, and files (there might be more or fewer levels in the hierarchy for a particular software product). We use SWING's JTree to implement such layout (see Figure 2). More exotic layouts have been proposed in the literature, see, e.g., [15]. Such layout strategy use similarity relations among units to determine their relative positions. For source code, however, the directory structure appears to be the most familiar and intuitive representation.

We found the simple item list layout (as used by swing's JList) to be sufficient to display an organizational view. Organizations could also be represented hierarchically. The organizations in which ExB was deployed, however, had only very limited hierarchy. In cases where units of code larger than network elements were involved, and larger parts of the organization are involved, a tree or other hierarchical layout might also be appropriate for the organizational view.

3.4 The elements

The visual representation of individual product units and organization units has several parts:

- identification of the unit (file's name, person's name);
- the total number of EAs that are associated with the unit;
- the fraction of EAs identified by the selections in other organization or product unit views. Several types of EAs may be shown simultaneously using, e.g., different color to represent different types of EAs as in a stacked bar chart.

The identification of the unit is shown as a textual label identifying the file, module, person or organization. The number of EAs pertaining to the unit is mapped to the height of the visual element representing the unit. The textual label may have the font size vary according to that height.

Because of the limited space on the screen, a linear mapping between the amount of EAs and the vertical size is not effective. Large units occupy most of the screen and smaller units have to be pixel or sub-pixel height. To eliminate that problem we use square root or logarithmic mapping, where the square (or exponent) of the height represents the number of EAs.

The selected subset of EAs is shown as a highlight covering a portion of the unit. The proportion of the unit that is covered is the same as the proportion of EAs selected in the unit.

3.5 The detail views

The detail views perform several primary functions:

- provide drill-down details on product or organization units selected in other views;
- provide textual search and select functions to identify desired product or organization units. This can be done by typing in a person's, organization's, or code unit's identifier in the corresponding entry of the detail view.
- provide ways to contact a person by email. Future versions will incorporate functionality so that with a single click user can initiate any of the multiple modes of communication including email, instant messages, and telephone call.

3.6 Deployment details

The tool was deployed as a set of Web pages. Each page contained an ExB applet surrounded by a textual explanation of its use (see, e.g., [14]). The ExB applet was implemented using JDK1.2 and required a Java plug-in to be viewed using Netscape or IE browser.

The initial page of Expertise Browser showed information on the network element software broken down according to functionality into subsystems. It also contained a set of links to other pages. The links list pages showing information for a set of releases (bcfn2.5, bcfn3, bcfn4) and a page containing information on all products (not just this particular network element) developed in the organization. All pages but the initial one were divided into code units representing directory structure. The names of individuals in the expert view were color-coded to indicate their primary geographic location.

4 USAGE OF EXPERTISE BROWSER

At the time of this writing, the Expertise Browser has been deployed in two organizations (with different code bases). ExB was deployed in organization A in February of 2000. This organization initially had about 120 developers at two sites (approximately 40 in England and 80 Germany), and grew to about 250 developers at three sites (England, Germany, and France). The organization is responsible for all software development for one network element of a recently released telecommunications product. The project originated at the German site, where there were relatively few novice developers. The UK site was new when the project started, and has always been considerably smaller than the German site. Finally, the French site had developers who had not been previously familiar with the project, and was the smallest in terms of numbers.

We also include results from another project (we will refer to former project as project A and later project as project B) deployed in October, 2000. In the second project, the two main sites involved were UK and Ireland with Ireland being a much larger site. The project was building operations, administration, and maintenance product for several network elements.

Because of their unfamiliarity, we expected that the developers at French site in project A and at the UK site in project B would need ExB the most. In both cases, this

was because of their newness and geographic distance from established sites.

The deployment was done as a part of a larger project that had the goal of addressing a broad range of issues related to globally distributed software development. In the context of hour-long two-on-one sessions (i.e., two trainers, one developer) where developers received training on several collaboration tools, about 35 members of organization A (about 15 in England, 15 in Germany, and 5 in France) received approximately 5-10 minutes of individual training on ExB. In organization B, about 22 people in Ireland and none in the UK received training. Awareness of ExB spread beyond this initial group by word of mouth, but we do not have a good measure of the extent to which other members of the organizations were aware of ExB.

4.1 Usage logs

We captured ExB usage in logs produced by downloaded applets sending UDP packet back to the web server from which the ExB applet was downloaded. It was then recorded into a log file on the server by a script listening on the specific UDP port. Except for occasional system or network down time, the script ran continuously.

Each packet included the IP address of the client, the time stamp, the type of the interaction, and additional parameters. We recorded when the applet was initialized (page loaded first time), started (page loaded first or subsequent time), stopped (other page has been loaded), destroyed (browser closed or applet removed by garbage collector).

Specific interaction events were generated by users clicking on a particular part of the ExB applet with a mouse. There were the following types of interactions:

- M: select a module or file to show experts associated with it.
- P: select a person to see their contact info and their work profile.
- O: select an organization to see its work profile.
- MF: select a module to see a list of files associated with it.

First, we selected IP addresses of the hosts that were in locations targeted by the ExB deployment. The four countries involved were Germany, United Kingdom, France, Ireland. To do that we had to identify the geographic location of all IP addresses in the logs and eliminated records from other locations. We also eliminated all hosts for which we could not determine the location and eliminated access to all but two projects targeted by deployment. Of the 94 hosts in total, we obtained useful data from 75. The remainder were IP numbers originating at other locations, or were IP numbers whose location we could not determine.

Then we combined access records from each of the 75 remaining host into a single time chain for each host. Each host chain was broken into subsequences starting with ``init" and ending with ``destroy". Because UDP packets are not guaranteed to be delivered, we also broke the chain where intervals between two user actions exceeded 10 minutes. We felt that a single ExB session is not likely to have breaks as long or longer than ten minutes. This resulted in the 143 nontrivial subsequences (we did not consider subsequences that did not have at least a single mouse click). Only 68 hosts had a nontrivial sequence; the most plausible interpretation is that around 10% of the hosts were not able successfully to run the ExB applet. The sequence length histogram in Figure 3 shows that the length of sequences varying substantially, but about half of the nontrivial subsequences had less than 10 user interactions, although there were more than hundred interactions in some subsequences.

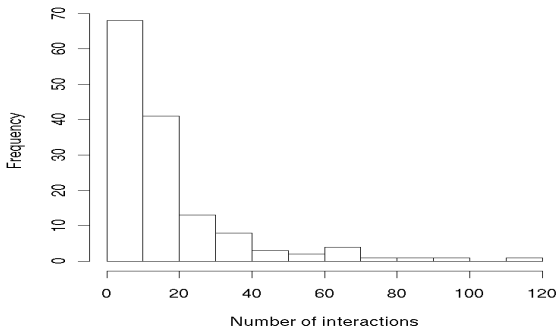

Figure 3. Distribution of session length.

We analyzed the data further broken down by project, because there were substantial differences between the projects.

Table 1. Number of interactions by type and site.

Project	Site	M	MF	O	P	Total
A	Germany	184	2	28	113	327
A	UK	164	12	32	112	320
A	France	226	21	18	141	406
Subtotal		574	35	78	366	1053
B	UK	145	*	4	79	228
B	Ireland	414	*	18	165	597
Subtotal		559	*	22	244	825

The record of usage logs presented in Table 1 shows that there were more interactions at the French site despite much smaller number of people involved over shorter

period of time (French site started participating 8 months later). (The * indicates that that MF type of interaction was not available in project B.)

We can look at the relationships between interaction types and sites by inspecting deviation from the assumption that the site and the interaction type are independent (by looking at the difference between the actual count data and independence model). The most significant deviations involve the French site more frequently inspecting the list of files in a module and the German site almost never doing that. Furthermore, the French site less frequently inspected organizational structure with the UK site being relatively most interested in that aspect. Table 2 shows relative deviations from independence model for projects A and B.

Table 2. Relative deviation from site/type independence model.

Project	Site	M	MF	O	P
A	Germany	.03	-.8	.2	0
A	UK	-.06	.1	.4	0
A	France	.02	.6	-.4	0
B	UK	-.06	*	-.34	.17
B	Ireland	-.02	*	.13	-.07

In project B the two main sites involved were UK and Ireland with Ireland being a much larger site. The tool was demonstrated only at the Irish site. Despite the smaller size and relative lack of exposure, there is a substantial usage of the tool from the UK site. This usage likely reflects the fact that the UK site was not as familiar with the project, and because of its small size, did not have experts in all parts of the product. UK developers seemed primarily interested in finding out where particular people worked in the software. The more established Irish site looked relatively more frequently at the larger picture presented by the organizational level, presumably to understand where different groups were working in the product.

4.2 User feedback

In addition to the quantitative usage data, we collected qualitative user feedback after the training sessions by asking developers if they found the tool useful, how they intend to use it, and if they have any suggestions for improvement. We discovered several things of interest in these sessions.

Everyone indicated that the interface was fairly easy to understand and use, which we think helps to validate our design choices, especially the use of simple list and tree structures for visualizations. Users did, however, make several requests for changes in the user interface (e.g., richer contact information to make it easier to contact the expert, text input capabilities so they could select people or code by name rather than using the graphical interface.

Of potentially more interest were several suggestions we received proposing novel uses for ExB or extensions of ExB. We showed ExB to the manager who had had overall responsibility for one release of the network element. After using ExB to explore the changes for that release, he told us that he had discovered more in the last few minutes about who actually did what on that release than he had ever known while he was actually managing the development. This was a use of the tool we had not really anticipated, i.e., using it to get an overview of project activity. This notion was reinforced by project managers, who were particularly interested in seeing an overview of very recent changes (over the last day or two) done in the project to be more aware of what is going on.

Testers proposed another feature to help them use the tool in a novel way. They desired to see a set of recent changes touching a particular module. When they find that there is a problem with a module (not all tests pass), they first need to make sure that that problem has not been already reported. This involves looking over descriptions of all recent changes and understanding if the change was attempting to fix the problem they are observing. That involved a significant amount of work, and they wanted to use the tool to focus on changes that pertain to the module containing the problem, thereby reducing the amount of search they need to perform. This was another use we had not anticipated.

5 DISCUSSION

The tool described in this paper solves the expertise-finding problem in a way that meets the criteria we initially proposed. I.e., it provides a way of identifying project-related expertise. One can easily find people who worked on particular parts of a project, and can see easily how their experience is distributed over the product, i.e., highly specialized or broad, and exactly where the contributions occurred. Appropriate filtering of expertise atoms allows display of tool, language, or release and other types of expertise. Second, this is accomplished by using data that are collected automatically. It imposes no burden on individuals to describe their expertise. It also provides alternatives to querying project architects and other well-known experts, increasing the chances that the user will find an expert quickly, and reducing the overload on the "expertise experts."

Analysis of usage data strongly suggests that satellite sites that are either new to the project or do not have the sufficient breadth of the expertise are likely to be the most active users. Unexpectedly, however, the main sites not only used it less, but used it differently. The more established sites were also the larger sites. The patterns of use at site A, for example, suggest considerable interest in

exploring where particular organizational units (i.e., development teams) worked in the code. In the newer, smaller, less established site, the tendency was to begin with some part of the product and look for experts. In the older, larger, more established sites, the tendency was to go from people or organizations and find where they worked in the product.

5.1 New applications based on ExB ideas

The techniques exploited for the purpose of locating experts have other compelling applications that we have begun to explore, some of which were suggested by user feedback. One is an "activity awareness" tool that allows a user to browse recent changes that may impact his/her work. Ordinarily, it is difficult to keep abreast of ongoing work that may impact your own, and organizations struggle to make sure that the work of different individuals and groups does not conflict. Providing a way of recognizing potential conflicts very quickly, and avoiding them if possible, would be very beneficial. The awareness version of the tool accomplishes this by choosing related changes either on a static basis (e.g., change the same files the user recently worked in) or a dynamic basis (e.g., changes in code that the user's code calls or is called by). The user can then discover immediately where such changes occurred, when they were made, and who made them. Contact information and capabilities are also provided to support any needed communication.

Another application is a "quantitative resume," an interactive visualization of a person's experience. It includes views that capture tools and technologies used, projects and parts of the product worked on, and roles (e.g., developer, tester, manager) occupied. Figure 4 illustrates one example. The top three views show language, company, and position and the bottom view shows rate (number of changes per year). The resume user has selected the last period in the rate view. The corresponding yellow highlighting is shown in the remaining views indicating that during the selected period, the person had experience with Java and Perl, worked for the company named b2b, and had positions of team leader and developer.

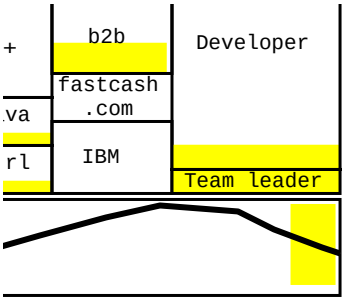

Figure 4: Quantitative resume.

5.2 Refining expertise measures based on EAs.

While in this paper we consider EAs as a very straightforward measure of expertise, it is possible to use EAs measure expertise in more sophisticated ways, using techniques described in [7]. As we mentioned above, there are three goals that are typically of interest in software development: minimize effort, reduce delay, and maximize quality. When estimating someone's expertise, it may be desirable to look at one or more of these qualities, more specifically, how much effort or time it takes to complete a particular task.

We propose to obtain these characteristics for a person or a group of people based on records of past behavior. We concentrate here on estimating the productivity of the subject, because estimating interval and quality is more straightforward. It is worth mentioning that interval and effort are not the same thing in software development. Interval reflects not only the efficiency with which people work, but also the ways in which different tasks are assigned, prioritized, and interleaved. The effort estimation algorithm described below takes into account such overlap of work to obtain accurate estimates of effort.

We consider the measure of productivity as the number of normalized changes (defined below) divided by a unit of effort, for example, ten changes per person month. Because we are interested in expertise pertaining to a particular task, we may want to estimate it as pertaining to a particular part of the system functionality, a particular type of work (fixes vs new development), a particular programming language used or in other ways described above. In principle, if measurements of effort for each change completed by developers were available to us, we could fit a regression model such as

$$E(effort) = size^{\alpha}\beta_{type}\gamma_{developer,\mathrm{exp}erise}$$

in order to obtain estimates of the effects on effort of the following variables:

- type: type of change, which ranges over the values of new and fix;
- size: size of change, which is the number of deltas in a change;
- developer: developer identity;
- expertise: an indicator of whether or not the change pertains to the expertise area of interest.

For a discussion of which variables (besides expertise area) it is important to include in the model, see [7].

Since the direct measurement of change efforts would be impractical, we used an algorithm in [7] to estimate it from the widely available change management data. The estimates are used to define normalized change and to

estimate programmer expertise (in the desired expertise area), as well as, to determine relative impact of various factors on the effort it takes to complete a normalized change. The normalized change is defined by selecting values for all predictors that are not related to a person or an area of expertise. For example, given the model above, we chose a normalized change to be a change done to add new functionality and containing one delta. Then a change to fix a bug containing two deltas would take effort of $\beta_{fix} / \beta_{new} \times 2^{\alpha}$ normalized changes. The productivity dimension of expertise for developer i is expressed by $\gamma_{developer_i, \exp ertise}$. If the ratio $\gamma_{developer_i, \exp ertise} / \gamma_{developer_i, \neg \exp ertise}$ is greater than 1, then that developer is more productive or skilled in that particular area than in other areas she worked on.

5.4 Change management systems and collaboration

Change management systems are designed to support collaborative work on documents and software. Consequently, they contain massive, yet largely untapped sources of many kinds of information about people, activity, and experience. We believe there is enormous potential in this information for tools to support collaborative software engineering. We look forward to the day when change management systems become powerful tools for effective collaboration, as well as control.

6 REFERENCES

[1] Ackerman, M. S. and C. Halverson. Considering an Organization's Memory. in *Computer Supported Collaborative Work*. 1998. Seattle, WA: ACM Press 39-48.

[2] Allen, T. J., Managing the Flow of Technology. 1977, Cambridge, MA: MIT Press.

[3] Carmel, E., Global Software Teams. 1999, Upper Saddle River, NJ: Prentice-Hall.

[4] Cedeqvist, P. et al, *CVS Manual*. May be fond on: http://www.cvshome.org/CVS/.

[5] DeMarco T., and Lister, T. Peopleware: productive projects and teams. 1987, New York: Dorset House Publishing.

[6] Grinter, R. E., J. D. Herbsleb, and D. E. Perry. The Geography of Coordination: Dealing with Distance in R&D Work. in *GROUP '99*. 1999. Phoenix, AZ

[7] Graves, T. and Mockus, A. Identifying productivity drivers by modeling work units using partial data. *Technometrics*, 43(2):168–179, May 2001.

[8] Herbsleb, J. D. and R. E. Grinter. Splitting the Organization and Integrating the Code: Conway's Law Revisited. in *21st International Conference on Software Engineering (ICSE 99)*. 1999. Los Angeles, CA: ACM Press 85-95.

[9] Herbsleb, J.D., *et al.* An Empirical Study of Global Software Development: Distance and Speed. In proceedings, *International Conference on Software Engineering* (ICSE 2001) Toronto, Canada, May 15-18, pp. 81-90

[10] Herbsleb, J. D. and R. E. Grinter. Architetures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software, September/October 1999, pp. 63-70.

[11] Herbsleb, J.D. & Moitra, D. Global Software Development. (2001). IEEE *Software*, March/April 2001, pp. 16-20.

[12] Herbsleb, J. D., *et al.* Object-oriented analysis and design in software project teams. *Human-Computer Interaction* 10, , 1995, 249-292.

[13] Herbsleb, J. D., *et al.* Distance, Dependencies, and Delay in a Global Collaboration. in *CSCW 2000*. 2000. Philadelphia, PA

[14] Hibino, S. L., et al. A web based approach to interactive visualization in context. In *Advanced Visual Interfaces*, pages 181–188, Palermo, Italy, May 23-26 2000.

[15] Kohonen, T. The self organizing map. *IEEE Transactions on Computers*, 78(9):1464–1480, 1990.

[16] Kraut, R. E., C. Egido, and J. Galegher, Patterns of Contact and Communication in Scientific Research Collaboration, in *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, J. Galegher, R.E. Kraut, and C. Egido, Editors. 1990, Lawrence Erlbaum Associates: Hillsdale, NJ. 149-171.

[17] McDonald, D. W. and M. S. Ackerman. Expertise Recommender: A Flexible Recommendation System and Architecture. in *ACM Conference on Computer Supported Cooperative Work*. 2000. Philadelphia, PA: ACM Press 231-240.

[18] Midha, A.K. Software configuration management for the 21st century. *Bell Labs Technical Journal*, 2(1), Winter 1997.

[19] Mockus, A. and Weiss, D. M. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, April–June 2000.

[20] Perry, D. E., N. A. Staudenmayer, and L. G. Votta. People, Organizations, and Process Improvement. *IEEE Software* 11, 4, 1994, 36-45.

[21] Wills, G.W. Linked data views. *Statistical and Computing Graphics Newsletter*, 10(1):20–24, Summer 1999.