

A Lightweight Framework for Cross-Application User Monitoring



The human-computer interaction community has often focused on the face applications present to the user. The authors present a new framework that monitors higher-level events to learn how people access, create and modify information rather than how they use applications.

*Kurt D.
Fenstermacher*
*Mark
Ginsburg*
University of Arizona

People access information in many ways—listening to the radio, watching television, speaking with others—but at work they rely primarily on the computer. Users often shift among applications as they seek needed data, integrating new material into e-mail messages, memos, and other documents. For example, a manager might begin writing a memo in a word processor, incorporate budget estimates from a spreadsheet, update the projections after reading a colleague's e-mail, browse the Web for background facts, and finally send the document to other team members. To date, however, no automated means has been available to track the flow of information through such multiapplication tasks. The only record of this activity is a series of newly created files—the “connectedness” among the various tasks is lost.

We propose a lightweight framework for rich monitoring of multiapplication client sessions that captures cross-application activity—metaphorically, in full-motion video rather than as static snapshots. This framework enables complex analyses of how users access and create information, even when they do so using several applications in concert.

APPLICATION MONITORING

Many researchers have developed application-monitoring systems to facilitate the study of appli-

cation usage. In a recent survey of the state of the art, David Hilbert and David F. Redmiles¹ classify and describe a broad range of systems. In general, these systems track user interface events through the system's window manager or within an individual application. Capturing key presses, mouse clicks, mouse drags, and other events offers a glimpse of the interface at the lowest level. Human-computer interaction (HCI) experts can use such information to identify unused features or to recognize inefficient action sequences by novice users.

For such systems, however, transforming low-level user interface (UI) event streams into high-level, semantically meaningful events is often difficult. For example, a cut-and-paste operation that transfers an image from one document to another can generate an extended event sequence with dozens of mouse movements, key presses, and clicks. As Hilbert and Redmiles observe, interpreting abstract events in context requires real-time transformation so that the system can recognize the event before the surrounding information is lost. Our goal is to develop a lightweight framework to study information access and usage in context within and across applications and to do so at a more abstract level than the UI events themselves.

Although we focus on instrumenting Windows applications to follow information flows, we also advocate redesigning applications to better accom-

**Supporting
ad hoc groups
would be a valuable
additional
knowledge-
management tool.**

modate information flows—a special case of traditional HCI research that aims to improve elements of the user interface.

If, for example, a group is working on a shared editing project and members are consulting numerous information sources during a synchronous or asynchronous task, a richer framework for user monitoring makes possible real-time granular analysis of their knowledge work. This analysis enables real-time changes to the user interface. Supporting ad hoc groups—which constantly form and reform across the enterprise, crossing formalized hierarchical bounds²— would be a valuable additional knowledge-management tool.

Today, much analysis is based on window manager UI events or, in the case of the Web, incomplete information stored in server log files. Thus, our understanding of the worker's interaction in this information- and feature-rich environment is sketchy.

Our proposed client-centric framework provides the infrastructure for powerful new techniques to assist individual and group knowledge tasks. We can apply our framework to build understanding in two major ways: monitoring information resource usage and analyzing how people use applications and tools.

Managing information resources

Organizations make large capital investments in computer infrastructure and productivity tools for knowledge workers. In addition, they maintain extensive internal databases and spreadsheets as well as subscriptions to external information sources such as Seybold or Gartner reports. Without awareness of user activities, however, corporations cannot accurately gauge the costs and benefits of these expenditures. Thus, companies should be strategically motivated to determine, for example, whether employees are accessing a subscription service and, if so, whether they are using the information they find constructively.

Academic settings have related concerns. Discovering which Web sites students are using, determining whether they are using online documents to construct word-processed reports, and identifying other such information can offer insight into reuse patterns and provide a more sophisticated basis for detecting plagiarism. Academic librarians could also build better navigation pathways to databases and other information sources that students find useful.

Client monitoring offers more granular data about Web trajectories and other desktop productivity applications. Combining detailed Web and

application usage patterns can illuminate an information source's usefulness for particular tasks. Thus, a Web page has high utility in a corporate setting if many workers copy sections of it to create daily memoranda; a Web page with high activity but low utility might be a splash page—the cover image and blurb that intranet users see when they start their browsers.

Uncovering specific data regarding the utility of Web pages, databases, vendor-provided subscription reports, and other sources will let corporate information managers and academic librarians achieve one of business computing's key measures: quantitative metrics to order the relative worth of information resources, both inside and outside the organization.

Monitoring application usage

While application usability has been studied extensively, we focus on the flow of information among applications. Suppose, for example, a user begins a new document and then switches to a Web browser to search for information on a competitor. The author may conduct a Web search, review results, copy a quote from a news story regarding a recent product offering into the word processor, and e-mail the finished report to a colleague.

One of our goals is to recognize such connected actions and identify opportunities for modifying application interfaces. In the above case, designers might conclude that it would be beneficial to incorporate Web searching more directly into word processors. InfoLab's Watson project (<http://dent.infolab.nwu.edu/infolab/projects/project.asp?ID=5>) does just this, offering a dynamic list of relevant Web pages based on the currently edited text in Microsoft Word.³ In addition, a system could automatically attach a list of Web site visits categorized by level of interest as inferred from time spent on a page, links followed from a page, which pages were copied from, and other factors.

Given the importance of Web-accessible resources, studying the interrelationship among productivity applications and the Internet is fertile ground. The framework we present aims to fill a gap between past HCI work on interface design within an application and recent research on Web mining.

GOALS

Building on prior research in monitoring application usage,¹ we have designed the framework to automate data collection in an unobtrusive manner and support multiple applications. We have also added two more goals: a balance of abstrac-

tion and inference, and ease of adoption by developers. Each of these goals has design implications.

Unobtrusiveness

Users who are required to perform additional actions to support monitoring will invariably seek ways to lessen the burden, thereby distorting the monitoring results. Monitoring should therefore be as unobtrusive as possible to users. Transparent monitoring in the workplace, however, presents ethical dilemmas. For example, an employee who deletes an embarrassing e-mail before sending it could be surprised to learn that the message appears in the tracking database. It is not only possible to record every Web site visited, e-mail sent, and document created, but monitors can create copies of such records that are beyond the user's control.

In other environments, such as university computer labs, user access probably should not be monitored without at least implied, if not explicit, consent. In such cases, cryptographic techniques can preserve data integrity while breaking the data's link with a known person.

Of course, unobtrusiveness must also be balanced against data-gathering needs. For example, requiring that users thoughtfully comment on every action they take—from examining a link but choosing not to follow it to copying text and images from a Web page and pasting them into a document—would generate rich data but is clearly untenable in the long term.

Cross-application integration

Traditionally, usability studies employing automated monitoring systems have focused on the design of a single application or the consistency of design across related applications, such as productivity suites. In tracking the flow of information access and creation, however, following the trail of information “bread crumbs” from one application to the next is important. In addition, interprocess communication mechanisms, such as system clipboards, do not correspond to unique applications. Thus, a framework for studying information access must support capturing flows across applications.

Cross-application logging could then support new inferences about user behavior. Consider, for example, a person who is using a word processor to compose a memo and a Web browser to search for background information to copy into the document. Without knowing what the user copies and pastes, there is no way to distinguish among all the pages visited. Incorporating the word processor into the monitoring framework, however, enables

client-side data mining to reveal the most frequently copied Web sources for a particular user.

Balancing abstraction and inference

A key challenge facing any automated data-collection system is inferring events of interest from the recorded data. Recording low-level events supports low-level inferences—for example, by tracking mouse movements, designers can identify common command sequences that require extended mouse travel. Low-level event recording leads to new problems, however. Hilbert and Redmiles¹ point out that transformation of the UI event stream—which incorporates abstraction—is a key challenge in drawing conclusions from monitoring systems. A monitoring framework should thus balance the granularity of data collection and the needed inference level.

Ease of adoption

Existing systems for application monitoring are complex, offering as many options as the applications they monitor. Although such systems present powerful insights into usability, adopting them requires studying user behavior. Developers must be able to easily adopt the framework to create their own monitoring systems. Our framework is thus lightweight both in its implementation burden for developers and its overhead for applications users.

FRAMEWORK FOR USER MONITORING

The tight integration required between a monitoring framework and user applications precludes using a common framework for multiple platforms. A solution based on the widely adopted Microsoft Windows operating system, however, would address the needs of many users. Our framework combines Microsoft's Component Object Model technology, Internet Explorer, Microsoft Office suite applications, and Python (<http://www.python.org/>),⁴ an open source programming language.

COM

Component Object Model provides the infrastructure that supports our monitoring framework.⁵ Most of what users can do through an application's user interface, programmers can do through COM. With COM, Windows programs can control, as well as receive event notifications from, applications.

By setting properties and invoking methods, developers can use COM to control the applica-

Cryptographic techniques can preserve data integrity while breaking the data's link with a known person.

COM and Binary Reuse

Most IT professionals are familiar with today's silver bullet: object-oriented programming. OO emphasizes reuse—instead of creating what you need from scratch, incorporate another's work. Most OO languages, including C++ and Java, include features to ease the burden of reusing another programmer's code.

Source code reuse

Adapting existing code to fit a new problem often requires examining the code to make the appropriate changes. The problem with this *source-code reuse* is that there are as many versions of the code as there are programmers working on it. The promise of reuse—that a community of developers will create a set of interchangeable components everyone uses—clashes with the need to adapt past solutions to today's problems.

Binary reuse

Microsoft's Component Object Model (COM) embodies *binary reuse*, an alternative philosophy in which programmers reuse past solutions already implemented in existing applications or components. With binary reuse, programmers cannot alter past implementations because they do not have access to the source code itself. Although this approach sacrifices flexibility for consistency, the popularity of Visual Basic demonstrates its success. VB programs are often simply short segments of glue code that rely on COM to control complex objects.

Automation and hosting

COM enables programmers to either reuse an application as a stand-alone component (*automation*) or incorporate one application's functionality into another (*hosting*). Automation involves a client requesting a separate application—the COM server—to perform some action. For example, a word processor asks a browser to preview a Web document it has edited in the word processor. With hosting, a client incorporates a server into the application itself. Internet Explorer users see hosting in action when they click on an Adobe Acrobat document—a version of Acrobat Reader appears inside Explorer, and users can manipulate the document as if they had opened it in a separate Acrobat Reader window.

tions through a second channel while users continue to use them normally.

The “COM and Binary Reuse” sidebar discusses the attractiveness of using COM for our framework. Through COM, most applications offer three categories of interaction:

- *properties*, or attributes;
- *methods*, actions the object can perform; and
- *events*, notifications that something interesting has happened.

For example, the Web browser COM object has a `Visible` property, which determines whether Internet Explorer appears among the running applications on the desktop. The Web browser control responds to the `Navigate2(URL)` method by loading the Web page at the given URL. COM signals a `A.StatusTextChanged` event whenever the

browser's status text is updated—typically because a user has moved the cursor over an HTML link.

Windows clipboard

A cross-application framework must monitor interprocess communication as well as the applications themselves. On Windows, the system clipboard is the most common mechanism for users to transfer information among applications as well as within applications. However, because the clipboard itself is not a COM object, integrating it requires additional effort. Instead, the clipboard is accessed through Microsoft's Win32 API, using Windows's messaging framework.

In particular, Windows sends a `WM_DRAWCLIPBOARD` message to all applications that have requested notification of a change in clipboard data. By requesting such notification, monitoring code can determine when the clipboard has been updated. Although the clipboard can only store a single item, it can do so in multiple formats. For example, if a user copies an image from a Web page, the browser can store both the image itself and its “ALT” text on the clipboard. The clipboard API enables developers to query available formats for the current data item and select the format in which they wish to retrieve it.

Python

A COM-based framework allows many choices of development tools. Perhaps best known is Microsoft's Visual Studio, which includes Visual C++ and Visual Basic. Other vendors offer COM-capable development tools, such as Borland's popular Delphi. Why choose Python then?

Python is a high-level interpreted language that combines features of a scripting language with support for object-oriented programming, exception handling, garbage collection, and interactive development. It offers the flexibility to quickly build and test systems while being easy to learn. In fact, Python's syntax is simpler than either Java or C++, making it easy for programmers with experience in either language to switch.

Although Java's popularity ensures widespread support and a knowledgeable pool of programmers, COM access is more complex through Java than Python. Java also lacks Python's interactive and dynamic nature, slowing development time. Finally, Java and Python are integrated in Jython, an implementation of Python in Java rather than the standard implementation in C. Jython offers most of Python's features as well as easy integration of existing Java code.

Table 1. Selected events for Internet Explorer, Microsoft Word, and the Windows clipboard.

Application	Event	Event fires when . . .
Internet Explorer	NavigateComplete2	Explorer loads new page; includes page's URL
	StatusTextChanged	Status text (displayed in lower left of window frame) changes— notably when user moves cursor over or off a hyperlink
	Quit	User closes browser window
Microsoft Word	DocumentChange	New document is created, an existing document is opened, or active document changes
	WindowSelectionChange	Selection changes in active document window; used to detect passing of text or objects into document
	Quit	User quits Word
Windows clipboard	WM_DRAWCLIPBOARD (message)	Clipboard content changes, usually when user performs a copy or cut operation

Today's programming languages are in part judged by the standard library support available. Like Java, which has a dizzying array of classes to support database access, Web programming, graphical interfaces, and more, Python includes extensive library support but often at a higher level of abstraction. For example, the Python module `SimpleHTTPServer` already defines a complete Web server, enabling programmers to start a simple Web server with a single line of Python code. In particular, the Python library includes support for programming with the Extensible Markup Language (XML) and, via Jython, access to both standard Java class libraries and other projects' custom Java libraries.

Interface

Although Python per se does not support COM, freely available add-on modules do.⁶ Win32 extensions let developers access the full range of COM capabilities. For example, the following snippet of Python code starts Internet Explorer, loads the IEEE's home page, and makes the application visible to the user:

```
from win32com.client import
    Dispatch

ieBrowser = Dispatch("Internet
    Explorer.Application")
ieBrowser.Navigate2("http://www.
    iee.org/")
ieBrowser.Visible = 1
```

Reacting to events is more complicated because it requires that the developer first define event handlers and then notify Python which ones to use when starting an application. With the Win32 extensions, the developer first defines a class with methods to handle the events of interest and then starts the desired application, specifying the Python class.

Monitoring user actions through application-generated events reveals one of COM's weaknesses:

lack of standardization in application interfaces. An architectural goal for a monitoring framework is a uniform monitoring structure, but the COM standard only describes how COM objects interact with one another—not what interactions they are capable of.

Support from vendors other than Microsoft varies, but many popular applications, including Microsoft's Office suite, provide excellent access through COM. We believe that the ability to control applications as well as the facility to react to user actions makes COM appropriate for developing a monitoring framework on the Windows platform.

ARCHITECTURE

Client-side monitoring of user interactions requires specifying which applications to monitor and what events to track. For example, a developer might want to know when a word-processor user pastes an image into a document. For each tracked event, developers must further indicate what actions to take—generally, log the action and associated data. A Web-monitoring developer thus might wish to record a trace of the user's page views in a log file for later processing.

In our framework, the developer decides which applications to monitor, reviews the COM events the application triggers, and writes event handlers that encode the actions to take upon the event trigger. Monitoring is initiated when the user starts the Python script, which in turn begins the application while continuing to run the given event handlers.

MONITORING USER BEHAVIOR

A developer wishing to monitor user behavior in Internet Explorer would review the COM events the application supports and select those that capture the desired information. Table 1 briefly describes some Explorer events used in our current work.

One challenge in applying the framework is matching available events and properties to needed data. For example, Explorer does not expose a link hover event that fires when the user lingers over a

Table 2. Flat-file representation of partial user log file.

Step	Time stamp	Application	Event	Parameter	Copy from	Paste to
1	1/3/02 9:12:30	Internet Explorer	NavigateComplete2	http://www.stern.nyu.edu/networks/		
2	1/3/02 9:13:22	Internet Explorer	NavigateComplete2	http://www.washingtonpost.com/wp-dyn/business/specials/microsofttrial/timeline/		
3	1/3/02 9:14:44	Microsoft Word	DocumentChange	Template.doc		
4	1/3/02 9:16:50	Windows Clipboard	WM_DRAWCLIPBOARD (message)	<hash digest of the copied selection>	Microsoft Word/ Internet Explorer	Clipboard
5	1/3/02 9:18:22	Internet Explorer	NavigateComplete2	http://www.stern.nyu.edu/networks/quotes/quotes.html		
6	1/3/02 9:22:17	Microsoft Word	WindowSelectionChange	<hash digest of the pasted selection>	Clipboard	Microsoft Word

monitoring applications—such as intelligent assistants, personified by Microsoft Office’s much-maligned animated paper clip—already exist, but the real potential lies in mining recorded data. Behavior patterns, such as choosing between two links, can offer new insights into application usage and, even more important, how people use applications together. Monitoring many applications ensures rich data collection to support user behavior analysis.

INTEGRATED APPLICATION MONITORING

Our framework can support any application with a COM interface. Although the COM specification states *how* applications should interact with one another, it does not specify *what* they should say. Thus, there is no uniform set of interfaces that all applications or even categories of applications support. For example, one word processor exposes a `DocumentBeforePrint` event, another uses `BeforeDocumentPrint` to expose the event, and a third chooses not to expose such an event at all. Nevertheless, some applications—particularly Microsoft Office suite applications—do offer extensive access to their capabilities.

Integrated application monitoring is more complex for two reasons. First, it involves managing data that multiple user applications generate. Second, for Web monitoring to benefit from better integration, developers must track the relationships among these applications. For example, suppose that a researcher writing a journal article shifts from a word processor to a Web browser to look up a citation on the NEC Research Institute’s ResearchIndex Web site (<http://citeseer.nj.nec.com/>). After browsing the results, the researcher copies several references and pastes them into the article one at a time. The key to this pattern is the repeated shift between the two applications, strengthened by their copy-and-paste relationship. Examining the text the researcher is editing at the time could further narrow the connection.

Information access, use, and creation

Another example is an intranet setting in which a researcher is studying issues related to network economics in the Microsoft antitrust case. His goal is to transform a preexisting outline into a finished white paper—in this case, an advancement of the document “token”⁷ passed on for further editing or routing in an arbitrarily long corporate workflow.

The “Cross-Application Monitoring and Document Management” sidebar describes how our framework uses a versioning process that increases the relative worth of archived documents by iteratively modifying their content. This process consists of three basic phases. In the *access* phase, the researcher identifies useful archives, such as databases and document repositories, and selects individual items for retrieval. After browsing the material, he chooses phrases from a Web page to paraphrase, quote verbatim, or support original opinions in the *use* phase. In the final *creation* phase, he alters the draft based on knowledge obtained during the access and use phases.

These phases can all be inferred from the log files that our framework generates. Table 2 is a flat-file representation of the first six steps in a user activity log—we use an XML document-type definition to generate output.

In Step 1, the researcher visits Nicholas Economides’s Web site, a good starting point for literature on the economics of networks. In Step 2, he clicks through to a *Washington Post* timeline article, and in Step 3 he opens the `Template.doc` file, his document skeleton. Step 4 is a copy event—the framework traps the clipboard message `WM_DRAWCLIPBOARD` and invokes `GetClipboardOwner` to retrieve the window handle of the clipboard owner, in this case Internet Explorer. The framework also records a hash digest of the copied selection. In Step 5, the researcher navigates to a set of quotations about the antitrust case. Step 6 is a paste event that the framework intercepts on Microsoft Word; the framework matches the pasted mater-

By associating information sources with tasks and processes and monitoring user actions, the system can offer task-specific help.

ial's hash digest to that of the selection copied in Step 4.

Thus, Steps 1, 2, and 5 illustrate information access; Step 4 shows information use; and Step 6 is an example of information creation. The actual log file is substantially more complex in width (attributes) as well as depth. Each session contains many records because the framework traps all Word/Explorer `Status-TextChange` events, which occur frequently as the user moves the cursor over or off hyperlinks, as well as all Word `DocumentChange` events.

Group work analysis

Our monitoring framework is also well suited for group work analysis. Extending our example, assume that four geographically distributed economists will share editing duties on the document token described above. They face the typical distributed-work obstacles of infrequent communication and incomplete resource discovery—valuable information resources found by one worker may go unnoticed by the others.

Our system can help facilitate the group task: It collects individual log files as above but stores them in a network-accessible location. It then polls the client machines at intervals, or relies on client push, to stream the log data to a central server, where it can analyze group activity to uncover interesting behaviors. For example, if several members are heavily using a certain Internet document, evidenced by numerous copy-and-paste actions, the system can establish mechanisms using e-mail or more immediate interface changes to notify all workers. It might be helpful to distinguish and inform the group when an interesting intranet resource—for example, an internal Domino database—has been discovered to supplement general Internet research. In fact, process discovery on the server side to assist groups and optimize the corporate information environment is a full research endeavor in its own right.

PROACTIVE INFORMATION SYSTEMS

Inferring connections among integrated applications or tasks primarily relies on educated guesswork. We are currently exploring the use of a larger COM-based monitoring framework to index into a library of known process descriptions that incorporate information about applications and how they are used. Such a framework would go beyond passive monitoring to proactively assisting the user based on real-time pattern analysis. Once a process is recognized, the system can drive other applications.

The central notion of process-oriented knowledge delivery is that by associating information sources with tasks and processes and monitoring user actions to infer generic processes, the system can offer task-specific help to users. Our recent efforts have focused on building a framework that lets developers define processes, annotate them with metadata, and integrate off-the-shelf applications through a process-aware engine.

A current demonstration project assists with restaurant reservations by monitoring actions of Web users who access known restaurant recommendation sites such as Zagat Survey (<http://www.zagat.com/>) or city dining guides in Citysearch (<http://www.citysearch.com/>). Once the system notes a user's visit to a review site, it proposes other sites with similar coverage. After the user selects a particular restaurant, the system uses the browser's document object model representation to parse the page, extracting relevant information such as restaurant name, address, and phone number. The system then offers to verify the phone number, retrieve a map and directions, and notify others via e-mail about the time and date once a reservation is made.

The reservation assistant relies on Internet Explorer's `NavigateComplete2` event to infer when a user is engaged in a reservation process. When the process is read in at browser startup, the assistant generates a table of review sites and their URLs. As the user browses the Web, generating `NavigateComplete2` events, it passes the URLs to the event handler to index into the table and looks for a match to a known restaurant review site. If it finds a match, the assistant uses COM's automation capabilities to continue the reservation process on the user's behalf.

Tracking mouse clicks and movements as well as keystrokes offers interface designers powerful insight into how people use applications rather than how designers anticipate their use. Extending such systems to monitor information flow through documents and across applications is challenging. Our framework monitors higher-level events to learn how people access, create, and modify information rather than how they use the applications themselves.

Although the framework is lightweight and simple for programmers to adapt to their needs, monitoring user events nevertheless requires some programming skill. To ease this burden, we intend to introduce a toolkit that lets nonprogrammers select applications and events they wish to monitor and choose actions to take.

In addition to incorporating more productivity tools into the monitoring framework, we plan to perform server-based pattern analysis on client-activity logs to provide feedback to clients as they interact with the Web and other productivity tools, thus providing adaptive user interfaces within workgroups as they undertake knowledge tasks. Our overall goal is to implement a client-monitoring framework with as broad a reach and as simple a configuration interface as possible. ■

References

1. D. Hilbert and D.F. Redmiles, "Extracting Usability Information from User Interface Events," *ACM Computing Surveys*, Dec. 2000, pp. 384-421.
2. I. Nonaka, "A Dynamic Theory of Organizational Knowledge Creation," *Organization Science*, Feb. 1994, pp. 14-37.
3. J. Budzik, K.J. Hammond, and L. Birnbaum, "Information Access in Context," *Knowledge-Based Systems*, Mar. 2001, pp. 37-53.
4. D.M. Beazley, *Python Essential Reference*, 2nd ed., New Riders Publishing, Indianapolis, Ind., 2001.
5. D. Box, *Essential COM*, Addison-Wesley Professional, Boston, 1998.
6. M. Hammond and A. Robinson, *Python: Programming on Win32*, O'Reilly & Associates, Sebastopol, Calif., 2001.
7. W. Weitz, "SGML Nets: Integrating Document and Workflow Modeling," *Proc. 31st Ann. Hawaii Int'l Conf. System Sciences (HICSS 98)*, vol. 2, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 185-194.

Kurt D. Fenstermacher is an assistant professor of management information systems and computer science and associate director of the Hoffman E-Commerce Laboratory at the University of Arizona. He is currently researching the application of artificial intelligence techniques to problems in knowledge and information management, with an emphasis on process-based approaches. Fenstermacher received a PhD in artificial intelligence from the University of Chicago. He is a member of the IEEE, the ACM, the AAAI, and Computer Professionals for Social Responsibility. Contact him at KurtF@Eller.Arizona.edu.

Mark Ginsburg is an assistant professor of management information systems at the University of Arizona. His research interests include social and technical aspects of virtual community platforms and collaborative systems to support document and knowledge management. Ginsburg received a PhD in information systems from New York University. He is a member of the IEEE Computer Society, the ACM, and the Association for Information Systems. Contact him at Mark@Eller.Arizona.edu.