Dealing with ghosts: Managing the user experience of autonomic computing

by D. M. Russell
P. P. Maglio
R. Dordick
C. Neti

Although the goal of autonomic computing is to make systems that work continuously, robustly, and simply, no one imagines that people can be excluded entirely. Whether it is end users getting their jobs done by interacting with autonomic systems or system administrators maintaining, monitoring, and debugging large-scale systems with autonomic components, humans will always be part of the computational process. As autonomic systems become part of the computing infrastructure, new demands will be placed on all users. How do users understand what autonomic systems are trying to do? How should systems portray themselves to users? How can we design the experience of autonomic computing to amplify user capabilities? This paper presents an analysis of the user experience challenges of autonomic computing and discusses design requirements for user interaction. Our main point is that autonomic computing makes effective design of the user experience even more challenging and critical than it is now. The reason is that autonomic actions taken by the system must be understandable by the user and capable of review, revision, and alteration. Because such actions are often made autonomously, a heavy burden is placed on the ability of the system to explain what it is doing and why.

Autonomic computing systems manage themselves, taking self-directed action when deemed appropriate and making choices about what needs to be done and how exactly to do it. Choices are made and actions are taken in the hope of reducing overall system complexity and cost. Yet, this idea of autonomic computing is not entirely new or surprising. For instance, computer systems have been managing their resource pools for years: memory is allocated and freed on a regular basis without human intervention, program components are brought into service and taken out of service when appropriate, memory is monitored for errors and corrected on-the-fly deep within random access memory, and communication channels are monitored for mistakes, which are automatically corrected when they arise. Nevertheless, computing systems are now so large and complex that moment-by-moment human management is often the dominant cost.[1,2] There is a need for computing systems to take care of themselves at a higher level, eliminating much of what is done today by human monitoring, maintenance, and control.

The key analogy of autonomic computing is that of the autonomic nervous system, which takes care of many low-level functions in animals. For example, pupils dilate, stomachs balance enzyme and acid levels, and body posture and balance are maintained under a wide variety of conditions. Although humans are sometimes aware of the functioning of the autonomic nervous system, autonomic functions are, for the most part, unconscious and self-regulating.

Autonomic body functions are implemented by a mostly separate system of nerves—the sympathetic and parasympathetic nerves, associated ganglia, and a few others—a system that has been refined and debugged by evolution. Behaviors created by the autonomic nervous system are primarily direct responses to internally and externally sensed environmental conditions. Breathing and heart rate are modulated by levels of carbon dioxide and other chemical signals in the blood. The "fight-or-flight" response is a rapid, coordinated release of hormones to prepare the body for sudden immediate action required in response to a perceived threat.

Although the autonomic nervous system is finely honed and wonderfully responsive, it has become what it is in response to evolutionary pressures. In many ways, it is a system that best fits hunter-gatherers on an African savanna. The fight-or-flight response, so useful when confronted with an immediate threat requiring rapid physical response, is often inappropriate in the modern world. As humans, we suffer from constant pulls on the chain of our autonomic nervous system—stressors in our environment invoke this system in ways that end up causing stress-related disorders. Rather than constantly confronting relatively short-term physical threats, modern life calls for more calm, measured responses over a long period, such as those required when sitting before a computer screen, while driving a car, performing difficult surgery, or awaiting the results of a grant review.[3,4] Yet given extended periods of stress, the autonomic nervous system interacts with other systems, both internal and external, to produce less than optimal results. Of course, the autonomic nervous system is merely a metaphor, suggesting sophisticated self-managing and self-regulating systems, rather than a biologically plausible guide for design. But if we are not careful in developing autonomic computer systems, we might design ourselves into the same fate: self-regulating systems that suffer from unintended and unforeseen interactions that lead to undesirable results. In particular, we focus here on the human-computer interaction aspects of this new kind of system, or the user experience of autonomic computing.

In what follows, we tell our story in four parts. First, we consider how users might view autonomic systems, arguing that as systems become more automatic, they often become less understandable. Second, we describe autonomic computing from the perspective of the system, arguing that a three-layer model describes nearly all autonomic systems. Third, we examine several existing systems that have autonomic properties, and we learn some lessons about what works and does not work from the user's perspective. Finally, based on this analysis, we discuss several challenges that will be faced by autonomic system designers.

The user view of autonomic computing: Ghosts in the machine

When systems manage their own very high-level tasks, their behaviors and the interaction of their behaviors might be unpredictable and inscrutable, complicated, and hidden from view. From a user's perspective, an autonomic system might seem to have "magical" properties, where things happen without apparent or determinate causes. The basic issue for users of autonomic systems will be understandability, which depends on the complexity of the system and the user's trust that the system will behave in the way it has behaved previously. Complexity results from the interactions of a variety of systems and unexpected behaviors. Trust reflects the relationship between user and computer, the history the two share, what the user knows about the ability of the system to control things, what the user has seen the system do, and how well the system has explained its actions and decisions, among others.

Human-computer interaction is a kind of "joint activity," just as human-human conversation is a shared activity between participants in a conversation.[5] To accomplish some computational task, a person works with a computer, each side—human and machine—contributing something to the process and over time building up a shared understanding or common ground. There is recent evidence that when persons interact with computers at higher and higher levels (e.g., verbally), the persons assume "human" sorts of common ground at the start.[6,7] If this expectation is not met—and yet the computer seems to be a high-level participant in some joint activity—it might well seem as though there is a "ghost in the machine," one that users must understand to be effective participants themselves. Thus, being "magical" is what is just out of reach of understanding.

The phrase "ghost in the machine" comes from Ryle,[8] who pejoratively described the Cartesian view of mental states as the "ghost" that inhabits physical brains, doing invisible work with perceivable consequences. In fact, Ryle argued convincingly against the Cartesian view of a mind-body dualism, showing that the distinction between mind and body was

based on a category-mistake, supposing that the mind is the same kind of entity as the body or brain and so must be explained in the same way. Ryle comes to the view that minds are not like brains but, rather, minds are simply what brains do, made of a constellation of interacting abilities and skills. Now, Ryle

Ironically, automation can decrease system transparency, increase system complexity, and limit opportunities for human-system interactions.

may further inspire us in the structure of our argument. If a modest goal of autonomic computing is to eliminate common and time-consuming user tasks, then computational processes can replace some user tasks. If we take the supposition further, imagining that the ultimate goal of autonomic computing is to eliminate all user tasks, then it follows that user tasks and computational processes are wholly interchangeable; that is, what users do and what computers do are relatively the same. Is this a reasonable conclusion? We think not. Though it may be argued that thought is a computational process and, therefore, that user activities and computational activities amount to the same kind of activity, from a human-computer interaction perspective, it may also be argued that users and computers serve different purposes: computers are tools that enable users to accomplish certain jobs. In this case, user and computer are different kinds of entities: controller and controlled. Moreover, if we suppose that humans and computers have different abilities—do different things more or less easily than each other—then it is appropriate to distribute the computational load between users and computers. For instance, people are good at many kinds of visual pattern recognition, whereas computers are good at counting and correlating. Given these different abilities, it might be best to have a computational process analyze some data and then display the data to users who can easily pick out patterns. This approach is the one taken by work in data visualization. The bottom line is that it makes sense to combine the processing of people and machines to create a more effective overall computation than can be done by either one alone.

Consider, for example, that disasters in domains with tight safety and design guidelines, such as nuclear reactors and aircraft, are blamed on human error 60 percent to 80 percent of the time.[9] Although this number seems high, many such "normal accidents" actually stem from design flaws, specifically, failures of communication in the human-computer joint system. In retrospect, the flaws seem evident. But in predicting failure cases, designers often deem failure probabilities too low or the need for rapid human intervention too unlikely to design for failure. Nevertheless, it is well documented that humans make all kinds of errors,[10–13] including *slips* or *lapses*, which are errors in execution (not doing what was intended), and *mistakes*, which are errors in planning (choosing and executing the wrong course of action). *Automation irony* is the observation that automation does not cure human error because designers automate things that are easy to automate, leaving the most complex and intractable problems for people to solve.[14] Though designers know that people make mistakes, they often try to design systems that reduce the need for human intervention. Consequently, automation usually addresses the tasks that can be easily automated, leaving only complex, difficult, and rare tasks. In this way, automation reduces the chance for operators to obtain hands-on experience; having been taken out of the loop, they are no longer vigilant or completely aware of the current operating context. Thus, ironically, automation can decrease system transparency, increase system complexity, and limit opportunities for human-system interactions, all of which can make a system harder for people to use and make it more likely that they will make mistakes.[11]

From the user's point of view, the fundamental issue concerns how the overall behavior of the system can be understood. Our approach is to design autonomic behaviors not simply for the sake of the system, but so that users can be effectively engaged in a joint activity with the system.

View of autonomic computing by the system: sense-model-act

As described in IBM's vision statement about autonomic computing, an autonomic system has the properties of self-management, self-healing, and self-optimization.[15] What is the common thread throughout these desiderata? In our view, an autonomic computing system is one that (a) senses its operating environment, (b) models its behavior in that environment, and (c) takes action to change the environment or its behavior. It is important to the sense of autonomy that the actions take place without human intervention.

Sensing. Autonomic systems must perceive what is going on in the world and within the system. Often sensing is purely programmatic, based on internally represented measures of conditions such as buffer pool size, number of currently running applications, errors thrown by applications, and so on. But increasingly, sensing for the user experience will involve detecting what the user is doing: Is the user using an input device; has the system been untouched for some amount of time; is there a person in front of the keyboard?

Model. The model represents the goals, environment, and behaviors of the system. These characteristics can be fairly simple (such as the set-point of a thermostat representing the ideal temperature at which to activate the furnace), or a model can be rich and complex (such as the SMART system of DB2**, which adapts multiple database parameters to achieve continuous high levels of performance).[16] In all cases, the model is an explicit representation of the ideal world state or condition that the autonomic system uses to derive what action to take to achieve the ideal. (Bear in mind that the computation on the model might well be embodied in a piece of code that encapsulates the representation, rather than a more symbol-processing style of explicit symbol manipulation.)

Act. After the world is sensed and the model consulted, actions are taken to modify the internal system or the external world. Generally these are straightforward tasks: back up a file, adjust a system setting, adapt some property of the user interface. But some actions can be long-lasting and complex: restoring a system to a previous working version, or automatically installing new software.

Simply put, an autonomic system senses the world state, models what should be the ideal case, and acts (or chooses to not act) to transform the world to what the model says it should be. This is the most basic of control paradigms, but one that when embedded in a rich, complex environment creates a rich, complex operating milieu for the human user to understand. Autonomous action is taken by the system to achieve some desired goal state, such as an optimized database or a system running at the best throughput levels. The goal, of course, is that it all just works, that systems behave in the way we expect them to behave. Yet someone must establish guidelines for system behavior. Somehow, the system must know the right thing to do at any time under widely varying conditions. For autonomic systems, establishing the right and appropriate action to take is an integral part of enabling the user's understanding of behavior.

Thus, a complete model of the system necessarily includes the user. Ultimately, the computer systems we build exist for some human-centered reason. Even middleware services have users at the end of their processing. And any reading of the literature on software engineering assures us that software errors will be with us for some time,[17,18] suggesting that people will, for the foreseeable future, be involved in the use, care, maintenance, and repair of computer systems.

Autonomic action happens at different levels. Some actions are visible in the user interface (UI), such as a desktop display that periodically examines the desktop and reorganizes icons to "clean up their layout." Some are far less visible, such as a file transfer that is interrupted by network connection loss, but that resumes when next connected to the network. Rather than trying to achieve just a single goal, an autonomic system works to achieve many goals simultaneously, at many different levels of user visibility and awareness.

Note that simply making software better is not the same as creating an autonomic computing system. For example, it is well known in the software engineering community that the simplicity and ease-of-use of the Macintosh** operating system is superb. A common user task such as "printing a file" is significantly simpler from the Macintosh system than from Linux** or Microsoft operating systems. There is no magic involved: protocols for resource discovery and the widespread implementation (on printers) of the AppleTalk** network communication standard and PostScript** standard on printers make for effective printing. Despite the confusing welter of drivers, complex network naming schemes, and incompatibly implemented standards, Macintosh owners can plug their machines into a network, and printing simply works. Other kinds of systems expose the confusion and complexity to the user, making printing less automatic. Although laudable and useful, the Macintosh approach to printing does not fall under our definition of autonomic computing. It is just good software engineering. In this paper, we focus on autonomic computing that takes place out of the user's sight—work that is automatically performed, driven by an underlying model of the system, not overtly directed by the user, and not conspicuous in its action.

Autonomic lessons and analysis

Autonomic systems have existed in various guises for some time. According to our sense-model-act paradigm, we can see that many basic operating system functions are in fact autonomic systems. For instance, dynamic memory allocation, with its sensing of fragmentation and model of maximizing contiguous block size, can be thought of as a basic autonomic subsystem. But autonomic systems often have a human-facing side that we have argued is a critical part of the overall story. To make our argument concrete, we now present six brief case studies, each with a set of object lessons in the design of autonomic systems from the user perspective.

Case 1—Importance of UI design. In July 1988, the USS Vincennes was patrolling the restricted waters of the Persian Gulf with the Aegis missile defense system onboard. During an attack by several small gunboats, the Vincennes was in a defensive posture, with the ship making rapid turns and with troops on active battle status. The effect of the maneuvers was dramatic: "Books . . . and loose equipment went flying off desks. Desk and file drawers flew open. Many of those on board had to grab for the nearest support to avoid being thrown to the deck . . . the situation aboard the Vincennes that day was one of confusion and disorder."[19]

In this chaos, the crew of the Vincennes interpreted their Aegis display as signaling an attack by an incoming Iranian F-14 fighter. The operator—on the basis of information displayed on the Aegis user interface—believed that the fighter was rapidly descending to prepare for an attack approach on the ship. After ineffective efforts to communicate with it on military voice channels, the Vincennes was authorized to fire two missiles and remove the threat. Sadly, the threat was misidentified, and IranAir Flight 655 was destroyed with the loss of 259 lives.

In later analysis, Matt Jaffe, one of the designers of the Aegis display interface, reported that the altitude information was difficult to interpret correctly.[20] After all, Aegis, like many fire control systems, is highly automatic. Threats are identified and targets selected and tracked. It is an autonomic system with a sense-model-act architecture.

In this instance, the threatening aircraft altitude was not shown on the main display, but required that the operator request it, when it would be shown in a subwindow with other ancillary data. And rather than show a rate of altitude change (as is common in aircraft displays), the altitude of the threat was shown as a numeric display, requiring that the Aegis operator do mental arithmetic to determine altitude increase or decrease—difficult in normal circumstances, although clearly learnable. But under the stress of battle it would be all too simple to make an error in arithmetic, especially while the display is rapidly changing. Thus, it would be simple to believe that the unknown incoming jet really was in an attack flight pattern, and difficult to believe it was not a hostile aircraft.

The Aegis system and display was designed for a demanding task: the rapid deployment of defensive armament in a battle situation. In this case, the unanticipated circumstance of a civilian airliner in a war zone during a time of high stress led to a rapid decision. Everything worked as planned, except for the initial conditions, which were unanticipated in all planning scenarios, and the difficulty of using the Aegis interface, which lent itself to a predictable, high-probability mistake (in cognitive load terms).

Lessons learned:

1. Autonomic applications need to support reliable user assessments of system and world states to ensure proper operation in an unpredictable environment.
2. Information needed to make an appraisal of the situation must be displayed so the user can easily find the data and make sense of the data.
3. User attention is a vital resource: autonomic systems need to model what to tell the user in particular situations. Just telling everything (or hiding everything) is not a good solution.

Case 2—Visible autonomic interaction. The spell-checking and automatic correction facility of the Microsoft Word** program for text processing can be seen as an autonomic system. Running in real time, the Word program checks each completed word as entered, and subtly signals the user whether it is considered incorrect or not. In simple, obvious cases, the word is automatically corrected. Behind the scenes, the Word program senses the user's completion of the word, and compares it against a model of proper spelling. It then takes an action to mark the word as valid (no mark) or incorrect and unguessable (a red, wavy underline) or, if the word is an "obvious" misspelling, replace the word with a better choice (one that fits the model of correct behavior).

In practice, many people find this capability tremendously useful. We tolerate the red underlining of proper names and the occasional overcorrection of certain words. In earlier systems, such as the Interlisp-D DWIM (Do-What-I-Mean) system, correction was not as graceful, requiring much user intervention to prevent common spelling corrections from mangling program code. In the main, because of the degree of interventions needed to ensure proper operation, people turned DWIM off. When people are engaged in a rapid-fire, productive task such as writing with a text editor, distractions are extraordinarily damaging to task performance.

Such disruptions still occur (just apparently below a user's annoyance level) when the corrector in a program mistakenly "corrects" a word to something the user did not desire. As an example, try typing the word Hawai'i using the Word program. The apostrophe is present in the correct spelling, yet Word insists on "correcting" the last "i" to be an uppercase "I." Practiced users know that control-Z will undo the mistake of the spellchecker by undoing an autonomic action.

By contrast with its spelling corrections, the Word grammar checker does not receive quite as much universal acclaim. Grammar has a much richer, less constrained definition of "correct," and there is a breadth of opinion about what should be considered proper usage. In our anecdotal evidence, far fewer people use (or believe!) the recommendations of the grammar checker, and it too, like DWIM, is often turned off.

Lessons learned:

1. Carefully tune autonomic interactions for smooth use, avoid interrupting a user's primary tasks.
2. Allow users to back out of changes easily, including autonomically driven actions.
3. Make actions visible whenever the actions are relevant to user understanding of system behavior. (But be aware of the cognitive load demands being placed on the user.)
4. Autonomic decisions to act must be correct (to the user's belief) in the vast majority of cases.

Case 3—Trust and latent errors. On January 31, 2000, Alaska Airlines flight 261 was flying an MD-83 airplane from Puerto Vallarta to San Francisco in clear, smooth air when the plane became difficult to control. After many minutes of valiantly trying to regain control, the big plane suddenly inverted and plunged into the sea just west of Los Angeles at 219 knots.

The horizontal stabilizer jackscrew—the main actuator of a vital control surface—was later found to have become stripped, with loose strips of metal detached from the helix of the screw.

In the cockpit, the pilots experienced a rapid and sudden loss of ability to control the attitude of the aircraft. The plane rapidly made a transition from normal to very erratic flight, becoming essentially uncontrollable in a matter of moments.

What happened? Part of the answer seems to lie in a *latent error*—an error somewhere in the system that does not become apparent until much later.[13] In the MD-83 there are control systems that compensate for the gradual deterioration of some subsystems, such as with normal wear on control surface drivers. As their performance degrades, the compensation mechanisms take over—in this case, silently—continuing to make the plane fly well and function normally.

A serious problem arises when the compensatory systems can exceed their ability to manage. When the jackscrew mechanism went from difficult-to-manage to uncontrollable, the compensators became irrelevant: nothing could drive the jackscrew, and the plane spiraled out of control.[21]

The deeper issue for autonomic systems design is one of trust and understanding. When errors occur, often they can be corrected autonomically. The system can hold itself together by an accumulation of dynamic compensations and small patches until a critical event occurs and overwhelms the whole.

In many complex systems, errors need to be automatically handled. Simply reporting all such errors to the operator would overwhelm the humans with additional tasks that need correction. " . . . automating a process may reduce the chance of a human operator doing the wrong thing, but such a change merely shifts trust from one set of human beings to another . . . "[22]

Lessons learned:

1. We need to understand how compensation mechanisms might mask other problems, ones that could become critical to overall systems perfor-

Figure 1 Standard file browser window in Windows 2000

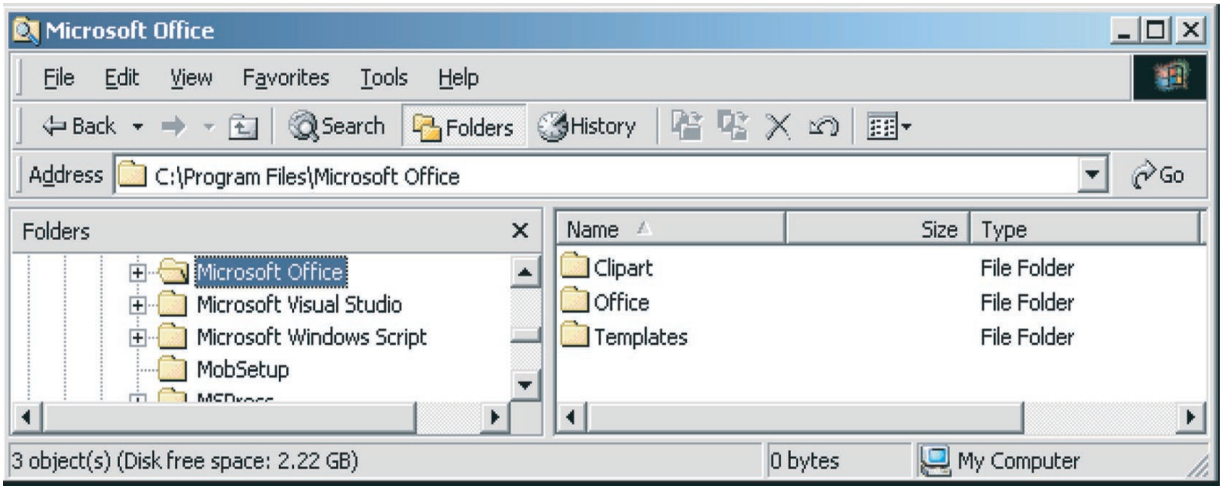
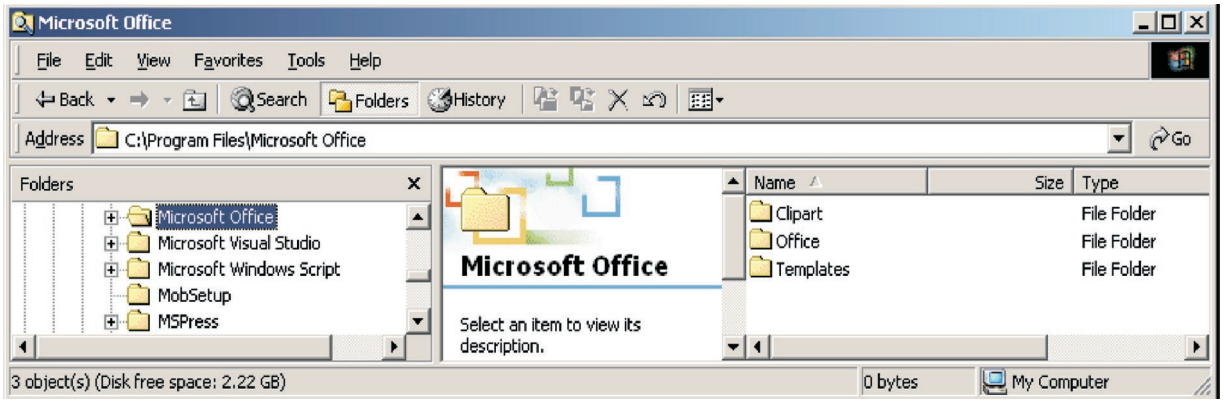


Figure 2 A wider file browser window



mance. When should an autonomic system tell a human that something is amiss?

2. More generally, how can we best design information—revealing systems so the right information is available and readily accessible (or even intrudes into consciousness)?

Case 4 — Guessing user intent. You might think that guessing a user's intent would be a straightforward task. After all, everything that can happen on a computer system is right there, exposed for all to see. So system builders try to anticipate the best and most common uses of the tools they provide.

For instance, in Microsoft Windows** 2000, the file browser attempts to adapt its behavior to what the user wants to do. As the user makes the window larger, the layout dynamically adapts itself to offer more information as more space becomes available. Unfortunately, if the user is trying to see the file modification dates, that is (by default) the fourth column. When the window is small, the file preview area is not visible. (See Figures 1 and 2.)

As Figure 1 shows, a standard file browser window does not have enough space to show the file modification dates. If a task depends on knowing when

a file was modified, just making the window larger to see those dates takes much longer than would be expected.

Figure 2 shows that making the window wider (to expose the file modification dates) causes the browser to adaptively add a "file preview area" (the window pane with colored artwork), which perversely moves the file modification dates still farther off the right edge. In this case the automatic adaptation is making the user's task harder by requiring still another window enlargement. The point is not that this adaptive window manager is wrong—it is a genuinely useful feature. It is just not helpful in all cases, and there is (currently) no way to understand that what the user wants to really see are the file modification dates. That level of detail and precision of understanding the user's intent makes it a difficult problem.

In autonomic computing terms, the system is sensing the user's behavior, modeling what the right display should be to fit this kind of action, then acting to modify the browser display.

Yet this kind of thing happens all the time in user interfaces. As users know from wide-ranging experience, doing a "search" within a collection of texts is straightforward: the user enters a string into a text box and starts the search. But what does "search" mean? It varies from application to application. In the Eudora e-mail client, the default search is to find the specified string anywhere it appears as a substring in each e-mail message (including header content). By contrast, "search" in the Lotus Notes e-mail client means "find this string only as a separate token—never as a substring of another, larger word." Thus, if it is necessary to find all messages from a colleague with a long and difficult-to-remember name that consists of 11 characters, merely remembering the unique first five characters of his or her name will not work. All 11 characters will have to be entered. (For example, searching for "Czerw" in your e-mail will not find any instances of mail from "Czerwinski.")

Lessons learned:

1. How does the user know what an action means? Suggestion: better feedback about what is actually going on, done in a progressive disclosure style.
2. Improve our systems ability to watch what a user does and take reasonable action based on that behavior.

Case 5—Inscrutable systems. How often does a current computer system do things that are apparently inexplicable? Daily? Hourly? And this is without highly autonomic systems. Users calling into a corporate help desk frequently complain that the system did something of its own accord. The callers often have no idea why a particular behavior is suddenly made manifest, nor what they can do to change its behavior. For example, "Why does my laptop sometimes wake up at midnight and dial into the Internet?" or "Why did it start and how do I stop it?" When confronted with mysterious system behavior at an end-user level, people most often call on an expert, frequently someone local, to help understand the problem and suggest a solution.

Autonomic systems might be similarly opaque: they have the potential for making inscrutability intractable. If there is no way to examine the history of a system, then it becomes increasingly difficult for us to learn what is going on and why.

Lessons learned:

1. Changes to systems should be recorded, especially changes to system behavior that are not performed by overt user action (e.g., by a wizard).

Case 6—Anticipating failure. RAID (redundant array of independent disks) disk systems anticipate failures by distributing stored content over multiple independent disks with some redundancy. If a single disk fails, it can be simply removed from the array and replaced. The system then automatically corrects the data and brings the new drive into the system by correctly updating its contents, making the whole array once again protected from failure.

Research into RAID systems suggested that as long as there were standby spares on which to rebuild lost data, RAID-5 would recover from bad drives, and so they assured others. A system administrator remarked recently that every administrator he knew had lost data on a RAID-5 system at one time in his or her career, even though they had standby spare disks.[23] How could that be? In retrospect, the quoted mean time to failure of disks assumes ideal temperature and limited vibration. Surely, some RAID systems were exposed to higher temperatures and more vibration than anticipated, and hence had failures much more closely correlated than predicted. A sec-

ond problem that occurs in RAID systems is operator removal of a good disk instead of the failed disk, thereby inducing a second failure on top of the first failure.[19]

Lessons learned:

1. When anticipating failure modes, we need to take into account not just hardware and software failures, but also potentially incorrect human interventions.
2. RAID-5 is enough for faults we could anticipate under nominal conditions. But wise engineering practice should have recommended RAID-6 (tolerates up to two disk failures) to accommodate the unanticipated faults in systems that were working under real conditions. Design for reality.

Challenges for user experience

An ideal autonomic system just takes care of itself, sensing changes in the internal and external world state, modeling those changes in terms of ideal system behavior, and taking action accordingly. On the joint activity view, system behavior must be understandable, or else users and systems cannot work together effectively. What are some potential design guidelines for user experience? The lessons from our case studies provide some suggestions.

Design for joint human-computer activity. It seems clear that user interfaces must make autonomic actions visible (or at least easily accessible) to the user. It is inevitable that at some point, a human will need to understand what is going on in the system. Visible behaviors are reasonably straightforward, although potentially complex. Invisible behaviors are commonplace—one needs only to look at an automobile or jet aircraft to see that a great deal of system management can happen automatically. The real design issues arise when behavior varies from the expected. When the user wants to do something different, when errors arise, or when complex behavior emerges, the user must take action, such as describing a change or fixing an error. System builders must take account of human cognitive and perceptual abilities, incorporating a model of what users need to know, when users need to know it, and how best to portray information.

Design with human goals in mind. Users need to describe what the desired system behavior ought to be. In many ways, autonomic computing adds one whole new level of indirection. Users will have to describe behaviors in terms of goals, rather than in terms of actions; yet we know that people are especially poor at describing goal states. Rarely does someone speak of wanting a quarter-inch diameter hole; usually someone says he or she needs to drill a quarter-inch hole. Some systems have very natural goal description tools: an autopilot is programmed by specifying destination, whereas flying by wire (a complex human autonomic activity) is naturally done through the control stick and pedals of an aircraft. Whatever the task and whatever the domain, goals must be described in a simple manner that is understandable to the user in terms the user understands.[24]

System behavior must be understandable, or users and systems cannot work together effectively.

Design for limited sensing. A system might like to do something to help, but is constrained by its ability to sense what is going on in the world. For instance, we can imagine building an autonomic system that automatically logs in the user when presented with a log-in dialog box; however, this is a huge security risk unless the person sitting in front of the machine can be positively identified. Further, unless the system can identify the source of the log-in dialog (i.e., where are you logging in to?), the problem of figuring out which identifier and password to use becomes difficult (and potentially a hassle, as some log-in challenges give only a limited number of attempts before locking the user out for some period of time). More generally, sensing of the world state (both external and internal system status) is not a systemic goal. Although the keyboard of a computer can be thought of as 70 (or more) touch sensors, the activity of the person using the keyboard is not sensed and represented as touch but rather as a byte stream of characters.

Design for limited modeling. Misbehaviors also creep into systems because the models they are based upon have real limitations. What is worse, such models are typically not aware of their boundary conditions. Modeling is, by definition, an abstraction of the full complexity of reality into a computationally tractable form. Although good, accurate, and predictive

models can be built, rarely are they constructed with a notion of appropriate use and context. What works (and is predictive) in one environment, with one particular set of software, may not be applicable or useful when that environment changes. And although advances have been made in modeling user behavior and user state, much remains to be done.

For instance, text gathered by monitoring user interactions with the computer can be combined and analyzed to produce a small list of keywords that characterize the users's working topic.[25–27] In many cases, text is gathered from user keyboard input, from user e-mail, from Web pages read, and from files visited with editors. Keywords are derived from these text sources by determining the frequency of the words in the pooled text at a given time relative to the frequency of the words overall. The keywords are those words whose frequency is high in the current set relative to their overall frequency. Collectively, this forms a kind of model of the user's working (operational) interests. This kind of model is intrinsically limited in what it can capture and model because of the relative lack of information about what the user is actually doing. Of course, more complex user models can be constructed from sensed data, including statistical or probabilistic models,[28–30] but models are always limited by their abstractions and the modeling technology available at the time. Autonomic systems design must take these limits into account as much as any other factor.

Design for errors. There is one essential and deep truth in computer science: errors will always be with us. Errors arise from software failures, hardware failures and not incidentally from user-caused mistakes.[10] When they occur, humans must understand what is going on and be able to resolve the issue or at least come up with a way to work around it.[18,31] The biggest question for autonomic systems will be one of designing for those cases when errors happen and human intervention is needed, particularly in systems that have grown so large and complex that it could be that no one truly understands how the entire system operates.

New research directions. Human-computer interaction must be present in autonomic computing design from its beginnings. Yet we see that the following basic research questions remain:

1. How do people create an understanding of how systems function? We know that relatively few people read long, detailed manuals of operation. How should systems present their ongoing state and their moment-by-moment status? How do we design for joint human-computer activity and for the establishment of human-computer trust?
2. How do we know whether autonomic systems actually improve human use of computers? To determine this, we need to quantify what humans are actually doing when using computers to discover the impact of specific automated functions. We might find, for instance, that automating parameter tuning to optimize system performance provides only a small benefit to users compared to the benefit of providing more informative error messages. But how can we measure user activity? How can we determine the high-level tasks human-computer systems are engaged in?
3. How should we notify users of problems or of normal system operation? We need to better understand user notification and awareness of the system model.

Conclusion

Our argument should be clear: Increasingly autonomic computing systems require an ever greater attention to the design of the user experience. Whereas the goal of autonomic computing is to reduce the total cost of ownership by reducing the number of decisions the user must handle, the trade-off is to potentially increase the amount of "magic" and inscrutable behavior in the system. This is particularly true as systems grow ever more like composites of many parts. Clear and simple decisions that can be made by a system in isolation become difficult and complex when embedded in a larger web of other interacting systems.

Although having "ghosts in the machine" is probably a necessary precondition for the future growth of computing systems, there is specific work to be done on the critical path to autonomic computing "nirvana." We need to change our thinking about the interaction between humans and systems to consider the work underway as shared, joint activity between human and system. In particular, we must develop (a) user attention management models to avoid swamping users with system messages, (b) system inspectability for postfailure analysis, understanding, and recovery, and (c) behavior-reporting mechanisms that will allow users to understand what the system is doing with a measure of trust and confidence.

In summary, our main arguments are the following:

- People cannot be removed from interacting with computing systems for anything but the simplest, most predictable of tasks. System tasks that operate under well-understood principles (memory allocation, buffer pool management, load balancing, etc.), with accurate sensing and well-understood actions, are well suited for autonomic computing. Tasks that rely intrinsically on user interaction, or critically depend on external world-state are often not well suited for autonomic computing.
- Complex systems can seem like magic, especially when they take action of their own volition. But they need not seem like magic—understandability depends on creating (or portraying) a model of the system for the user and the creation of trust that the system will behave in the way the user's mental model suggests.
- Humans must understand what is going on to be effective participants in the joint activity of human-computer interaction. To understand what is happening in the system, humans must be able to observe enough of the processes and interactions to build up a trust of the system. As a consequence, we must design autonomic systems so that their processes are transparent, providing explanations of their actions and running commentaries of their internal states.
- Finally, humans must be able to communicate with these systems in very high-level terms. With autonomic systems that can take care of so much, people will begin expecting them to be more human-like in capabilities and responses, dragging social psychology and an entirely new raft of human-computer interaction issues into the mix.

Acknowledgments

No paper of this scope is written in isolation. The authors would like to thank Rob Barrett, Kazuo Iwano, and Donald Norman for very useful background discussions on these topics.

Cited references

1. T. K. Landauer, *The Trouble with Computers*, MIT Press, Cambridge, MA (1995).
2. D. A. Patterson et al., *Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies*, UC Berkeley Computer Science Technical Report, UCB//CSD-02-1175, University of California, Berkeley (March 2002).
3. R. Sapolsky, "Why Stress Is Bad for Your Brain," *Science* **273**, 749 (1996).
4. R. Sapolsky, "Stress in the Wild," *Scientific American* **262**, 116–123 (1990).
5. H. H. Clark, *Using Language*, Cambridge University Press, Cambridge, UK (1996).
6. B. Reeves and C. Nass, *The Media Equation*, Cambridge University Press, Cambridge, UK (1996).
7. P. P. Maglio, T. Matlock, S. J. Gould, D. Koons, and C. S. Campbell, "On Understanding Discourse in Human-Computer Interaction," *Proceedings of the Twenty-fourth Annual Conference of the Cognitive Science Society*, LEA, Mahwah, NJ (2002).
8. G. Ryle, *The Concept of Mind*, University of Chicago Press, Chicago IL (1984).
9. C. Perrow, *Normal Accidents*, Princeton University Press, Princeton, NJ (1999).
10. D. A. Norman, "Categorization of Action Slips," *Psychological Review* **88**, 1–15 (1981).
11. D. A. Norman, "The 'Problem' with Automation: Inappropriate Feedback and Interaction, Not 'Over-Automation'," *Philosophical Transactions of the Royal Society of London* **B 327**, 585–593 (1990).
12. I. Peterson, *Fatal Defect: Chasing Killer Computer Bugs*, Vintage Books, New York (1996).
13. J. Reason, *Human Error*, Cambridge University Press, Cambridge, UK (1990).
14. L. Bainbridge, "Ironies of Automation," *Automatica* **19**, 775–779 (1983).
15. P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*, IBM Corporation (October 15, 2001); available at http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.
16. G. Lohman, et al., "A DB2 That Manages Itself?," *Proceedings of IDUG 2001: Europe*, Florence, Italy (8–11 October, 2001), International DB2 Users Group, Chicago, IL (2001).
17. P. G. Neumann, "Putting on Your Best Interface," *Communications of the ACM* **34**, No. 3, 138 (March 1991).
18. P. G. Neumann, *Computer Related Risks*, ACM Press, New York (1995).
19. G. I. Rochlin, *Trapped in the Net: The Unanticipated Consequences of Computerization*, Chapter 9, Princeton University Press, Princeton, NJ (1997).
20. P. G. Neumann and M. Jaffe, "Aegis, Vincennes, and the Iranian Airbus," *ACM SIGSOFT Software Engineering Notes* **14**, No. 5, 20–21 (1989).
21. T. Allard, personal communication.
22. J. Reason, *op. cit*., p. 48.
23. David Patterson, personal communication (May 2002).
24. B. Myers, "Challenges of HCI Design and Implementation," *Interactions* **1**, No. 1, 73–83 (January 1994).
25. H. Lieberman, "Letizia: An Agent That Assists Web Browsing," *International Joint Conference on Artificial Intelligence*, AAAI Press (1995), pp. 924–929.
26. P. P. Maglio, R. Barrett, C. S. Campbell, and T. Selker, "An Architecture for Developing Attentive Information Systems," *Knowledge-Based Systems* **14**, 105–112 (2001).
27. B. J. Rhodes, "Margin Notes: Building Contextually Aware Associative Memory," *Proceedings of the Conference on Intelligent User Interfaces (IUI 2000)*, ACM Press, New York (2000).
28. D. Heckerman and E. Horvitz, "Inferring Informational Goals from Free-Text Queries: A Bayesian Approach," *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (1998), pp. 230–237.
29. E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse, "The Lumiere Project: Bayesian User Modeling for

Inferring the Goals and Needs of Software Users," *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (1998), pp. 256–265.
30. F. Linton, D. Joy, and H. Schaefer, "Building User and Expert Models by Long-Term Observation of Application Usage," *Proceedings of the Seventh International Conference on User Modeling*, Springer, London (1999), pp. 129–138.
31. L. Suchman, *Plans and Situated Actions: The Problem of Human-Machine Communication*, Cambridge University Press, Cambridge, UK (1987).

Accepted for publication September 25, 2002.

Daniel M. Russell *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: daniel2@us.ibm.com).* Dr. Russell is the senior manager of the User Sciences and Experience Research (USER) lab at the Almaden Research Center. The main interests of the laboratory are in the areas of designing the complete user experience of computation, especially in the domains of highly sensed or attentive environments, formalizing the characteristics of human behaviors for input mechanisms, and creating new ways of placing computation into the work space. Prior to coming to IBM, Dr. Russell worked at Xerox PARC and in the Advanced Technology Group at Apple Computer Inc. He founded and managed the User Experience Research (UER) groups at both companies. He received his B.S. in computer science from the University of California, Irvine, in 1977, and his M.S. and Ph.D. degrees from the University of Rochester in 1979 and 1984, respectively.

Paul P. Maglio *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: pmaglio@almaden.ibm.com).* Dr. Maglio is a manager and research staff member at the Almaden Research Center. He holds a B.S. degree in computer science and engineering from the Massachusetts Institute of Technology, and a Ph.D. degree in cognitive science from the University of California, San Diego. He joined IBM Research in 1995, where he studies how people think about and use information, among other things.

Rowan Dordick *20 Union Street, Briarcliff Manor, New York 10510 (electronic mail: glifon@aol.com).* Mr. Dordick received a B.S. degree in physics from Marlboro College and an M.A. degree in the history and philosophy of science from Indiana University, where he also completed all the courses and qualifying exams for the Ph.D. degree. He retired from IBM in 2002, having worked in communications for more than 20 years, 12 of which he spent as editor of *Think Research* magazine. He has two patents in the field of human-computer interaction and is a coinventor on a patent disclosure in the area of autonomic computing.

Chalapathy Neti *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: Chalapathy_Neti@us.ibm.com).* Dr. Neti is a manager in the Human Language Technologies Department at the Watson Research Center. He currently manages a research effort on audio-visual speech technologies that is working to develop algorithms and technologies for joint use of audio and visual information for robust speech recognition, speaker recognition, and multimedia content analysis and mining. He has been with IBM since 1990. His main research interests are in the area of perceptual computing (using a variety of sensory information sources to recognize humans, their activity, and their intent), speech recognition, multimodal conversational systems for information interaction, and multimedia content representation for search and retrieval. Dr. Neti received his Ph.D. degree in biomedical engineering from The Johns Hopkins University, his master's degree in electrical engineering from Washington State University, and his bachelor's degree in electrical engineering from the Indian Institute of Technology, Kanpur. He is an associate editor of *IEEE Transactions on Multimedia*.