Usable Autonomic Computing Systems: the Administrator's Perspective

Rob Barrett Paul P. Maglio Eser Kandogan
IBM Almaden Research Center
650 Harry Rd.
San Jose, CA 95120 USA
{barrett, pmaglio}@almaden.ibm.com, eser@us.ibm.com

John Bailey
IBM WebSphere
4205 S. Miami Blvd.
RTP, NC 27709 USA
baileyj@us.ibm.com

Abstract

One of the primary motivations behind autonomic computing (AC) is the problem of administrating highly complex systems. AC seeks to solve this problem through increased automation, relieving system administrators of many burdensome activities. However, the AC strategy of managing complexity through automation runs the risk of making management harder. We performed field studies of current administrator work practices to inform the design of AC in order to ensure that it simplifies system management. In this paper, we analyze what system administrators do in terms of three important activities: rehearsal and planning, maintaining situation awareness, and managing multitasking, interruptions and diversions. We provide guidelines for constructing AC environments that support these activities.

1. Introduction

Autonomic computing (AC) will fundamentally transform interaction between system administrators and computer systems. In particular, AC seeks to shift the specification of system behavior from low-level configuration settings to high-level business-oriented policies [7]. Such supervisory control will allow systems to be much more dynamic (changing more rapidly) and of much larger scope (administrator controls affecting more systems and more diverse systems) than today's systems. As a result, administrator controls will be both more powerful and more dangerous than existing controls [7]. It is well-known that poorly designed interfaces to automation can have disastrous results [12], and so it is critical that administrators have effective tools for managing this increased power (see also [11]).

Because AC is based on improving the administrator experience, we sought to understand current work practices and problems (see also [3]). To this end, we conducted a series of ethnographic field studies of database and web system administrators at large industrial computer service delivery centers in the US to observe and begin to understand work practices [8]. Two researchers participated in each visit, which lasted three to five days. Typically, we followed the work activities of one administrator per day. One researcher took notes and occasionally asked questions, while the other videotaped human-computer interactions and other activities. In all, nearly 200 hours of videotape were collected, reviewed, and analyzed to varying degrees.

In this paper, we detail three aspects of administrator experience culled from our observational data: (1) rehearsal and planning, (2) situation awareness, and (3) multitasking, interruptions and diversions. First, we describe in general terms the tools and practices of system administrators. Next, we analyze our observations according to these three aspects, detailing ways in which AC should be careful to enhance rather than hinder the work of administrators. Finally, we present a brief case study of how a specific application server is beginning to incorporate AC technology to improve manageability.

2. Administrators and their Tools

Computer system administrators face many kinds of technical problems when installing, configuring, deploying, and updating computer systems. AC promises to improve their work through automation. The current toolset for the system administrator includes command-line based consoles (often referred to as command-line interfaces or CLIs), standalone graphical applications (graphical user interfaces or GUIs), and web-based management tools. Sometimes

administrators must use several tools together, as one administrator explained to us, *"I can't think of one GUI or CLI tool that could do 10% of what I do today"*. Another was even more explicit:

> *It is rarely the case that syadmins do most of their work through a unified tool. At best, they use a handful of various tools for specific functionality. Especially in UNIX, there is rarely an admin UI that can address the various tasks a sysadmin need perform.*

Command-line based consoles are favored by system administrators for a variety of reasons. First, administrator multitasking is nicely supported by command-line interaction, which can be used to run tasks simultaneously on multiple open terminal sessions and supports quick switching between them. The history feature of a terminal session is of further help for administrators, reminding them of context as they can see previous commands and previous output.

A second reason administrators like to use command-line consoles is that they tend to support fast and reliable probing of disparate parts of a system and let them drill down on details. During problem solving (an almost continual aspect of administrator activity), being able to move about a system in a dynamic fashion is helpful. However, command-lines fail to provide dynamic peripheral information while administrators work on individual tasks.

Command-line tools also support scripting better than graphical tools, which is especially important for large installations, as one administrator noted: *"GUIs also don't allow you to automate, which seems problematic – as if the vendor expects one sysadmin to manage a handful of machines each (the windows admin model) ☹."*

By contrast, GUI tools tend to support novice users in complex system administration. Through a variety of wizards and advisors, such as the DB2 Performance Advisor, system administrators can engage in a structured conversation with the system by following a series of dialogues and answering questions related to difficult problems or tasks. As one administrator put it, "*CLI is faster for familiar tasks. GUI is faster for rarely performed tasks.*"

One concern among system administrators is the correctness of what tools or interfaces display, which is necessary for establishing trust. One of our administrators described his concern this way: *"I prefer the CLI. These tools seem to be the most truthful and accurate for administration. GUI's seem to be buggy, and do not update state as often*".

Another problem with GUIs results from the lack of systems support for graphics. One of our administrators told us,

> *In unix production land, servers are deployed without graphical cards or X window system. Additionally, firewalls are often configured to block forwarding of X system calls. Essentially, our only interface to applications is through the CLI which is attained either by direct console access, console server access, or secure shell.*

More and more vendors are providing web-based system management tools. These tools do not depend on graphics support on the managed systems, yet can display information graphically. Another advantage of web-based tools is that they can be easily integrated to provide an organized suite of web pages that can be used for daily tasks. Nevertheless, such interfaces may not provide sufficient support for multitasking, suffering from the same problems as GUIs.

Finally, administrators use their own scripts to automate monitoring of system health, to perform operations on a large number of systems, and to try to eliminate errors on common tasks that take many steps. One of our administrators explained it this way: *"Many administrators build their own scripts to manage the daily routines, but often these could be included in a GUI."* The very existence of administrator-authored scripts might seem to be evidence that supplied tools are inadequate in supporting the work of system administrators. Customization and automation are a normal part of the work and professional tool designers simply cannot foresee all possible tasks, needs, and requirements. Many tools fail to support activities that result from the scale, complexity, and risk of operations. System administrators often apply long-running operations to very large numbers of objects, making automation and scripting crucial. Most GUIs we observed fail to support this. CLIs offer more power, but with less ease of use. At one site, we observed a database administrator who had developed a set of monitoring scripts that periodically gathered data from a large number of databases, created web pages with status reports, and triggered alarms when certain criteria were met. A web administrator at another site had configured a similar system: a program that regularly checked various servers, sending e-mail or pager messages in case of errors. In essence, these administrators created their own tools, determining what information they needed and when and how to present it. However, most system administrators we

observed did not have the skills or time to build most of the tools they would need.

3. Analysis and Guidelines for AC

We have observed that system administrators face daunting challenges when managing large and complex computer installations. AC addresses this problem through high levels of computer automation. However, there are enough examples of automation actually making complex systems *less* reliable that some thought must be put into creating the kind of automation that will be most helpful [12]. To structure our analysis, we consider system administrator work practices according to three particular challenges that administrators face. Clearly they face many other challenges, but these three are both significant across many kinds of administrator work, and could become more difficult with the introduction of autonomic computing technologies rather than easier. Because AC is about making system administrator easier, designers of AC technologies should actively seek to ease: (1) rehearsing and planning for changes that are to be made on critical production systems, (2) maintaining situation awareness over systems and environments that are too complex to fully understand, and (3) working on multiple lengthy tasks while being diverted by unexpected interruptions. We examine each in turn.

3.1. Rehearsing and Planning

Administrators often work with production systems that should never go down except during narrow time windows of scheduled maintenance. Even brief system failures are often intolerable, and loss of data is never acceptable. Therefore, we found that most actions are carefully planned and rehearsed before they are performed on production systems. The amount of time devoted to this preparatory work should not be underestimated, as several weeks of preparation may go into the execution of a handful of commands during a maintenance window.

3.1.1 Rehearsing Database Operations. Database administrators we observed had the most extensive planning and rehearsal procedures, but we observed web administrators also doing considerable planning before system changes. In the case of database systems, three levels of servers were typical. In one case, we observed four: *sandbox* systems that allowed experimentation without any limitation but that had no data; *test* systems that had sample data and applications; and *staging* servers that were exact replicas of *production* servers. Changes were most often promoted from staging to production by automated processes, with very few people having access to production servers. Updates typically worked their way through rehearsal on each system before being done on the production server during a time window designated for system maintenance. Rehearsals not only gave administrators opportunities to demonstrate correctness of operations, but also practice at solving problems and timing steps to could accomplish tasks during allotted time windows.

Nevertheless, planning and rehearsing sometimes leads to problems. Consider the case of Christine, a database administrator we observed perform a database operation that she had never done before. The task included moving a number of database tables to a new file system on the production server to manage disk space. Her colleague, Mike, helped through the task using notes and executable scripts created the last time he had done it. As the task involved production servers and a limited maintenance window, they rehearsed operations on test servers first. Mike sat with Christine during rehearsal and verified each operation she typed. The instructions included specific commands to run as well as notes such as "Check that the tables were created properly." As commands and scripts were tested on each system, they were manually edited in a text editor to modify server names.

In the final rehearsal, errors appeared during the execution of one script because a semicolon was deleted accidentally when the script was edited. The script was aborted by hand, but several commands in it had run nonetheless. Mike and Christine thought the script had created an incorrect database table though in fact it had not. When they tried to delete the nonexistent table, they received error messages that suggested they had made syntax errors in the table-delete command. They looked up documentation and manually executed many different commands to try to delete the table. It took them a long time to realize that the table had not been created.

3.1.2 Error and Misunderstanding. Rehearsing and planning of changes to critical systems are necessary because of both the chance for human error in effecting the change and the danger of unforeseen consequences resulting from even a perfectly executed change. Autonomic systems may increase both of these dangers. First, human error in working with conventional computer systems is limited to such mundane mistakes as mistyped commands, omitted steps, misreading system responses, and so on. But with autonomic systems that are driven by high-level

policies, there is the additional problem of misunderstanding between human and computer.

Second, even if changes are executed flawlessly, autonomic systems will have a greater risk of unintended consequences resulting from changes because of the greater scope of autonomic systems. As the scale and degree of coupling within complex systems increases, new patterns of failure may develop through a series of several smaller failures [16]. Conventional systems are controlled largely in a component-by-component manner, with most problems occurring at interfaces between components. Nevertheless, system-wide problems do occur and are some of the more difficult ones to solve. For example, improving end-to-end performance in a complex system can be difficult simply because so many components are involved that it may take a long time to find the slow culprit. In autonomic systems, it will be commonplace for autonomic managers to control a wide variety of components based on the policies that administrators specify. As these autonomic managers automatically reconfigure subsystems, the consequences on the overall system may be difficult to predict.

3.1.3 Guidelines. Rehearsing and planning will be even more critical for ensuring proper operation of autonomic systems than of conventional ones. Autonomic systems must provide facilities that make rehearsing and planning easy. There are several ways to do this. First, it should be easy to build test systems with various degrees of fidelity to production systems, and to verify that such systems remain configured consistently with the production systems they simulate. Because no simulation is perfect—especially for large complex systems—there will probably be a need for many test systems that simulate different aspects of the full system.

Second, systems should be designed to allow administrators to quickly *undo* changes, making operations (whether on production systems or test systems) less risky and therefore easier [3]. In the case of Christine's missing semicolon, it took over an hour to bring the system back to its starting point so the corrected script could be run. Unfortunately, providing undo capabilities can be a technical challenge, such as when administrator commands lead to reorganization of large amounts of data. When undo capabilities will not be available after a particular command, administrators need to be aware of this.

Third, rehearsals are only useful if the results of rehearsed operations can be tested. Autonomic systems will need to have enhanced capabilities for testing complex end-to-end systems so that administrators will be confident that their changes are not having unintended consequences. Because autonomic systems (like conventional systems) will be deployed for accomplishing tasks that component designers cannot imagine, testing will best be enabled by providing administrators with facilities for developing their own tests, as well as for running common system tests. One possibility is to introduce automation changes gracefully as in Sheridan's levels of automation [9]. Here, changes to automation would first come in the form of recommendations. As the administrators become comfortable with applying these recommendations, changes can be put into automation where execution is carried out fully automatically.

Rehearsing and planning will grow in importance for AC. Autonomic systems must facilitate this activity so that administrators will have confidence that production systems will perform correctly.

3.2. Situation Awareness

Having good situation awareness is vital for making decisions and quickly reacting to changing environmental and system conditions [4]. Simply put, having situation awareness means knowing how to answer three questions [15]: What is it doing? Why is it doing that? What will it do next? Much is known about how to provide appropriate awareness for automated control systems, but less is known about providing situation awareness for computer systems more generally [1].

3.2.1 What's going on? It is easy to find examples of poor situation awareness leading to problems in computer system administration. For instance, Oppenheimer [9] recounts how an operator reformatted a database backup disk assuming there was a secondary backup. In fact the secondary had failed long ago and was never repaired. As fate would have it, the main database crashed at the same time the backup was being reformatted, leading to significant data loss.

Administrators deal with dynamic and complex processes at many different levels of abstraction. They need to be aware of systems that are not only complex, but that also change frequently. Furthermore, administrators must share situation awareness across shifts and areas of responsibility. In one troubleshooting case, an administrator we observed discovered that a change in a product made at the customer site had caused a problem—but because the administrator was unaware of what the customer had done, it took a long time to find the cause. Yet for

administrators, having incomplete mental models of the systems they manage is normal. As one put it, "*If understanding the (whole) system is a prerequisite for operating the system, we are lost.*"

We saw many problems caused by faulty situation awareness in our observations. For example, in one case, communications were blocked between one server and another because of a misconfigured firewall. In this instance, situation awareness depended on understanding the interaction between several components in an overall system. Each system had its own management interface and so gaining overall situation awareness was very difficult. The administrators managed this complexity by rapidly moving among multiple tools and working together with many experts in the absence of a single view of the entire system [8]. A simple drawing of the entire configuration would have made the situation clear and avoided hours of troubleshooting.

3.2.2 Guidelines. Automation has a history of negatively affecting situation awareness by reducing operator vigilance, encouraging operator passivity, and reducing system feedback [4]. Typically, vigilance is replaced by complacence when operators begin trusting systems to perform properly. Exacerbating complacence is the fact that system operators shift from actively being involved with the system to passively observing the system, reducing their ability to detect and intervene when problems arise. Finally, automated systems typically hide details of system operation from operators because designers conclude that such details are no longer relevant to operators or they can be overwhelming. The result of such automation is that operator workload decreases during normal operating conditions, but that workload *increases* during critical conditions [10]. When something goes wrong, operators must quickly acquire situation awareness.

Autonomic systems must address these potential situation awareness liabilities of automation. It is counter to the goals of AC to insist that operators maintain active vigilance over autonomic systems, as decreasing overall workload is a driving concern. However, systems can make situation awareness more attainable through two approaches. First, systems should represent their operations in a manner that prompts a sufficiently complete mental model in the operator for normal operations. Second, even systems that do well at representing normal operations must also provide facilities for rapidly gaining deeper situation awareness when problems arise. Because complex computer systems cannot be comprehended in full by administrators, when problems occur, they must provide the ability for learning on-the-fly, for drilling down into and integrating the details of suspected problem areas, and for developing an understanding of what is going on, why it is going on, and what will soon be going on. Administrators will sometimes need to know arbitrary levels of detail about systems' inner workings.

3.3. Multitasking, Interruptions, Diversions

Because of the nature of their environments, administrators have a complex interleaved workflow with multiple tasks conducted in parallel—yet their workflow is often diverted because of missing information, unfulfilled prerequisites, broken tools, or required expertise. Multitasking is particularly an issue for administrators, as they routinely manage a large number of long-running tasks but they must be quite efficient overall.

3.3.1 Managing Multiple Tasks. When tasks are only loosely related, multitasking seems to work without much trouble. For example, we observed a database administrator occasionally monitor the status of a long-running database task in a terminal session while updating some documentation. Yet when tasks are related and attention needs to be divided between tasks, problems may arise. We observed one case where an administrator launched the wrong type of backup because she was discussing a related topic while working at her console. Administrators develop techniques to avoid such problems. We observed a case where a database administrator, during a lull in the middle of a complex and critical task, asked a colleague to go to his office and perform a simple procedure on a test system. The administrator was worried about mixing up her two tasks and typing a command into the wrong console. When asked about this, a group of administrators joked that they kept a running award for the person who had most recently made this kind of error!

When an administrator is multitasking, control consoles should allow simultaneous operations on different systems with enough contextual prompts to avoid confusion. As discussed previously, command consoles tend to do this better than GUI control consoles, which often assume that the operator only needs a single instance of the workspace at a time.

Diversions are a common and expected part of the work. Our analysis of administrators solving problems during routine work suggests that much troubleshooting centers on tools, infrastructures, environments, and other people that are not directly related to the

problems at hand, but that must be dealt with nonetheless. That is, while solving specific computer system problems, administrators often must solve problems that arise outside the scope of the initial problems themselves. For instance, we observed an administrator trying to fix a misconfigured web server. To do this, he needed access to the server machine, which in turn required finding the person responsible for controlling access and convincing her to grant permission. In this case, the original problem concerned software configuration parameters, but the solution required dealing with other sorts of systems and people. And this is not an isolated incident: Observational data from three of our troubleshooting episodes show that about 25% of time was spent on these sorts of diversions.

Usable systems are designed to be flexible, avoiding the trap of assuming and enforcing a particular workflow. Nevertheless, wizards are common artifacts in contemporary GUI applications with such potential usability problems [14]. A wizard provides a multi-step interface for performing a complex task. Unfortunately, many wizards require the user to complete or cancel the wizard to work with another part of the system. It is an unfortunate user who has struggled through a complex wizard to reach the last step only to discover the need for a piece of information that is stored in another part of the application.

3.3.2 Guidelines. The administration of autonomic systems may require significantly more multitasking, interruptions, and diversions from straight-line workflows than conventional systems, as components of autonomic systems will be more tightly interconnected. Administrators will be concerned about diverse components that are currently divided between towers of responsibility. This is one natural result of administrators becoming more efficient through AC: A single administrator will be empowered to control an entire end-to-end application, switching focus between network, storage, database, web server, and countless other parts of the system. Yet all of this power may come at the cost of more multitasking (e.g., as there are more simultaneous problems to solve) and more diversions (e.g., as more problem-solving trails will end up in unexpected places). Even if autonomic computing systems present proper cues to inform administrators of problems (maintaining situation awareness), the sheer scale and scope of such systems may encourage administrators to do more at once and to become lost more easily when troubleshooting than with conventional systems.

Furthermore, autonomic systems will have more levels of control than conventional systems because of the addition of hierarchical autonomic managers. Administering conventional systems means working with many components, but each component works relatively independently. Autonomic systems will consist of basic components, their autonomic managers, and higher-level autonomic managers that manage the managers [7]. (Conventional systems exhibit some of this hierarchical control when clusters and other virtualized subsystems present themselves as a single system). Because each level will affect a component's operation, it will be difficult to design into the user interface a general workflow for debugging.

4. Case Study: Web Application Server

Issues in planning and rehearsing operations, maintaining awareness of tasks and situations, and managing activities given interruptions are mitigated or exacerbated to varying degrees by available system management tools. In this section, we take a more detailed look at IBM's Websphere Application Server (WAS) as a case study.

As the Internet has grown in technical sophistication, web application hosting has evolved through a number of technologies. From the very basic management of HTML files, to CGI scripting, to the J2EE Standard [13] web application servers continue to evolve, but at the cost of added complexity. WAS is IBM's product for serving Java web applications [6]. Like other application servers, WAS v5.1 adds proprietary capabilities to the J2EE standard for market differentiation. All the features and capabilities in WAS, coupled with the database management systems, load balancers, messaging servers, etc. that constitute the web application infrastructure, are well beyond the comprehension of a single person. In this section, we consider the evolution of the administration functions of WAS, paying particular attention to situation awareness, planning and rehearsal, and managing multiple tasks, diversions, and interruptions.

4.1. WAS Administrative Console

The WAS administrative console is the primary graphical interface into the WAS environment. The WAS console has evolved over the past four major product releases from a Java-installed client to a web browser-based thin console that can be used across all product editions and all operating systems, including the IBM mainframe platform. This is an improvement

for WAS users, because in previous releases, users were forced to deal with different consoles across editions (a web browser-base console for single server, and the installed-Java console for advanced edition) and with different consoles across platforms (distributed and mainframe used different consoles and systems management infrastructure). The benefits of a single WAS administrative console are noteworthy. First, the user has a consolidated view of all the applications, application servers, messaging servers, and resources. This is a step toward enabling situational awareness by aggregating constituent parts of the application server environment. By eliminating distractions of negotiating multiple administrative views and systems management tooling, the administrator has more cognitive bandwidth to focus on the information necessary for achieving situational awareness.

Beyond controlling the server, there is also a need for administrators to monitor activity and performance. In our field studies, we rarely saw administrators sit in front of a console and continually navigate among various system views to monitor status. Instead, we observed a lot of reactive responses to system failures. Thus, notification mechanisms could further improve situation awareness by alerting administrators via pager, cell phone, or email when a predetermined event occurs. Indeed, as described previously, this capability has been deemed important enough by some administrators that they have written custom tools and interfaces.

The WAS administrative console offers wizards to help administrators perform complex tasks, e.g. application deployment, that have optional steps and a variable number of steps depending on application type and prior choices. The wizard design identifies each step clearly and allows users to browse task steps without committing to them. This greatly improves activity management over earlier standalone Java-installed client wizards, where administrators could only view one wizard panel at a time, with no composite view of all the steps. Unfortunately, interruptions and diversions are not gracefully supported, because if an administrator is multi-tasking and the console session times-out or if they switch to another task in the console, wizards steps completed thus far are not preserved, and the user must start from the beginning.

Performance data provided with WAS has become more functionally complete, and can provide detailed insights into the current state of servers and applications. Nevertheless, it is only with considerable effort and experience that an administrator can retrieve detailed information by launching free-standing tools, and taking steps to locate and retrieve what is relevant. To further improve situation awareness in a WAS environment, the administrative console should provide the most important health indicators by default while making detailed metrics available by drilling down. An aggregate layer across multiple instances and abstracted metrics that convey a high-level health summary should also assist in more quickly comprehending what is going on. Furthermore, as mentioned previously, many administrators prefer (or require) command line tooling, which suggests that CLI equivalents to all GUI functions must be provided.

4.2. WAS Autonomic Capabilities

Autonomic capabilities are being added to WAS to support the management of complex installations. For example, the WAS Performance Advisor is an AC feature that acts as a performance analysis expert. As a planning tool, the Performance Advisor assists administrators in understanding the run-time performance characteristics of their system configuration in a test environment. It eliminates the complex drudgery of manually collecting relevant system performance data, and provides the capability for automatically analyzing the data and suggesting actions.

Another WAS AC feature is the Log and Trace Analyzer (LTA) [5]. The LTA is a standalone tool that imports activity logs from the application server, web servers, and the backend database. Because it extends beyond the application server, LTA begins to assist administrators in understanding what is happening in a larger part of their overall systems. The LTA allows an administrator to correlate events from different logs (e.g., application server and web server). Using this feature, an administrator can track a series of events as they occur across system components. These correlated logs can be useful for diagnosing problems. In addition, log entries can be compared to symptom databases that contain diagnostic information. The symptom database entries provide insight into potential causes of an event that have been determined from previous diagnostic activities.

As autonomic computing matures, WAS will incorporate increasingly capable features for information gathering, analysis, and eventually proactive actions on behalf of system administrators. These capabilities will be necessary for managing increasingly complex installations. However, caution must be exercised to avoid the problem of distancing system administrators from the workings of their

systems, alienating the very ones who are responsible for their correct function.

5. Conclusions

System administration is a difficult task and is rapidly becoming more difficult as systems relentless grow more complex. The autonomic computing initiative aims to dramatically transform the way systems are managed by introducing automation at every level. Automation can greatly ease human burdens, but also carries risks if it is not implemented well. Rehearsal and planning will become even more necessary in autonomic systems, necessitating the creation of test replicas of production systems, fast backtracking from errors, means for building common ground between the system and the administrator about the meaning of high-level commands, and the ability to test the system-wide implications of changes. Maintaining situation awareness over systems too complex to comprehend means that the representation of the system to the user should be sufficiently complete for all normal situations while providing access to arbitrary degrees of detail during unusual operations. Handling multitasking, interruptions and diversions in autonomic computing operations means that interfaces must allow fluid movement throughout the system and maintain enough contextual clues so that administrators can easily shift between tasks. These guidelines for the design of human interfaces to autonomic computing are a step toward minimizing its risks and maximizing its potential for relieving system administrator workloads.

6. References

[1] Bailey, J., Etgen, M. & Freeman, K. Situation awareness and system administration. In Barrett, R., Chen, M., & Maglio, P. P. (Eds). *System Administrators are Users, Too: Designing Workspaces for Managing Internet-scale Systems*, CHI 2003 Workshop.

[2] Barrett, R., Chen, M., & Maglio, P. P. *System Administrators are Users, Too: Designing Workspaces for Managing Internet-scale Systems*, CHI 2003 Workshop.

[3] Brown, A. B. & Patterson, D. A. Undo for operators: Building an undoable e-mail store. In *Proceedings of the USENIX Annual Technical Conference*, San Antonio TX, 2003.

[4] Endsley, Mica R., Automation and Situation Awareness. In Parasuraman, R., Mouloua, M., (Eds). *Automation and Human Performance – Theory and Applications*, Lawrence Erlbaum Associates, New Jersey, 1996.

[5] IBM, Log and Trace Analyzer for Autonomic Computing, AlphaWorks Release, available at http://www.alphaworks.ibm.com/tech/logandtrace.

[6] IBM, WebSphere Application Server, available at http://www.ibm.com/software/webservers/appserv/

[7] Kephart, J. O., Chess, D. M. The Vision of Autonomic Computing, *IEEE Computer*, January 2003, 41-51.

[8] Maglio, P. P., Kandogan, E., & Haber, E. Distributed cognition and joint activity in collaborative problem solving. In *Proceedings of the Twenty-fifth Annual Conference of the Cognitive Science Society*. Boston, MA, 2003.

[9] Oppenheimer, D. The importance of understanding distributed system configuration. In Barrett, R., Chen, M., & Maglio, P. P. (Eds). *System Administrators are Users, Too: Designing Workspaces for Managing Internet-scale Systems*, CHI 2003 Workshop.

[10] Parasuraman, R., Mouloua, M., Molloy R., Hilburn, B., Monitoring of Automated Systems. In Parasuraman, R., Mouloua, M., (Eds). *Automation and Human Performance – Theory and Applications*, Lawrence Erlbaum Associates, New Jersey, 1996.

[11] Russell, D. M., Maglio, P. P., Dordick, R., & Neti, C. Dealing with ghosts: Managing the user experience of autonomic computing. *IBM Systems Journal, 42*, 2003, 177--188.

[12] Sheridan, T. B. *Humans and Automation: System Design and Research Issues*. Wiley-Interscience, 2002.

[13] Spool, J. M., Snyder, C., Designing for Complex Products, *Proc. ACM SIGCHI*, Tutorials, 1995, pp. 395-39.

[14] Sun, Java 2 Platform Enterprise Edition (J2EE), available at http://java.sun.com/j2ee/ .

[15] Wiener, E. L. (1989). Human factors of advanced technology ("glass cockpit") transport aircraft (Technical Report 117528). Moffett Field, CA: NASA – Ames Research Center.

[16] Woods, D. D., Decomposing Automation: Apparent Simplicity, Real Complexity. In Parasuraman, R., Mouloua, M., (Eds). *Automation and Human Performance – Theory and Applications*, Lawrence Erlbaum Associates, New Jersey, 1996.

7. Acknowledgments

We thank Chris Campbell, Steve Farrell, Eben Haber, Madhu Prabaker, Anna Zacchi, and Leila Takayama for help collecting and analyzing data, and the many system administrators who let us watch.