Global Software Servicing: Observational Experiences at Microsoft

Shilpa Bugde[1*], Nachiappan Nagappan[2], Sriram Rajamani[3], G. Ramalingam[3]
[1] *Symbiosis Center for Information Technology, Pune 411004, India*
Shilpa.Bugde@gmail.com
[2] *Microsoft Research, Redmond, WA 98052, USA*
[3] *Microsoft Research, Bangalore 560080, India*
{nachin,sriram,grama}@microsoft.com

Abstract

Software servicing in an important software engineering activity that is gaining significant importance in the global software development context. In this paper we report on a study conducted to understand the processes, practices and problems in the Windows servicing organization in Microsoft's India Development Center. We report on our observations and experiences from this study on the main processes and practices adopted for software servicing in Windows and the main problems pertaining to information needs and communication issues. We also discuss our experiences in this study within the context of prior research defined in the global software development community to explain the ways in which Microsoft addresses these common problems.

1. Introduction

Global Software Development (GSD) is a field of research that has grown tremendously over the last decade [4, 8]. Herbsleb and Moitra [8] attribute the acceleration of GSD to the benefits it enables: (i) capitalize on the talent pool and use resources wherever available; (ii) business advantages of new markets; (iii) quick formation of virtual teams to capitalize market needs; (iv) improve time to market by utilizing "around-the-clock" development and (v) flexibility to capitalize on merger and acquisition opportunities globally.

The world's software maintenance expenditure is several hundreds of billions of dollars and is expected to grow substantially in the coming years. According to a market research analysis *the competitiveness of a software company's maintenance and support offering will play a major role in the company's ability to retain customers and increase revenue* [10]. Microsoft too spends significant amount of economic resources for software servicing of its products like Windows, Office, Visual Studio etc. One of the defining characteristics of the software service industry is that huge and complex code bases are maintained and evolved by programmers and testers who were not part of the actual development team. With recent trends in globalization large corporations have opened development sites in geographically distinct locations, and in particular, in countries like India and China for such software servicing tasks. People who work in these locations have difficulty in getting access to in-depth knowledge about the code, and the institutional memory associated with it. Thus, it is interesting to ask: how do such teams currently operate? With the objective of answering this question, we conducted a case study at WinSE, the Windows Servicing group in Microsoft's India Development Center (IDC), Hyderabad. A point to be noted is that substantial development of Windows also takes place in IDC but is unrelated to the direct context of our study and is hence not discussed.

The format of the study is as follows. We employed a three phased approach consisting of interviews, anonymous surveys and shadowing of engineers. First, we interviewed one person from each job profile: Software Design Engineer in Test (SDET) and Software Design Engineer (SDE), as well as managers in these two disciplines. We were able to codify the workflows that these people were involved in for software servicing of the Windows family of systems. Next, we conducted an anonymous survey of all the engineers involved in software servicing at IDC to find out the most time-consuming tasks and sub-tasks done by SDEs and SDETs, the information needs

* Shilpa Bugde was an intern with Microsoft Research, Bangalore when this work was done.

associated with each of these tasks, and how these information needs were met. To complement the survey, we conducted an ethnographic observation, i.e. "shadowing" sessions to observe SDEs and SDETs during their daily work. Finally, we present our experiences with respect to the process, practices and problems associated with GSD in terms of a set of problem dimensions framed by Herblsleb and Moitra[8].

The organization of this paper is as follows. Section 2 provides a survey of related work in the GSD context. Section 3 presents our case study in terms of the interviews, anonymous survey and shadowing experiment. Section 4 presents our observations on how Microsoft alleviates some common problems in GSD.

2. Related Work

Fred Brooks in the classic *Mythical Man-Month* [3] book states that schedule disasters, functional misfits, and system bugs arise in software systems from a lack of communication between different teams. In this section we summarize work related to communication and coordination in the software field from a GSD context. The work closest in spirit to ours is by Herbsleb and Grinter [6] that explores geographically distributed software development in a project based on teams working in Germany and UK at Lucent Technologies. Based on a total of 18 interviews, the prominent coordination factors identified were integration of the system built by the teams; specification of programming interfaces; process mechanisms and documentation. Consequently the primary barriers to team coordination were lack of unplanned contact; knowing the right person to contact about specific issues; cost of initiating the contact; effective communication and lack of trust. Herbsleb and Grinter [6] provide recommendations based on their empirical case study for organizations with respect to communication barriers and coordination mechanisms. From a theoretical perspective, Herbsleb and Mockus [7] formulate and evaluate an empirical theory (of coordination) towards understanding engineering decisions from the viewpoint of coordination within software projects. Open source development adds a new dimension to team coordination and communication as often most open source developers are not joined in a team by any financial binding, nor are they geographically bound to a region. Mockus et al. [11] investigate how different individuals across geographical boundaries contribute towards open source projects (Apache and Mozilla). The prominent observation Mockus et al. [11] made regarding development teams was that in open source development there is a core group of developers who control the code base. Further they also observed that in successful open source systems a group larger by an order of magnitude than the core will repair defects and another group larger by another order of magnitude will report problems. Gutwin et al. [5] observed the requirements and mechanisms for group awareness in three open source system (NetBSD, Apache httpd and subversion). They observed that open source developers maintain a general awareness of the team and knowledge about people they plan to work with.

Figure 1: Team locations in our study (approximate)

The primary means of awareness were mailing lists and chat tools: primarily text based communication mechanisms. For the perspective of experiences with GSD: Battin et al. [2] describe their experiences with GSD at Motorola developing a 3G cellular system with 20% of the required staff in the US and the remaining 80% in Tokyo, Beijing, Singapore, Bangalore, and Adelaide. Bass et al. [1] report on collaboration experiences at Siemens and the problems learned with key learning's in people and communication-related aspects of collaboration. Based on semi structured interviews of 30 engineers in the US, Netherlands and India, Sengupta et al. [12] show the use of tools for distributed requirements management and identify potential areas for research impact in GSD, like reverse-engineering, and maintenance of informal knowledge in a human-independent way.

3. Case Study

Our case study was performed with the software servicing organization at the India Development Center of Microsoft Corporation located at Hyderabad, India. The software servicing organization has to deal with the core engineering team in Redmond, Washington state (shown in Figure 1), USA located more than ten time zones away. The servicing organization has more

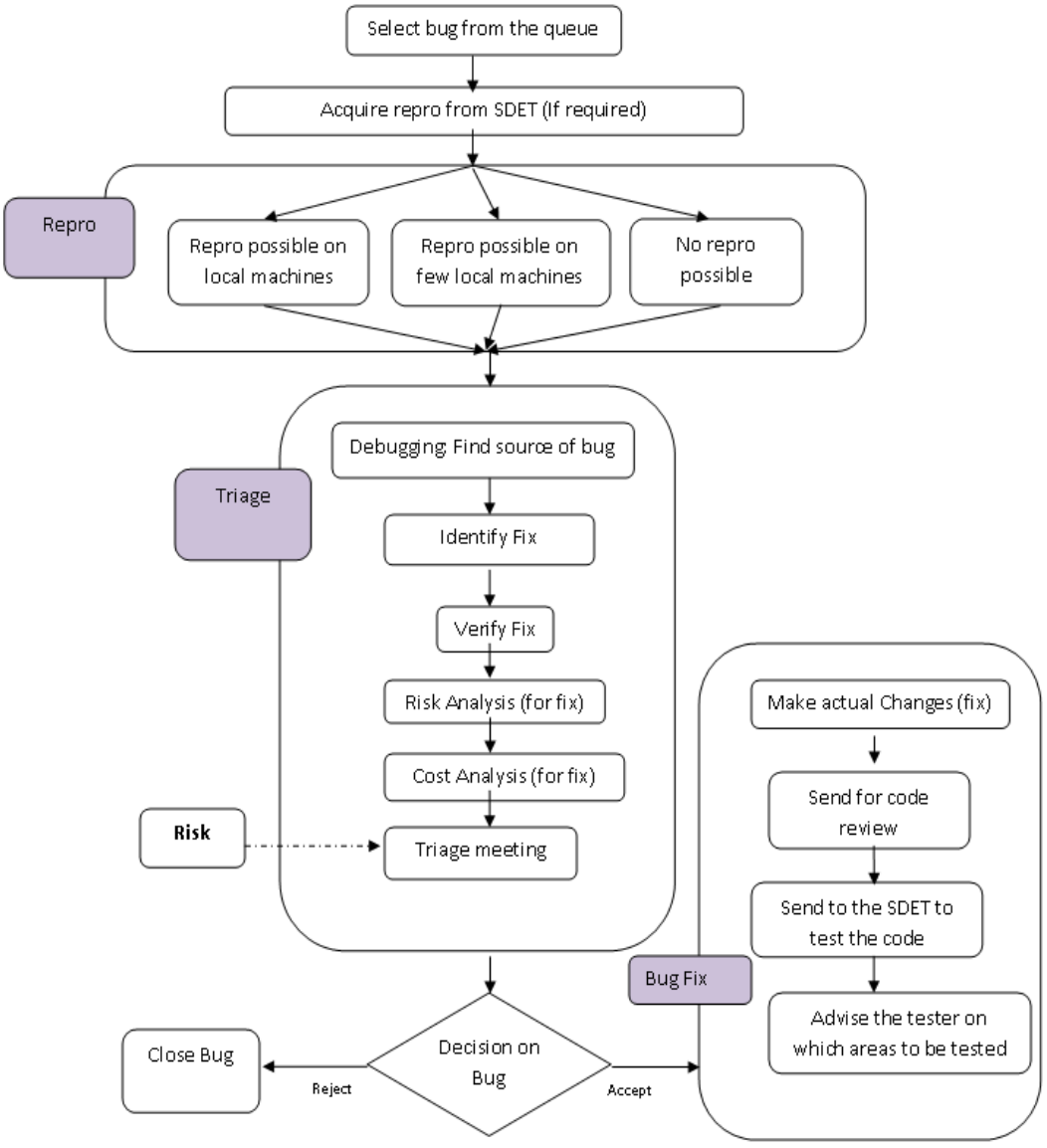


Figure 2: SDE workflow

than one hundred full time employees including experts in the various features of the Windows operating system like the kernel, shell, networking, user interface etc. All responses received towards our study were anonymized to remove any personal information, motivating the engineers to be more open in their discussions with us.

3.1 Interviews

In this section, we present the practices followed by the engineers in IDC for software servicing. We primarily focused on the work done by Software Development Engineers (SDEs), who work on development of bug fixes, and Software Development Engineers in Test (SDETs) who work on testing related tasks such as development of tests; test automation; and ensuring that the fix proposed by the SDE indeed resolves the bug, and does not introduce any other regressions. The workflow of SDEs and SDETs discussed in this section is based on our open-ended interviews with SDEs, SDETs, and leads.

Most, if not all, work done by SDEs and SDETs starts from a bug in the bug queue. The SDE workflow is shown in Figure 2. The SDE starts working on a bug from the queue, governed by the priority and the severity of the bug coupled with the experience of the SDE. The first step is to attempt to reproduce the bug. This phase is called the "repro", and takes about 5-15% of the SDE's time. Subsequent to the repro step, the SDE attempts to find the root cause of the bug, by for example attaching a debugger, setting up check points, flags etc. to find a potential fix. Note that if the repro phase was not successful, the debugging phase is harder and often starts a thread of discussion with the SDET on obtaining a reproducible bug accurately. Once the source of the bug is identified, the SDE evaluates possible fixes.

Then, the SDE identifies the best fix, and evaluates the cost and risk associated with the fix. The cost is a measure of the development and testing effort associated with this fix. The cost is usually determined as: *High, Medium or Low*. The risk is a measure of the probability of the fix resulting in a problem/failure when released. Risk is also expressed as one of *High, Medium or Low*. This phase of identifying the optimal fix assigning risk is called "triage". Triage takes 50-70% of the SDE's time, and within triaging, the dominant task is debugging. The next step is that of "verifying the fix". This phase, involves checking that the fix does resolve the bug. (This does not involve comprehensive testing which is done by the SDETs). As the last stage of the triage process, a decision is taken on the bug in a meeting made up of senior engineers. Here the risk of the bug is evaluated again in the historical context also, i.e. has this area been problem-prone in the past resulting in significant amount of rework, what were the root-causes for this problem, how can this be avoided in the future etc. If it is decided that this bug will not cause any other problems to customers the bug is approved to be fixed in the main code base, and then the fix is made, code reviewed, and tested. Note that, in this phase, "fixing the bug" is primarily committing the fix identified in the triage process to the official code base. The testing is done by the SDETs and will be discussed as in Figure 3.

The SDET workflow shown in Figure 3 also starts from a bug in the queue. The SDET works with PSS (Product Support Services – team that handles support phone calls and creates the bug) and first tries to produce a "repro". Then, the SDE works on root cause analysis, and once the cause and fix are identified, the SDET works on what tests need to be run to check if the fix breaks any already existing tests and contributes to the triage phase. The SDET also identifies new tests to cover the fix. When the bug fix is approved in the meeting of senior engineers, the SDET works to write new tests on a private branch of the code base to ensure that the fix doesn't cause any problems with the existing code base/functionalities. For any anomalies that are observed the error logs are collected and sent to the SDEs who debug and fix the problem again Once tested extensively in the private branch the SDET signs off on the fix and the SDE 'commits' the fix to the main code base.

3.2 Survey analysis

As part of the interviews that we conducted the important themes in both the SDE and SDET workflows were identifying the fix and debugging. We asked both SDEs and SDETs during our interviews as to how they went about collecting information for debugging and codified a set of ten sources of information. These sources are:

a) MSDN library
b) Existing knowledge of system
c) Ask colleagues in servicing – IDC
d) Ask colleagues in servicing – Redmond
e) Ask Windows Product group – IDC
f) Ask Windows Product group – Redmond
g) Search bug database
h) Take help from internal communities
i) Read specification document
j) Look up in the source control system who has worked on this piece of code before to fix bugs

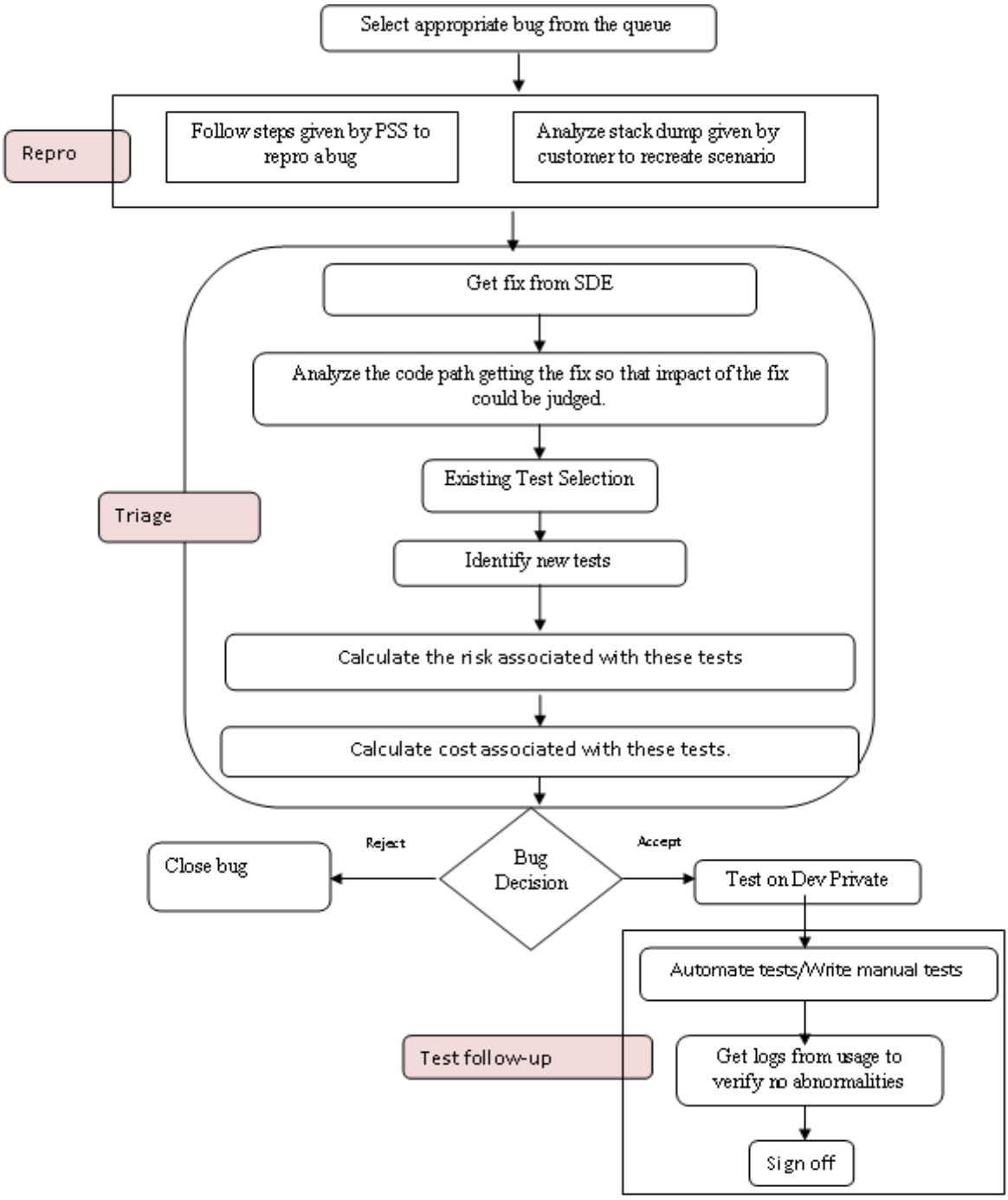


Figure 3: SDET workflow

To assess the extent to which each of these sources of information was used to cater to problems associated with information needs we created a survey where SDEs and SDETs could anonymously select the extent to which they used these practices. Our survey was sent to the entire servicing organization in India (around 100 SDEs and SDETs) and received 32 responses with a response rate of around 30%.

The results of the engineer's responses to the survey are shown in Figure 4 sorted in descending order (by using sum of Almost Always, Frequently and Moderate number of times). From the GSD context we see that discussions with engineers in Redmond for questions were very rare. None of the 32 respondents mention that they "almost always" talk with engineers in the Windows Product group in Redmond. Even very

few talk to colleagues in servicing in Redmond, though more than 80% of the respondents talk to their colleagues in servicing at IDC. This indicates that the SDEs and SDETs in IDC are fairly independent with their own experts in-house upon whom they rely upon for their information needs. From an empirical perspective we evaluate this by carrying out a focused shadowing described in greater detail in section 3.3.

3.3 Shadow Sessions

In this section we report on the details of the "shadowing" sessions where we observed four employees (two SDEs and two SDETs) for two hours each, interrupting them minimally, and taking careful notes on their activities. The aim of the shadow session was to observe as much as possible, and actually experience the work patterns of the SDEs and SDETs, their information needs and how they collect it. The time that they spent doing various tasks was noted and also the details like frequency of interruptions, information sources referred to in case of interruptions and, questions and queries people had during this time were also collected. This data matched considerably with the results obtained from the survey confirming that there was little dependence on the engineers in Redmond thereby making the servicing organization in IDC an independent work entity.

Figure 5 illustrates the work patterns of the people during the shadow session. Each activity is delimited by a line. More explanation can be found by reading the key. This key and coding format of the various activities is motivated by observations done by Ko et al. [9]

A new insight we gained through our shadow sessions is that code reading is a prominent task during debugging. This can be attributed to the fact that people read code most of the time to "debug" or find the root cause of the bug. It is interesting to note that the time people refer to reading bug data is significant. SDEs and SDETs search the bug database to see if a similar issue was fixed earlier and get pointers about what has to be considered while fixing the current issue. This helps the developers see how prior bugs were fixed, read the comments for the previous fixes and documentation to understand the design rationale behind the choice of those fixes. The bug information also provides developers with "people" information regarding who was the person who fixed the bug, who tested the fix so that if they have any questions they have a point of contact to start an investigation and clarify any questions they may have.

Figure 5 also shows that the nature of the work done by the people is heavily interrupt-driven. These interrupts can be external interrupts or internal interrupts. External interrupts mainly arise from colleagues asking questions regarding bug fixes, prior experience in fixing a bug in the same area etc. Also, a person may get internally interrupted when he/she faces similar hurdles. In such cases he interrupts his colleague and becomes a source for external interrupt.

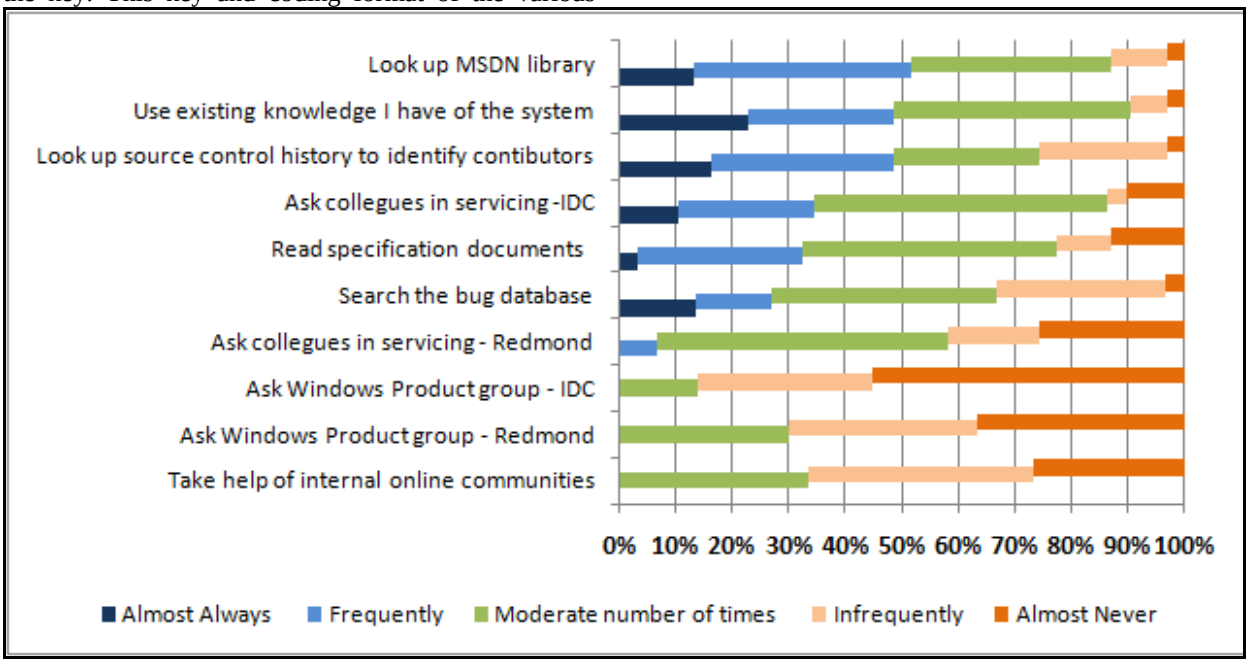
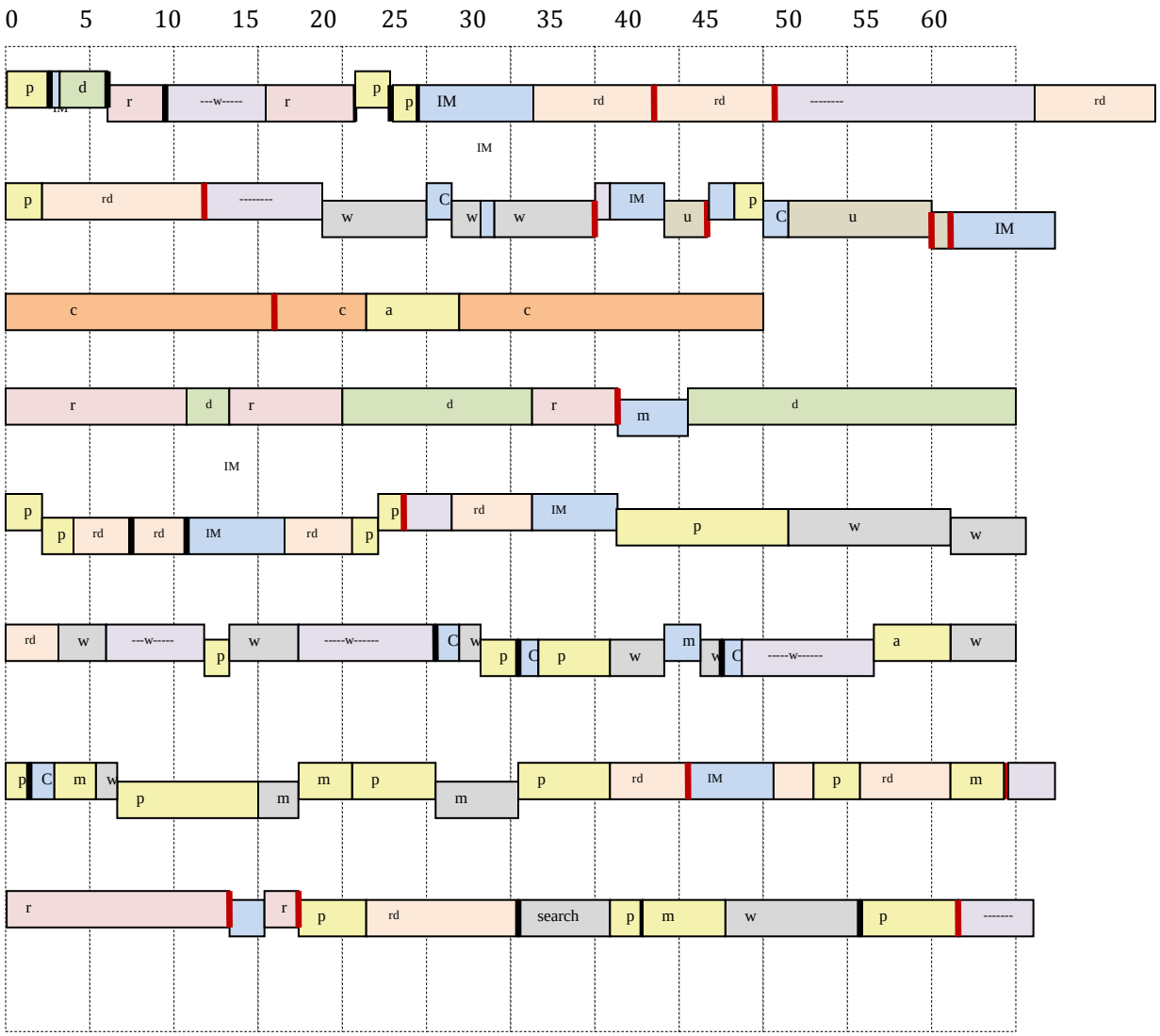


Figure 4: Survey responses for source of satisfying information need at IDC



	Instance of time external interrupt occurs. (People calling, Phones, IMs etc)		Interaction with people for questions (can be via email, IM, phone)
	Instance of time a person gets blocked (Cannot figure out what to do next, Cannot understand what to do next)	w	Activities "related" to work – updating bug database, installing a tool, connecting to a remote machine, writing formal emails.
r	Repro (following repro steps in the, trying out different settings, configurations to find out system behavior to check in what other ways a bug can manifest)	rdr	Reading code (code changes)
d	Debugging (Manual and automated)	u	Unblocking people (not counting responding to emails). Work done to help unblock others
p/m	Issue awareness (bug database and email)	--	No work activity (personal email, IM)
c	Code		Change of bug/issue

Figure 5: Coding of shadowing results

4. GSD:A broader perspective at Microsoft

In this section, we take a step back and examine how Microsoft does software servicing from the perspective of the main problems facing GSD as outlined by Herblsleb and Moitra [8]. Herblsleb and Moitra [8] categorize the main problems in GSD into six main dimensions namely:

- **Strategic issues:** determination of projects that are disjoint architecturally, as much as possible.
- **Cultural issues:** understanding various cultures – norms and practices.
- **Inadequate communication:** difference in time zones and the lack of immediate response to questions.
- **Knowledge management:** sharing product and domain knowledge between teams.
- **Project and process management issues:** synchronization between project and product management deadlines.
- **Technical issues:** bandwidth problems, problems in replicating code bases in different geographical locations.

We now describe how Microsoft addresses the GSD challenges in each of these dimensions.

1. *Strategic issues: determination of projects that are disjoint architecturally, as much as possible.*

In the WinSE team at IDC usually architecturally disjoint components of a software system or the complete system like Windows XP SP3 (Service Pack 3) are owned. Figure 6 shows an example architecture of Windows Server 2003. On the highest level, there are *areas* such as "Multimedia" or "Networking" [13]. Areas are further decomposed into *components* such as "Multimedia: DirectX" (DirectX is a Windows technology that enables higher performance in graphics and sound when users are playing games or watching video on their PC) and *subcomponents* such as "Multimedia: DirectX: Sound" which at the lowest level is comprised of *binaries* (.exe, .dll etc.) [13]. When work is divided between geographical locations it is mostly done according to these architectural separations to ensure that one area/component as a whole is owned in one location (in addition to one location owning the entire system in some cases).

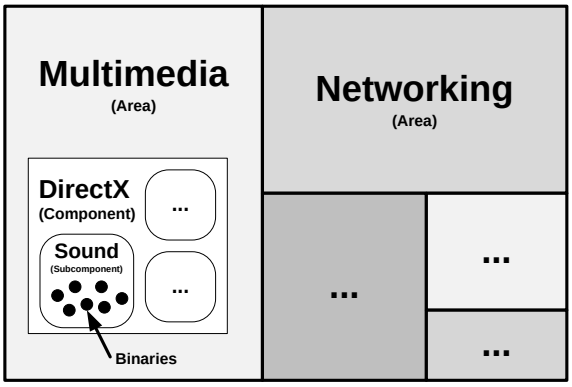


Figure 6: Architectural definition of Windows [13]

2. Cultural issues: understanding various cultures – norms and practices.

To address cultural issues strong face-to-face communication is encouraged. At any given point in time usually there are some engineers from IDC at Redmond and vice-versa. Further, most senior management in Redmond and IDC exchange visits to familiarize themselves with people in the other site. Due to this engagement a personal relationship is built between employees in both centers and this leads to cultural awareness and understanding amongst employees. Further, the fact that people in both locations are full-time employees with similar work environment in terms of employee benefits, compensation etc. means that the cultural issues associated with the outsourcing scenario like short-term engagement, routine low-end maintenance tasks etc, are avoided.

3. Inadequate communication: difference in time zones and the lack of immediate response to questions.

This problem still persists to a large degree as the 10-12.5 hour time difference causes employees at one end to stay up later than their normal working hours. This is alleviated to a large degree by having a round-robin delegation wherein members of a group in both Redmond and IDC stay online to answer any questions for the other team so that one team alone does not have to work irregular hours. Another possible solution that is being used is to have a few employees of IDC in Redmond act as a liason and attend all the Redmond meetings during the normal day to make sure issues pertaining to IDC are highlighted in the meetings at Redmond. This still does not solve the problems regarding email communications that takes place mid-day in either location. Also in addition to email, telephone calls and teleconferencing are strongly encouraged to enable engineers to clear problems

immediately avoiding lengthy email exchanges and develop a personal rapport between employees in different locations.

4. Knowledge management: Sharing product and domain knowledge between teams.

Most of the senior management in IDC have spent a significant amount of time working on Windows and have also worked at Redmond. With such experienced people available, IDC has a mentoring process wherein domain experts are usually available onsite, as determined by our survey results too in section 3.2 and shadowing session in Section 3.3. Further, a joint collaboration between Microsoft Research and WinSE has resulted in the ongoing development of an integrated knowledge base that stores information on source code, fixes, bugs, test cases, ownership information and other development artifacts in a single repository which can be mined to find similar fixes, test cases to be re-run after doing a fix, reading though previous fixes etc. to help engineers identify similar bug fixes and help in the debugging and triage process.

5. Project and process management issues: synchronization between project and product management deadlines.

Project and process management issues are managed by having a single point of ownership for both the Redmond and IDC teams. The overall Director of the servicing organization in IDC and Directors in Redmond are peers who report to the same upper level manager who in turn reports to the executives. Having a common management chain and a single point of responsibility ensures synchronization between project and product management deadlines to ensure the smooth completion of common goals.

6. Technical issues: bandwidth problems, problems in replicating code bases in different geographical locations etc.

Microsoft has invested significantly in developing infrastructure at IDC. There are no bandwidth issues with several dedicated lines to not cause any efficiency bottlenecks. Further all the required data (source code, bug repositories) are run off local servers in IDC so that there is no dependency for any type of data from Redmond.

We have so far discussed our experiences with the process and practices the servicing organization uses in IDC and the ways in which problems associated with information need are addressed by minimizing dependencies, having expert engineers in the servicing organization, access to all version control and bug repositories, exchanging visits between IDC and Redmond etc. Nevertheless there is significant need for new tools in the GSD community that can make room for improvement of the current practices and process employed for GSD. Current research at Microsoft has focused on tools for

- better communication and coordination;
- better search in code;
- finding similar bug fixes;
- developing a recommendation system with machine learning techniques to help in the debugging phase; and
- statistical risk models for bug triage.

These results will be discussed in forthcoming papers. This paper is primarily intended as an introduction to GSD at Microsoft with an example with our experiences. We plan to investigate this line of research further by discussing our experiences, tools and process in future studies and collaborating with researchers and academia outside of Microsoft to build an empirical body of knowledge in this area.

Acknowledgements

We would like to thank the servicing organization in IDC and Redmond for their support of this study. We would also like to thank Jameel Hyder, Jacek Czerwonka, Koushik Rajaram and Alex Tarvo for their help in reviewing earlier drafts of this paper. Shilpa Bugde was an intern with Microsoft Research, India when this research was performed.

References

[1] M. Bass, Herbsleb, J., Lescher, C., "Collaboration in Global Software Projects at Siemens: An Experience Report", Proceedings of International Conference on Global Software Engineering, pp. 33-39, 2007.

[2] R. D. Battin, Crocker, R., Kreidler, J., Subramanian, K., "Leveraging resources in global software development", *IEEE Software*, 18(2), pp. 70-77, 2001.

[3] F. P. Brooks, *The Mythical Man-Month, Anniversary Edition*: Addison-Wesley Publishing Company, 1995.

[4] D. Damian, Moitra, D., "Guest Editors' Introduction: Global Software Development: How Far Have We Come?" *IEEE Software*, 23(5), pp. 17-19, 2006.

[5] C. Gutwin, Penner, R., Schneider, K., "Group awareness in distributed software development",

Proceedings of Conference on Computer supported cooperative work, pp. 72 - 81, 2004.
[6] J. D. Herbsleb, Grinter, R. E., "Splitting the Organization and Integrating the Code: Conway's Law Revisited", Proceedings of International Conference on Software Engineering, pp. 85-95, 1999.
[7] J. D. Herbsleb, Mockus, A., "Formulation and preliminary test of an empirical theory of coordination in software engineering", Proceedings of European Software Engineering Conference/Foundations in Software Engineering, pp. 138-147, 2003.
[8] J. D. Herbsleb, Moitra, D., "Global software development", *IEEE Software*, 18(2), pp. 16-20, 2001.
[9] A. J. Ko, DeLine, R., Venolia, G., "Information Needs in Collocated Software Development Teams", Proceedings of International Conference on Software Engineering, pp. pp. 344-353, 2007.
[10] A. Konary, Boariu, A., Notarfonzo, R., "Worldwide Software Maintenance 2005-2009 Forecast and Analysis: Continued Growth," IDC June 2005. (Source: Business Wire, http://findarticles.com/p/articles/mi_m0EIN/is_2005_Oct_11/ai_n15685393)
[11] A. Mockus, Fielding, R.T., Herbsleb, J., "Two case studies of open source software development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, 11(3), pp. 309 - 346, 2002.
[12] B. Sengupta, Chandra, S., Sinha, V., "A research agenda for distributed software development", Proceedings of International Conference on Software Engineering, Shanghai, China, pp. 731-740, 2006.
[13] T. Zimmermann, Nagappan.N., "Predicting Subsystem Failures using Dependency Graph Complexities ", Proceedings of International Symposium on Software Reliability Engineering, pp. 227-236, 2007.