DOI:10.1145/1866739.1866755

Article development led by acmqueue
queue.acm.org

For sysadmins, solving problems usually involves collaborating with others. How can we make it more effective?

BY EBEN M. HABER, ESER KANDOGAN, AND PAUL P. MAGLIO

Collaboration in System Administration

GEORGE WAS IN trouble. A seemingly simple deployment was taking all morning, and there seemed no end in sight. His manager kept coming in to check on his progress, as the customer was anxious to have the deployment done. He was supposed to be leaving for a goodbye lunch for a departing co-worker, adding to the stress. He had called in all kinds of help, including colleagues, an application architect, technical support, and even one of the system developers. He used email, instant messaging, face-to-face contacts, his phone, and even his office mate's phone to communicate with everyone. And George was no novice. He had been working as a Web-hosting administrator for three years, and he had a bachelor's degree in computer science. But it seemed that all the expertise being brought to bear was simply not enough. Why was George in trouble? We'll find out.

But first, why were we watching George? George is a system administrator, one of the people who work behind the scenes to configure, operate, maintain, and troubleshoot the computer infrastructure that supports much of modern life. Their work is critical—and expensive. The human part of total system cost-of-ownership has been growing for decades, now dominating the costs of hardware or software.[2–4]

To understand why, and to try to learn how administration can be better supported, we have been watching system administrators at work in their natural environments. Over the course of several years, and equipped with camcorders, cameras, tapes, computers, and notebooks, we made 16 visits, each as long as a week, across six different sites. We observed administrators managing databases, Web applications, and system security; as well as storage designers, infrastructure architects, and system operators. Whatever their specific titles were, we refer to them all as system administrators, or sysadmins for short.

At the beginning of our studies, we held a stereotypical view of the sysadmin as that guy (and it was always a guy) in the back room of the university computer center who knew everything and could solve all problems by himself. As we ventured into enterprise data centers, we realized the reality was significantly more complex. To describe our findings fully would take a book (which we are currently writing).[6] In this short article, we limit ourselves to a few episodes that illustrate the kinds of collaboration we saw in system administration work and where the major problems lie. As we'll show from real-world stories we collected and our analyses of work patterns, it's really not just one guy in the back room.

The Story of George

George is a Web administrator in a large IT service delivery center. We observed him over a week as he engaged in various planning, deployment, maintenance, and troubleshooting tasks for different customers.[1] George is part of a team of Web administrators; he interacts with

ILLUSTRATION BY YAREK WASZUL

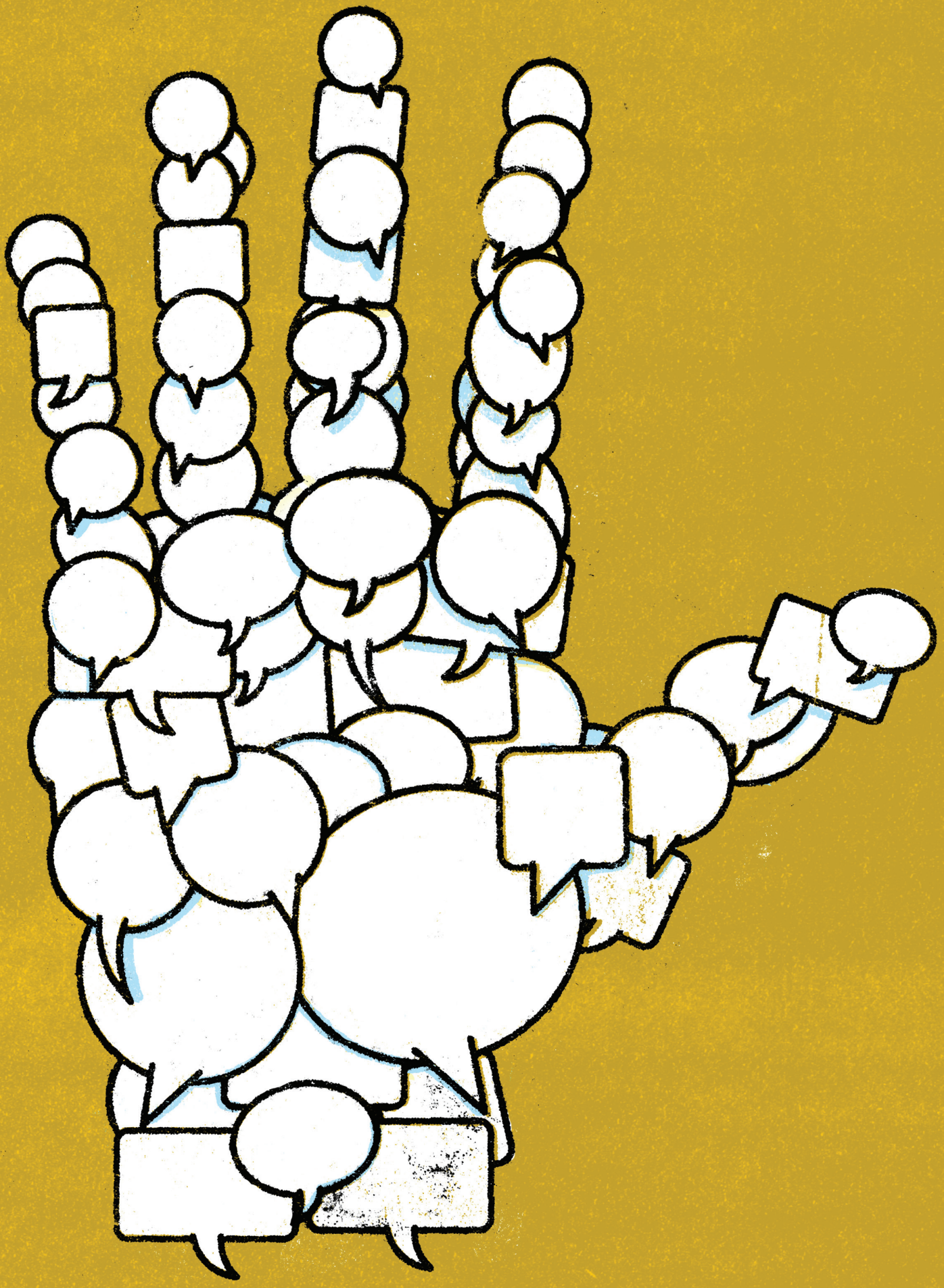
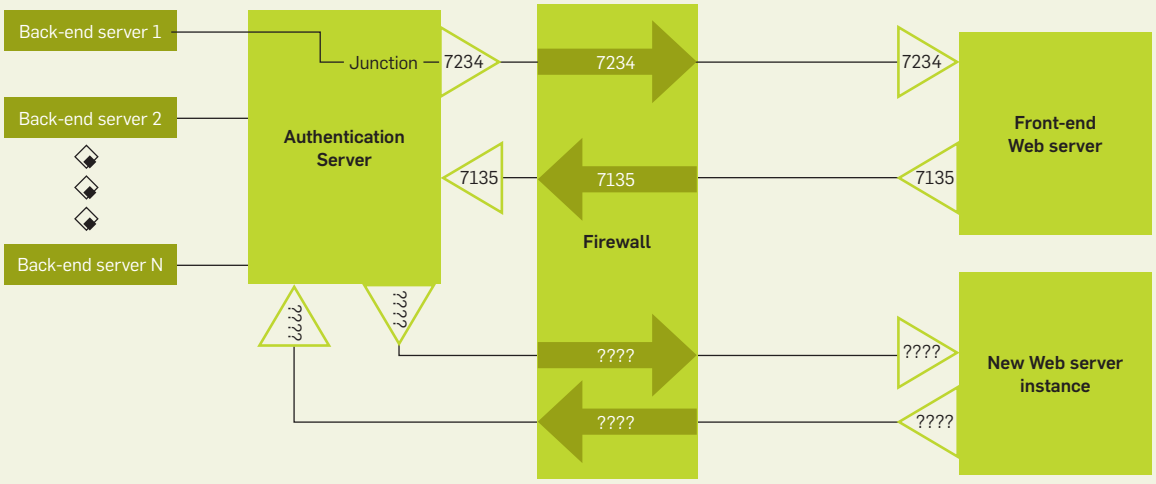


Figure 1. George had to add a new front-end Web server to an existing installation.



the other team members often, as work is distributed. They need to coordinate their actions, hand off long-running tasks, and consult each other (especially during troubleshooting). He also interacts with other teams that are in charge of different areas, such as networks, operating systems, and mail servers.

During our week of observation, one of George's tasks was to set up Web access to email for a customer. This involved creating a new Web-server instance on an existing machine outside the firewall and connecting through a middleware authentication server inside the firewall to a back-end mail server (Figure 1). George had never before installed a second Web server on an existing machine, but he had instructions emailed to him by a colleague as well as access to online documentation. The task involved several people from different teams. Early in the week, George asked the network team to create a new IP address and open ports on the firewall. Throughout the week, we saw him collaborate extensively with Ted, a colleague who was troubleshooting some problems with the authentication server. George's progress was gated by Ted's work, so they exchanged IMs all the time and frequently dropped into each other's offices to work through problems together.

By Friday morning, George had completed all preparations. The final steps should have taken just a few minutes, but this was where the action really began. A mysterious error appeared, and George spent more than two hours troubleshooting the error, mainly in collaboration with others. He had created the new Web-server instance seemingly without incident, and it registered itself with the middleware authentication server. Yet when he issued the command to the middleware server to permit the front-end Web server to talk to the back-end mail server, he got the following message:

```
Error: Could not connect to
server (status: 0x1354a424)
```

Given that three different servers were involved, the error message gave him insufficient information. The online docs and a Web search on the message provided no additional details, so he reached out for help. (For more on error messages, see "Error Messages: What's the Problem?" *ACM Queue*, Nov. 2004.[7])

George's manager suggested calling Adam, the application architect, and George and Adam started troubleshooting together, talking on the phone and exchanging system logs, error messages, configuration files, and sample commands via IMs and email messages (Figure 2). Adam did not have access to the troublesome system, so George acted as his eyes and hands, collecting information and executing commands.

They were not able to find the error, so about an hour in, Adam suggested that George call technical support. He used his office mate's phone (as his own was still connected to Adam), but quickly transitioned communications with tech support to IM. For the next 20 minutes or so, George continued to troubleshoot with Adam on the phone and tech support via IM, and Ted kept popping into the office to offer suggestions. After a while, George became unhappy with the answers from tech support, so Adam hooked him up with one of the developers of the middleware, and they started discussing the problem over IM. Throughout, George remained the sole person with access to the system—all commands and information requests went through him. He became increasingly stressed out as the problem remained unresolved.

Eventually, Ted went back to his own office and looked into the problem independently. He discovered that George had misunderstood one of the front-end server's network configuration parameters, described vaguely in the documentation as "internal port." George thought this parameter (port 7137) specified the port for communication from the front-end to the middleware server, when it went the other way. George, in fact, had made two mistakes: he didn't realize that every front-end server used port 7135 to talk to the middleware server (which was permitted by the firewall, see Figure 1), and he specified a port for communication from the middleware server to the front-end, 7137, that was blocked by the firewall. Communications worked in one direction, but not the other. The software only tested communications

in one direction, so the error was not reported until the middleware authentication server was configured. Ted found a solution to this complex situation, and tried unsuccessfully to explain it to George over IM:

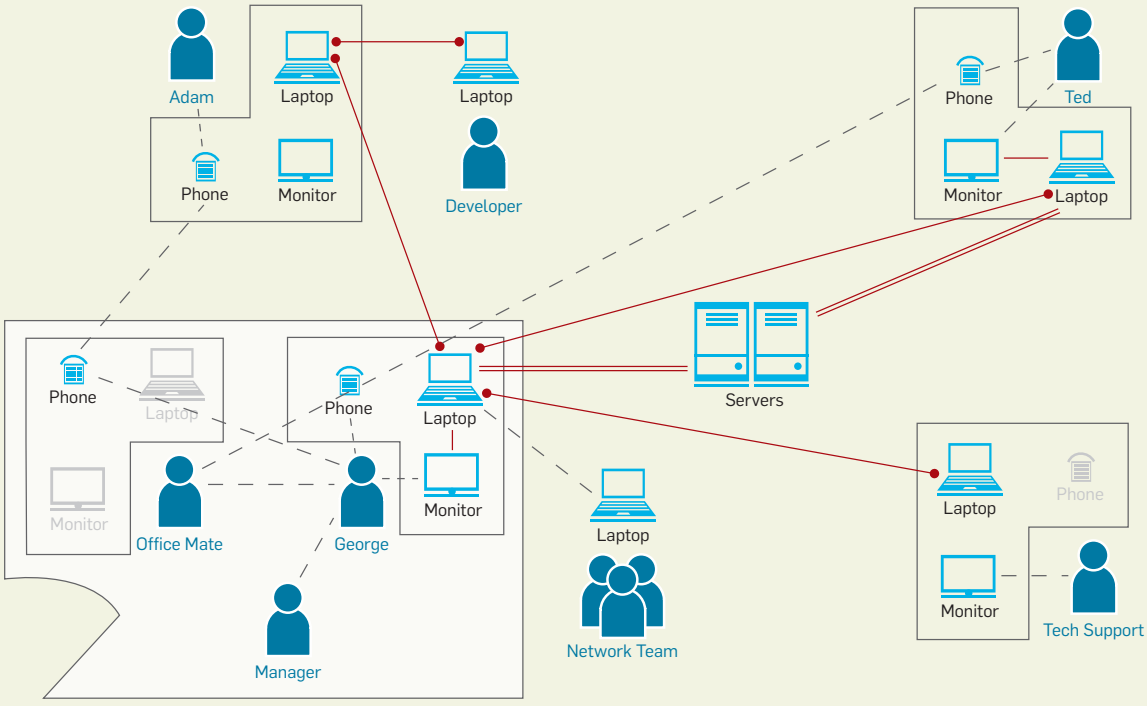
Ted: We were supposed to use 7236. Unconfigure that instance and...
George: Can't specify a return port... you only specify one port.
Ted: You did it wrong.
George: No, I didn't.
Ted: Yes, you did. You need to put in 7236.
George: We just didn't tell it to go both ways. The other port has nothing to do with this.
Ted: Well, all I know is what I see in the conf file.
George: We thought that was the return port. That is not a return port.
Ted: There currently is no listener on [middleware server] on 7137. So use 7236. DO IT!

Ted wasn't getting his point across, and George was getting ever-more frustrated. George told his office mate to call Ted (Adam was still on the other phone), and the conversation immediately switched modes. With the nuance of spoken words, Ted started to realize that George fundamentally misunderstood what was going on. Rather than continually telling George what to do ("DO IT!"), Ted explained why. The task had shifted from debugging the system to debugging George, and they tried to establish a common understanding on which network ports were going which direction.

George: What are you talking about? 7236? We thought that it came in on 7137 and went back on 7236, but we were wrong, that 7236 is like an HTTPS listener port or something?
Ted: It will still come in on 7135 to talk to [middleware] server apparently...
George: Right?
Ted: What's happening is it's actually trying to make a request back, um, through the 72... well, actually trying to make it back through the 7137 to the instance... and it's not happening.
George: I know. I know that. But I can't tell it to...
Ted: Just create it with the 7236. Trust me.
George: Why? That port's not..., that's going the wrong..., that's only one way, too.
Ted: Trust me.
George: It's only one way. Do you understand what I am saying?
Ted: 'Cause it's the [middleware] server talking back to the [Web-server] instance.
George: Yeah, but how does [the Web server] talk to the [middleware] server to make some kind of request?
Ted: 7135 is the standard port it uses in all cases. So we had it wrong. Our assumption on how it works was incorrect.
George: All right, all right.
Ted: If it doesn't work, you can beat me up after.
George: I want to right now. [Laughter on both sides]

How did George get into trouble? Like many failures, there were a number of contributing factors. George misunderstood the meaning of one of the front-end configuration parameters, not realizing that it conflicted with the

Figure 2. George engaged with at least seven different individuals or groups using various means of communication, including instant message (solid lines), email (dashed lines), phone (dotted-and-dashed lines), and face-to-face (dotted lines). Only George and his colleague Ted had direct access to the problematic server (double-solid lines).



firewall rules. The front-end did not test two-way communication, so that errors in the front-end port configuration were not reported until the middleware server was configured. The error message certainly did not help. Perhaps most important was the fact that for most of the troubleshooting session, *George was the only one who had direct access to the system*. All the other participants got their information filtered through George.

Examining the videotapes in detail, we discovered several instances in which George misreported or misunderstood what he saw, filtering the information through his own misunderstanding, and reporting back incorrectly. (One example occurred when George misread the results of a network trace, his misunderstanding filtering out a critical clue.) This prevented Adam and tech support from helping him effectively. The problem was found only when Ted looked at the machine state independently—and then he had to debug George, too. George had many tools for sharing information about system state, but none of them gave the whole picture to the others.

What are the lessons? Collaboration is critical, especially when misunderstandings occur (and from what we saw, incorrect or incomplete understanding of highly complex systems is a common source of problems for sysadmins). Yet collaboration can work only when *correct* information is shared, something that is impeded by misunderstandings and the limitations of communication tools. Proper system design can help avoid misunderstandings in the first place, and improved tools for sharing information could help more quickly rectify misunderstandings when they occur.

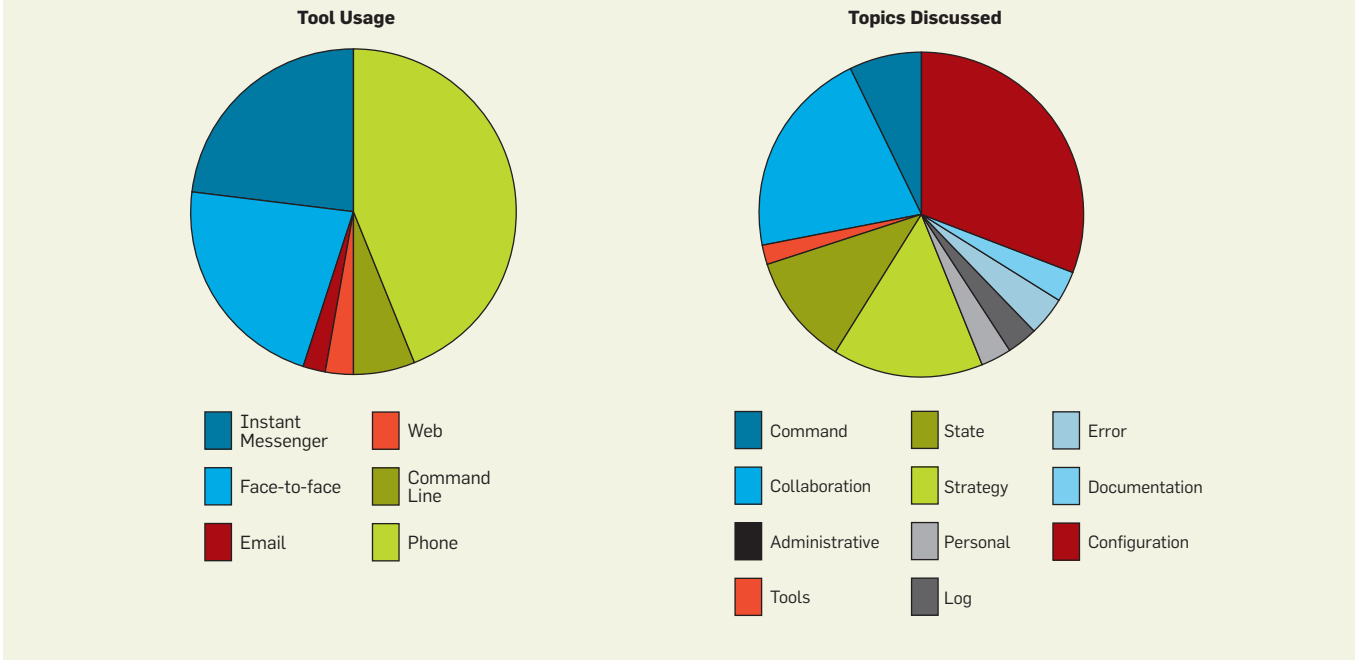
We analyzed the 2.5 hours of George's troubleshooting session, coding each 30-second time slice of what George did (see Figure 3). We found 91% of these time slices were spent in collaboration with other people, either via phone, IMing, email, or face-to-face. Only 6% of the time was he actually interacting with the system, whether to discover state or to make changes, as each interaction was followed by lengthy discussions of the implications of what was seen and what to do next.

While not every troubleshooting episode we witnessed had this extreme level of collaboration, we saw people working together to solve problems much more commonly than a single person toiling alone. We also coded for the topic of collaboration, which included expected topics such as configuration details, system state, ongoing strategy, and what commands to execute. Surprisingly, 21% of the communication involved discussing collaboration itself—for example, "Let me call you" or "Please email me that log file."

Collaboration is especially important in situations where a person's understanding must be debugged, as we saw in George's story. Misunderstandings are a fact of life, and here it was compounded by poorly designed error messages and late reporting of misconfiguration. It can take a long time for someone even to realize that his or her understanding is incorrect. An extra pair of eyes can really help to identify and correct misunderstandings, yet misunderstandings affect what a person reports—so getting a second opinion on the problem will help only if the collaborator gets an accurate picture of the system.

Another lesson is that different communications media are good for different things: the nuance and interchange of the telephone and face-to-face contacts help in getting complex ideas across and in assessing what other people know. IMing is excellent for quickly exchanging commands and error messages verbatim, but subtle personal cues are lost. Even for longtime colleagues like George and Ted, building trust over IM was difficult. Email is great for exchanging lengthy items such as log files and instructions or things that need to persist. Different communications media suggest different levels of commitment to the collaboration.

Figure 3. Accounting of time spent during George's troubleshooting session.



Given the need for collaboration to help sysadmins share their understanding of systems, it is possible to imagine better tools for sharing system state. These tools should take best advantage of different forms of communication to share more completely what is going on with both system and sysadmin alike.

We now turn to another example of collaboration we observed among system administrators working on a much more complex system exhibiting a problem that required incredible effort to understand.

The Crit-Sit

A critical situation, or crit-sit, is a practice that is invoked when an IT system's performance becomes unacceptable and the IT provider must devote specific resources to solving the problem as quickly as possible. Several sysadmins—experts on different components—are brought into a room and told to work together until the problem is fixed. Crit-sits occur more often than sysadmins would like (one we interviewed estimated taking part in four crit-sits per year), and they can last days, weeks, or even months.

We observed one crit-sit for a day, just after it had started, and followed its progress over two months until its solution was found. This was exceptionally long for a crit-sit. It involved an intermittent Web application failure resulting from a subtle interaction of a Web application server and back-end database. Other potential problems were found and fixed along the way, but it took more than 80 days for a dedicated team of experts to determine the true root cause.

At a micro level, being in the room during the crit-sit was fascinating. Eight to 10 people were present in the large conference room, either sitting at the two tables or walking around the room talking; an additional four to six people joined in via conference call and chat room (including technical support representatives for the various software products involved). At first, it seemed amazing to us that this many people had been instructed to work together in a single room until the problem was solved. Indeed, one of the people in the room complained via an instant message to a colleague offsite:

"We're doing lots of PD [problem determination], but nothing that I couldn't have done from home."

Collaboration is especially important in situations where a person's understanding must be debugged, as we saw in George's story. Misunderstandings are a fact of life, and here it was compounded by poorly designed error messages and late reporting of misconfiguration.

After watching the people at work, however, we saw real value in having all of them together in one place. The room was alive with different conversations, usually many at once diverging and rejoining, and with different experts exchanging ideas or asking questions. People would use the whiteboard to diagram theories, and could see and supplement what others were writing. When something important occurred, the attention of everybody in the room was instantly focused. A group chat room was also used as a historical record for system status, error messages, and ideas. Chat was also used for private conversations within the room and beyond, and for exchanging technical information. At one point we saw them build a monitoring script collaboratively through talking, looking at each other's screens, and exchanging code snippets over IM both inside and outside the room.

Not surprisingly, the people in the room appeared much more engaged than the remote participants. Being in the same room signified a level of commitment by the participants. Those on the conference call spoke up only when addressed directly; we assume that they were doing other work and keeping just one ear on the discussions in the room. It is also likely that remote participants could not follow the chaotic, ever-shifting discussions in the room.

At a macro level, following the logs of 11 weeks of troubleshooting was also fascinating. It tells the story of a significant, complicated problem that could not be successfully reproduced on any test system—a problem in which turning on logging would slow the system to the point of unusability at the load levels required to cause the failure. The story shows the crit-sit team interacting with the support teams for a variety of products, escalating to the highest levels, applying patch after patch and experimenting with configuration settings, new hardware, and special versions of the software. The process involved a lot of work by many different teams.

On the whole, the crit-sit was a collaborative effort by a group of experts to understand and repair the behavior of a complex system consisting of many components. They used a wide variety of technical tools: IMing, email, telephone, and screen sharing, yet it seems that

they received the greatest value from interacting face-to-face. By being in the same room, people could quickly shift from conversation to conversation when a critical phrase was heard, with a very low barrier to asking someone a question or suggesting an idea.

Although the crit-sit seems heavyweight and wasteful, we have no other approaches that can replicate the collaborative interaction of a bunch of people stuck in a room searching for a solution to a common problem. It would be a revolutionary advance for system administration if a tool were developed that could permit the same engagement in remote collaborators as we saw in the crit-sit room.

We next describe the sorts of collaborations we observed among security administrators at a U.S. university.

The "ettercap" Incident

When we first met the security administration team for a computer center at a large university,[5] they seemed somewhat paranoid, making such statements as, "I'll never type my password on a Windows box, because I can't really tell if it's secure." After watching them for two weeks, we realized they had good reason to be cautious. IT systems, as a rule, have no volition and don't care how they're configured or whether you apply a patch to them. Security administrators face human antagonists, however, who have been known to get angry when locked out of a system and work extra hard to find new vulnerabilities and do damage to the data of those who locked them out.

The work of these security administrators was centered around monitoring. New attacks came every week or two. Viruses, worms, and malicious intrusions could happen anytime. They had a battery of automatic monitoring software looking for traces of attacks in system logs and network traffic. Automated intrusion-detection systems needed to err on the side of caution, with the sysadmins making the final decision as to whether suspicious activity was really an attack. These sysadmins relied on communications tools to share information and to help them maintain awareness of what was going on in their center, across their campus, and around the world.

The security administration team shared adjacent offices, so back-and-forth chatter about system activity was common. They joked about taking down the wall to make one big workspace. They also used a universitywide MOO (multiuser domain, object oriented), a textual virtual environment where all the system administrators would hang out, with different "rooms" for different topics. The start of an incident would result in high levels of activity in the security room of the MOO, as security admins from different parts of campus would compare what was happening on their own systems. On a day-to-day basis, the MOO might hold conversations on the latest exploits discovered or theories as to how a virus might be getting into the network. The admins described the MOO's persistence features as really helpful in allowing them to catch up on everything that was going on when they came back after being away, even for a day. They also used a "whisper" feature of the MOO for point-to-point communication (like traditional IM).

One of our motivations for studying sysadmins is the ever-increasing cost of IT management. Part of this can certainly be attributed to the fact that computers get faster and cheaper every year, and people do not.

An example of MOO use for quick interchange of security status came when we observed a meeting that focused on hacker tools. The security administrators discussed a package called "ettercap." Being unfamiliar with this tool, one of us began searching the Web for information about it using the wireless network. A few minutes later, one of the administrators in the room informed us that a security administrator working remotely had detected this traffic and asked about it on the MOO:

Remote: Any idea who was looking for ettercap? The DHCP logs say [observer's machine name] is a NetBIOS name. Nothing in email logs (like POP from that IP address).
Remote: Seemed more like research.
Remote: The SMTP port is open on that host, but it doesn't respond as SMTP. That could be a hacker defender port.
Local: We were showing how [hacker] downloaded ettercap. One of the visitors started searching for it.
Remote: Ah, OK. Thanks.

In the space of only a few minutes, the sysadmin had detected Web searches for the dangerous ettercap package, identified the name of the machine in question, checked the logs for other activity by that machine, and probed the ports on the machine. He could see

that it was probably someone doing research, but checked the MOO to verify that it was in fact legitimate.

The participants also collaborated on a broader scale. During our visit, the site was dealing with a worldwide security incident targeting military, educational, and government sites across the U.S. and Europe. This was a particularly persistent attack—every time an intrusion was detected and a vulnerability was closed, the attackers would come back using a new exploit. The attackers would hop from institution to institution, compromising a machine in one place, collecting passwords, and then trying those passwords on machines at other institutions (as users often have a single password for accounts at different sites).

This broad-based attack required a broad-based response, so security administrators from affected institutions formed an ad hoc community to monitor and share information about the attacks, with the goal of tracing the attacks back to their source. When a compromised machine was found, they would let it remain compromised so that they could then trace the attackers and see where else they were connecting. This collaboration was like information warfare: it was important to share information about known compromised machines and exploits with trusted colleagues, but the information had to be kept from the attackers. You did not want the attackers to know that you had detected their attack and were monitoring their activities. When we first observed them, the security administrators used conference calls for community meetings. Later they found a special encrypted email listserv to keep their information under wraps—but because this tool was unmaintained, they had to adopt and maintain it themselves.

The world of security administration seems very fluid, with new vulnerabilities and exploits discovered every day. Though secrecy was a greater concern than with other sysadmins we observed, collaboration was the foundation of their work: sharing knowledge of unfolding events and system status, especially when an attack might be starting and time was critical.

Conclusion

One of our motivations for studying sysadmins is the ever-increasing cost of IT management. Part of this can certainly be attributed to the fact that computers get faster and cheaper every year, and people do not. Yet complexity is also a huge issue—a Web site today is built upon a dramatically more complicated infrastructure than one 15 years ago. With complexity comes specialization in IT management. With around-the-clock operations needed for today's enterprises, coordination is also a must. System administrators need to share knowledge, coordinate their work, communicate system status, develop a common understanding, find and share expertise, and build trust and develop relationships. System administration is inherently collaborative.

At first, it is easy to think that George's story shows poor debugging practices or worse, poor skills, but we don't think that's the case. The system was complex, the documentation poor, the error messages unenlightening, and no single person was responsible for all of it. Better error messages or better documentation would certainly help, but that misses the point. There will always be cases that go uncovered and complexities that are hidden until it is too late. Modern IT systems are so complex that people will often have an incorrect or incomplete understanding of their operation. That's the nature of IT. The crit-sit story and the security story also show it. The one constant in these cases—and in almost all the cases we observed—was collaboration.

We observed collaboration at many levels: within a small team, within an organization, and across organizations. We observed several different types of collaboration tools in use. We observed people switching from one tool to the other as needs shifted. We also observed simultaneous use of several collaboration tools for different purposes. Not surprisingly, system administrators use the same collaboration tools as the rest of us, but these are not optimized for sysadmin needs—whether it is team brainstorming and debugging or secure information sharing.

Though specific features can be implemented for system administrators, it is clear to us that because of the diverse needs among system administrators, a single collaboration tool will not work for all. There needs to be a variety of tools, and collaboration needs to be a first-class citizen in the work of system administration itself. Better collaboration support could relieve the burden on individuals of communicating and establishing shared context, and so avoiding missed information and enabling a persistent store for communication. We believe that improved tools for system administrator collaboration have great potential to significantly impact system administration work—perhaps even helping to restrain the ever-growing human portion of IT's total cost of ownership. C

Q Related articles on queue.acm.org

Error Messages: What's the Problem?
Paul P. Maglio, Eser Kandogan
http://queue.acm.org/detail.cfm?id=1036499

Oops! Coping with Human Error in IT Systems
Aaron B. Brown
http://queue.acm.org/detail.cfm?id=1036497

Building Collaboration into IDEs
Li-Te Cheng, Cleidson R.B. de Souza, Susanne Hupfer, John Patterson, Steven Ros
http://queue.acm.org/detail.cfm?id=966803

References

1. Barrett, R., Kandogan, E., Maglio, P.P., Haber, E.M., Prabaker, M., Takayama, L.A. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the Conference on Computer-Supported Collaborative Work*. 2004.
2. Gartner Group/Dataquest. *Server Storage and RAID Worldwide* (May 1999).
3. Gelb, J.P. System-managed storage. *IBM Systems Journal 28*, 1 (1989), 77–103.
4. ITCentrix. *Storage on Tap: Understanding the Business Value of Storage Service Providers* (Mar. 2001).
5. Kandogan, E., Haber, E. M. 2005. Security and Usability: Designing Secure Systems that People Can Use. In *Security Administration Tools and Practices*. L.F. Cranor and S. Garfinkel, Eds. O'Reilly Media, Sebastapol, 2005, 357–378.
6. Kandogan, E., Maglio, P.P., Haber, E.M., Bailey, J. (forthcoming). *Information Technology Management: Studies in Large-Scale System Administration*. Oxford University Press.
7. Maglio, P.P., Kandogan, E. 2004. Error messages: What's the problem? *ACM Queue 2*, 8 (2004), 50–55.

Eben M. Haber is a research staff member at IBM Research, Almaden, in San Jose, CA. He studies human-computer interaction, working on projects including data mining and visualization, ethnographic studies of IT system administration, and end-user programming tools.

Eser Kandogan is a research staff member at IBM Research, Almaden, San Jose, CA. His interests include human interaction with complex systems, ethnographic studies of system administrators, information visualization, and end-user programming.

Paul P. Maglio is a research scientist and manager at IBM Research, Almaden, San Jose, CA. He is working on a system to compose loosely coupled heterogeneous models and simulations to inform health and health policy decisions. Since joining IBM Research, he has worked on programmable Web intermediaries, attentive user interfaces, multimodal human-computer interaction, human aspects of autonomic computing, and service science.