Experience with Evaluating Human-Assisted Recovery Processes

Aaron B. Brown, Leonard Chung, William Kakes, Calvin Ling, and David A. Patterson
EECS Computer Science Division, University of California, Berkeley
Contact author: abrown@cs.berkeley.edu

Abstract

We describe an approach to quantitatively evaluating human-assisted failure-recovery tools and processes in the environment of modern Internet- and enterprise-class server systems. Our approach can quantify the dependability impact of a single recovery system, and also enables comparisons between different recovery approaches. The approach combines aspects of dependability benchmarking with human user studies, incorporating human participants in the system evaluations yet still producing typical dependability-related metrics as results. We illustrate our methodology via a case study of a system-wide undo/redo recovery tool for e-mail services; our approach is able to expose the dependability benefits of the tool as well as point out areas where its behavior could use improvement.

1. Introduction

Human operators play a key role in the dependability of modern server systems. In particular, they often drive the recovery processes that restore quality of service after system failures. While there are ongoing efforts to automate recovery [4] [7], today's reality is that human operators are a fundamental part of any large system's dependability strategy. It is therefore crucial that systems provide tools to help operators restore system behavior after dependability has been compromised. It is likewise crucial that we have a way to evaluate those tools: just as benchmarks from SPEC and TPC have driven CPU and database performance to dizzying heights, benchmarks for human-assisted recovery could encourage significant advances in recovery, and hence dependability.

In this paper, we present the first steps toward building a benchmark for human-assisted recovery processes and tools. Our approach is not yet a complete benchmark, as it does not address some of the practical concerns of benchmarking like distilling results to a single number, nor has it been validated across a wide range of recovery systems. But we will address the central challenge of a benchmark for human-assisted recovery: how to design trials that can quantify the impact of a human-driven recovery process on dependability-related metrics like availability, correctness, and performance. We will illustrate our approach through a case study of a system-wide recovery tool designed explicitly for use by the operators of Internet and enterprise services [2].

Our approach uses different techniques than those found in existing performance or dependability benchmarks [1] [6] [9] [13], even dependability benchmarks that incorporate human operator error into the faultload [14] [15]. Our evaluations are performed using actual human participants, much like one might see in an HCI usability evaluation study, although we retain the structure of a dependability benchmark by focusing on system-level metrics; we treat the human as a black-box perturbation to the system, similar to an injected error.

Using human participants is unavoidable, since operator-assisted recovery is by nature an interactive process, involving repeated cycles of problem diagnosis, repair plan formulation and execution, and testing. Were we to not use human participants, we would have to simulate their behavior during this process, a goal equivalent to replacing human system management with automation and one that is well beyond the current state of the art.

Involving humans increases the cost of an evaluation and raises concerns of variability. Thus much of our approach is focused on reducing cost and controlling variance. In doing so, we will have to make compromises that sacrifice the ultimate representativeness of our results. Such compromises are the bread-and-butter of benchmark design; the art of creating benchmarks lies in balancing the conflict between practicality and the desire to produce results that capture the full complexity of real environments.

We begin addressing this balance in Section 2, where we summarize our evaluation methodology and describe the compromises we chose to make. We then turn to a case study that illustrates our approach: Section 3 describes the experimental setup for evaluating a system-wide undo/redo recovery tool that we have built, and Section 4 presents the results of that analysis. We discuss related work in Section 5, and conclude in Section 6.

2. Methodology, challenges & compromises

Our approach to evaluating human-driven recovery is based on the methodology of a traditional dependability benchmark [1] [9] [6], with the exception that we include the human system operator as part of the system under test (SUT) and therefore require repeated experiments and an additional phase of human subject recruiting and training. The methodology consists of 4 phases:

1. **faultload development:** a selection of injectable errors and faults is chosen to mimic those likely to be seen in practical deployments of the SUT.
2. **workload and metric selection:** a simulated end-user workload is developed to apply to the SUT. This workload is most easily defined by reusing the workload specification of an existing domain-specific performance benchmark. Associated with the workload are metrics that quantify the SUT's quality of response: performance, availability, correctness, and so on.
3. **participant selection and training:** human subjects must be recruited to act as system operators during the evaluation experiments. This involves choosing a representative subject pool, recruiting from that pool, and screening and training recruited subjects to minimize variability in their background, experience, and skills.
4. **experiments:** the evaluation process consists of multiple experiments, each involving one human participant and taking the form of a human trial. The participant plays an active role in maintaining the SUT as it is subjected to the faultload and workload developed in the first two phases. The results of each experiment consist of the time-varying quality metrics collected by the workload generator.

The methodology as presented poses several difficult practical challenges, including faultload selection, participant selection and training, and experiment design. We have discussed these challenges in depth in previous work [3]; due to space constraints, we do not reproduce that analysis here. Instead, we enumerate our practical solutions to the challenges and identify the benchmark compromises that they embody.

Faultload development. As human operator error is the most significant source of failures in Internet and enterprise services [11], the most representative faultload would come from an ethnographic study of operator behavior across large deployed installations of the SUT. Due to the cost and impracticality of this approach, we compromise and produce the faultload by *surveying* operators of deployed systems in the SUT's application domain. Surveys are inexpensive and have the benefit of being based on real-world data, but suffer from bias due to self-reporting.

Participant selection and training. Ideally, one would select human participants from the same pool of operators that would manage deployed installations of the system under test. Since this is often impractical, a compromise approach is to use a subject pool consisting of people with reasonably-equivalent skill levels. To salvage some representativeness from the resulting less-than-representative subject pool, screening and training can be used. Screening filters subjects to ensure a minimum level of knowledge of systems operation tasks and familiarity with the application domain of the SUT. Training bolsters this background with SUT-specific knowledge.

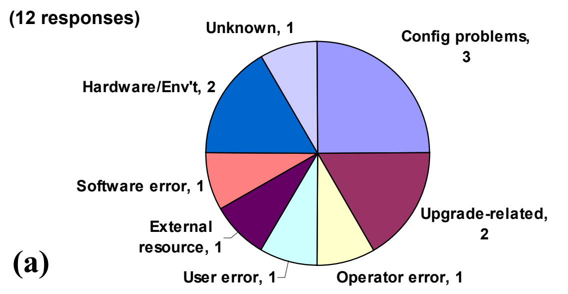
Experiment design. Most recovery evaluations will pit a supposedly-improved recovery tool or process against a baseline system; cross-system comparison benchmarks are a special case of this type of evaluation. A standard experiment design for such cases is to conduct a randomized trial with SUTs and faultload cases assigned randomly to each human participant. While simple, this basic format is appropriate only for very homogeneous or large participant pools, where the inherent variability between participants can be averaged out.

A compromise that makes the experiments more practical is to compare each subject only to himself, allowing the experimenter to draw independent conclusions for each subject as to whether a particular SUT's recovery process is more effective than another's. Such an experiment design is achieved by having each subject perform recovery for the same injected fault test case in two or more back-to-back trials, each involving a different SUTs. There is a danger of *learning bias*, where a subject learns information in the first trial that helps in the second. Randomization can average out this bias, but again requires large subject pools. A further practical compromise for the special case of comparing a new recovery mechanism against a baseline is to leave this bias in: by always performing the baseline trial second, the bias becomes systematic, and a positive conclusion can still be drawn from the evaluation if the improved system demonstrates better dependability than the baseline. Only if the baseline outperforms the improved system is a more complex randomized trial needed.

3. Case study setup: evaluating Undo/Redo

We now illustrate our approach to evaluating human-assisted recovery processes by applying it to an *Undo/Redo* tool designed explicitly for human-driven failure recovery in Internet and enterprise e-mail server systems. The Undo/Redo tool, described in depth in prior work [2], allows human operators to *retroactively repair* any effects that a failure might have had on an e-mail service's hard state, even when the root cause of the failure is unknown. The tool proxies end-user interactions and presents a time-travel interface to the operator, who can use it to roll back (undo) system state to a known-good point, then repair it and roll forward (redo) all intervening end-user work; the redo step reprocesses logged user interactions (such as reading/filing e-mail) in the context of the repaired system.

Our goal in evaluating the Undo/Redo tool was to determine if it could improve the end-user-perceived dependability of an e-mail store service as compared to a version of the same service without the tool. We focused on two aspects of dependability, correctness and availability. We define an e-mail server to be correct if it properly delivers all messages it receives and properly performs all user-requested operations that it acknowledges. We define availability at a network protocol level: an e-mail server is available if it accepts SMTP/IMAP connections and completes their protocol dialogues.



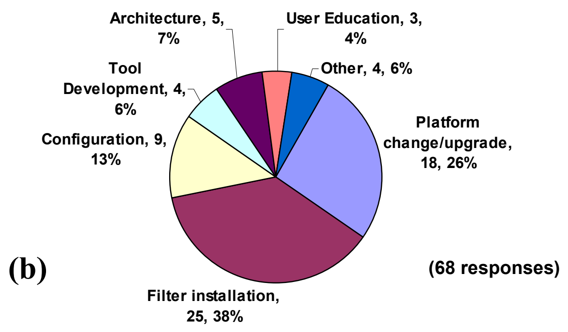


Figure 1. Survey of e-mail administrators. The graphs show the breakdown of responses given by practicing e-mail system administrators when asked to describe (a) the causes for actual situations where e-mail was lost, and (b) challenging e-mail management tasks they had performed recently. The data suggest that a benchmark faultload for e-mail should include configuration errors involving SPAM filters, upgrades, simple restart-repairs, and hardware failures.

Faultload selection. We used the survey approach described above to select a faultload. We developed a web-based survey and e-mailed a request for participation to the mailing list of SAGE, a membership organization dedicated to the profession of system administration.[1] We received 68 completed surveys, from which we manually extracted and categorized descriptions of 68 challenging e-mail management tasks and 12 scenarios where e-mail data was lost; Figure 1 shows the results. Our analysis shows that the most common failure scenarios involve configuration errors (typically involving SPAM/virus filtering software), failed upgrades of the e-mail server platform, and hardware or environmental failures. The dominant management challenges involve upgrades, SPAM/virus filter installation, and configuration management. Other survey questions revealed that another significant, but less challenging set of management tasks were simple repairs like restarting crashed server processes.

We chose three distinct failure scenarios, two that capture the dominant failure and management cases identified in the survey (configuration errors and upgrades), and one that captured the simple restart-crashed-server task. We used a cognitive walkthrough procedure [5] to identify the injectable operator errors and software failures needed to reproduce the scenarios on the SUT. The scenarios were:

1. **SPAM filter configuration error:** the injected fault is a mistyped configuration line in the MIMEDefang/SpamAssassin mail filter script. The resulting syntax error causes all incoming e-mail that is less than 200KB in size to be silently rejected.
2. **Failed e-mail software upgrade:** we simulate the error that occurs when the operator forgets to activate the compile-time option needed to enable mail filtering when upgrading Sendmail from version 8.12.9 to 8.12.10. The symptoms of the resulting failure are that once the upgrade is installed, all mail filtering ceases.
3. **Simple software crash:** we simply kill the Sendmail server process to simulate the effects of a software bug. The symptoms of the resulting failure are that no incoming e-mail is accepted by the mail server.

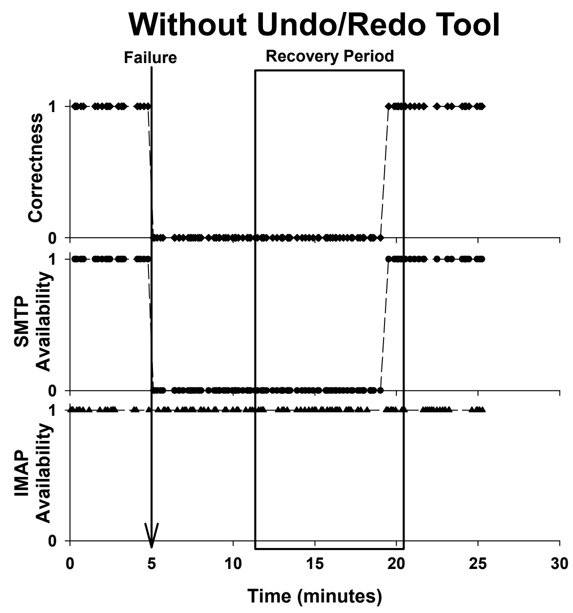
[1] Our survey and other screening and training documents can be found online at http://roc.cs.berkeley.edu/projects/HumanBench/

We expected Undo/Redo recovery to be useful for the first two scenarios, but unnecessary for the third.

Workload selection. We chose an e-mail–specific workload consisting of a stream of simulated incoming e-mail via the SMTP protocol and a stream simulating the actions of users checking mail via the IMAP protocol. Incoming e-mail was generated according to a Poisson process with a rate of 5 messages per minute and randomly-chosen message sizes based on the distribution used by the standard SPECmail2001 e-mail benchmark [12]. Each piece of incoming e-mail was hashed and stamped with a unique ID; at the end of each session the workload generator attempted to retrieve each message to verify whether it had been received, processed correctly, and filtered if appropriate. Simulated user IMAP sessions were also generated using a Poisson arrival process with a rate of 5 sessions per minute; in each session, the simulated user logs in, lists unread messages, randomly retrieves 80% of the new messages, then randomly deletes 10% of those messages.

Participant selection and screening. We recruited participants from the student population of the UC Berkeley computer science department. The typical member of this population is technically-savvy but does not have much experience with e-mail server operations. We screened respondents by asking them to self-report their own system management experience, and also included a skill test in which we tested knowledge of e-mail protocols and services. We received 18 responses to our solicitation, 14 of which met our screening criteria of having at least 60% of the maximum possible self-reported experience and no more than one error on the skill test. Of these 14 respondents, 13 agreed to participate in the recovery-evaluation experiments, and 12 completed the experiments. Participants were offered a $50 gift certificate to online retailer Amazon.com to compensate them for their time.

We trained participants by giving them a set of documents introducing the setup of the e-mail server system and describing the Undo/Redo tool. After reading these at their leisure, participants were encouraged to follow a simple task walkthrough that familiarized them with the e-mail server setup: they were asked to verify that the Sendmail e-mail server was running, to edit one of its configuration files, and to restart it.



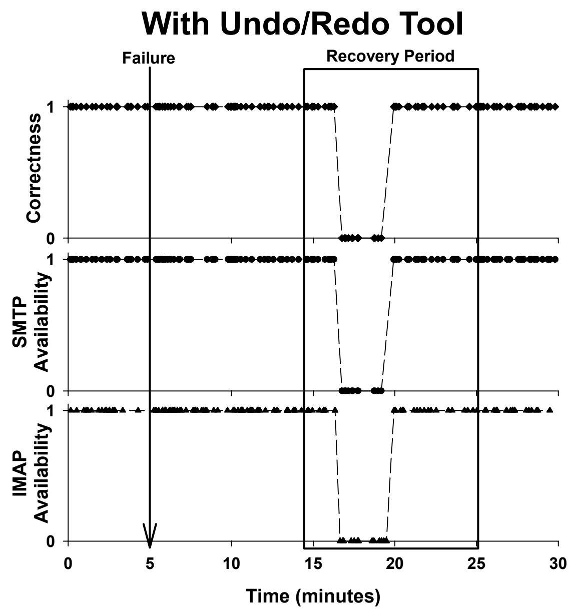


Figure 2. Time-varying behavior during recovery for one experiment. The graphs plot instantaneous correctness and availability over time for an experiment using failure scenario #1 (filter configuration error). The recovery period begins when the participant is informed of the failure, and ends after 30 minutes or when the participant declares success. The system under test is available at time *t* if it accepts network connections at that instant. In contrast, correctness is measured at the end of the experiment: the system is correct at time *t* if a message originally sent at time *t* has been properly received, handled, and stored on disk by the end of the experiment.

They were further given the opportunity to experiment with the Undo/Redo recovery tool and its user interface.

Experiment design. Since our goal was to compare Undo/Redo recovery to a baseline system, we chose the self-comparison design described above, with each human subject performing two consecutive trials on the same failure scenario, using the Undo/Redo-enabled system first and the baseline second. We explicitly focused the experiments on recovery, excluding consideration of problem detection and diagnosis.

In each trial, the participant received a symptom report describing one of the faultload scenarios, and was asked to recover the e-mail system to normal operation. The scenarios were randomly assigned to participants. Each trial had a 30-minute time limit. Participants had access to the Internet, a Sendmail manual, a day-old backup of the system, an e-mail client configured to send test messages, and all standard tools available on the e-mail server system. The Undo/Redo tool was available *only* during the first trial. While participants could ask any questions they wanted during training, we refused to answer questions during the trials, with one exception: each participant was allowed *one* question during each trial, just as in real-life an administrator might appeal to a guru for help. The goal of this "guru" resource was to prevent frustration should the participant get stuck; it was only used three times across the 26 trials we conducted.

System configuration. All of our benchmark sessions used identically-configured e-mail servers. Each server ran Debian Linux 2.4.18, Sendmail 8.12.9 as an SMTP server, UW-IMAP as the IMAP server, and MIMEDefang with SpamAssassin as the system-wide SPAM filters. All e-mail and mailspools were stored on a Network Appliance filer connected via gigabit Ethernet. The workload was generated on a separate machine also connected via gigabit Ethernet. The participants interacted with the e-mail servers via a Windows 2000-based console using ssh and Outlook Express. The console machine had a second video display attached that allowed the experimenter to unobtrusively monitor the participant's progress from a location out of the participant's line of sight.

4. Evaluating Undo/Redo: Results

Our experiment infrastructure produces two different flavors of results: per-participant longitudinal data showing the time-varying behavior of the system as the recovery process takes place, and cross-participant summary data useful for direct system-to-system comparison. The summary data can be used for hypothesis testing—in our case, to validate the hypothesis that Undo/Redo-based recovery improves net dependability—while the longitudinal data provides the details explaining why and how the hypothesis holds. As mentioned above, we focused on metrics of correctness and availability.

Longitudinal data. Figure 2 plots correctness and availability over time for one particular participant's benchmark session. This participant was asked to recover from the first faultload scenario (the misconfigured SPAM filter), and the results are typical of other participants. The two sets of graphs correspond to the two phases of the experiment: the left-hand set shows baseline results, and the right-hand set shows results with the Undo/Redo tool available.

In Figure 2, we can see that the Undo/Redo recovery process significantly improves the e-mail system's correctness under this failure scenario by reducing the number of incorrectly-dropped messages compared to the baseline. Furthermore, the experiment illustrates how the Undo/Redo tool achieves this advantage: it shows that the undo tool is able to retroactively extend its recovery to the point where the fault occurred, whereas baseline recovery can at best only correct errors that occur after recovery begins. Finally, the results point out that, despite its benefits, the Undo/Redo tool still has some weaknesses in terms of overall dependability: unlike non-undo recovery, it causes a temporary drop in IMAP service availability (even when the recovery process is aborted, as revealed by another of our trials), and still allows a number of messages to be handled incorrectly. These problems were anticipated during the design of the prototype undo tool, but the evaluation quantifies their effects and suggests where it might be worth spending more development effort.

Summary data. We aggregated our experimental data by computing the total number of mishandled e-mail messages and the total number of failed attempts to contact the e-mail service for each benchmark session and each participant, starting five minutes before the participant began recovery and ending five minutes after the participant signaled completion. Figure 3 plots these results by participant, graphically showing the comparison between each participant's first session (with Undo/Redo) and second session (the baseline). We have segregated the subjects by the failure scenarios they were given, and have split off the cases where the subjects chose not to use or complete the undo-based recovery process.

Figure 3 clearly shows the improvement in correctness with the Undo/Redo recovery tool: in the 7 cases where the tool was used, the number of incorrectly-handled messages is always less than half of the corresponding result for the baseline system. Because of the systematic bias in our self-comparison experiment design, this data is insufficient to quantify the *degree* of improvement from undo/redo. However, it does support the conclusion that the improvement is statistically significant under a binomial trial model (*p*-value of 0.045). Furthermore, the variance is significantly reduced with undo/redo recovery, indicating that the tool scales well across the expertise range of the participants, and suggesting that it can make effective recovery more accessible.

The 5 cases where Undo/Redo was not used break down into two sets. The first three cases correspond to the scenario of a crashed Sendmail process, for which Undo/Redo is not useful. All of the participants realized this and none attempted to use the Undo/Redo recovery. These cases show only the learning-curve effect, or the systematic bias introduced by our fixed-order experiment design. Finally, the remaining two cases correspond to scenarios where Undo/Redo would have been useful, but where the subject chose not to use the tool.

The results for availability, also shown in Figure 3, illustrate one of the limitations of Undo/Redo recovery. Except for failure scenario #1, Undo/Redo recovery does not offer an availability advantage over baseline recovery, and, especially for IMAP, can actually hurt it. In failure scenario #1, the undo-based recovery data does show an availability benefit; this is a side-effect of the way that the undo tool proxies incoming SMTP sessions, ensuring availability even when the SMTP server itself is misbehaving. In the end, the best conclusion we can draw here is that future work on Undo/Redo recovery should concentrate on improving the system's availability during the recovery process for both IMAP and SMTP protocols, so that the obvious correctness benefits of the approach are not lost as a result of poorer availability.

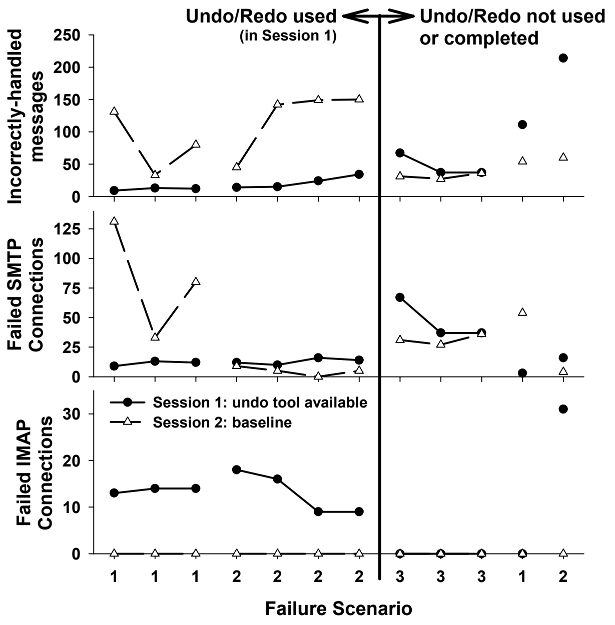


Figure 3. Summary results. The graphs plot overall correctness and availability metrics for all experiment sessions, enabling a direct comparison of Undo/Redo recovery to the baseline. Each point on the *x*-axis represents the results for an individual human participant. The results show a clear correctness benefit when Undo/Redo recovery is used, but also point out that in some cases, Undo/Redo recovery can reduce availability compared to the baseline.

5. Related work

The notion of benchmarking fault tolerance was first introduced by Tsai et al. [13]; their focus was primarily on hardware-level faults and automated recovery mechanisms. Recent work has broadened the applicability of so-called dependability benchmarks [1] [3] [6] [9] [15] [18], but has mostly overlooked the human aspects of server system dependability. A notable exception is work by Vieira and Madeira, who have studied the recovery behavior of database management systems in response to injected operator faults [14] [15]; however, unlike our approach, theirs does not use human participants and therefore can only evaluate recovery mechanisms that work without human involvement. Likewise, Zhu et al. propose benchmarks for evaluating and classifying the recovery behavior of general server platforms in response to crashes and hardware failures [17] [18], but again do not address the human component of the recovery process, assuming that the system is capable of recovering on its own. Furthermore, neither Zhu's nor Vieira's methodology can provide the kind of insight that ours can offer into the dynamic, time-varying dependability consequences of the recovery process.

There is obviously a great deal of similarity between our methodology and behavioral research methodologies used to study human-computer interaction, and indeed our approach was heavily influenced by HCI techniques such as those described in Landauer's excellent survey [8] and used by Maxion et al. to evaluate dependability effects arising from the user interface [10]. However, unlike these HCI approaches, where understanding the human's behavior is the main goal, the focus of our benchmarks is to quantify the *system*, with the human as a critical but indirect contributor to its behavior. In that sense our work is most similar to work in the security community on the effectiveness of security-related UIs, such as Whitten and Tygar's study of PGP [16].

6. Discussion and conclusions

Our approach to evaluating human-assisted recovery processes is a first step toward incorporating the effects of human behavior into a dependability-benchmarking framework. While our approach is still a far cry from a complete benchmark, it demonstrates that it is practical to conduct dependability evaluations using people, and that the cost of such evaluations may well be reduced to the point where a benchmark involving humans becomes feasible. In particular, the compromises—typical of any benchmark design process—that we made in developing our approach did not limit our ability to expose illuminating data regarding the behavior of our Undo/Redo recovery tool. Furthermore, we were able to obtain those insights with only a handful of human participants, an inexpensive survey-based faultload development process, and a non-traditional experimental design that traded a built-in bias for a smaller subject pool.

Still, there is much research to be done in the area of human-aware dependability benchmarks. Our approach needs to be extended to other lifecycle phases besides recovery, such as problem detection and diagnosis. More work is needed to further reduce the cost of our approach, perhaps by teasing apart the benchmark components that truly require human intervention from those that can be adequately simulated. And better understanding of variability and learning effects within subject pools is needed, both to produce more homogenous populations for a benchmark and to understand how reproducible a human-driven benchmark can be. Despite these challenges, we believe that the benefits and significance of human-aware dependability benchmarks are evident, and we look forward to the day when they can be found in every dependability researcher's toolbox.

References

[1] A. B. Brown and D. A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. *2000 USENIX Annual Technical Conference.* San Diego, CA, June 2000.

[2] A. B. Brown and D. A. Patterson. Undo for Operators: Building an Undoable E-mail Store. *2003 USENIX Annual Technical Conference.* San Antonio, TX, June 2003.

[3] A. B. Brown, L. C. Chung, and D. A. Patterson. Including the Human Factor in Dependability Benchmarks. *2002 Workshop on Dependability Benchmarking,* in *DSN 2002 Supplement*. Washington, D.C., June 2003.

[4] A. Fox and D. Patterson. Self-Repairing Computers. *Scientific American*, June 2003.

[5] M. Ivory and M. Hearst. The State of the Art in Automating Usability Evaluation. *ACM Computing Surveys*, 33(4):470–516, December 2001.

[6] K. Kanoun and H. Madeira. A Framework for Dependability Benchmarking. *2002 Workshop on Dependability Benchmarking,* in *DSN 2002 Supplement*. Washington, D.C., June 2003.

[7] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer* 36(1):41–50, 2003.

[8] T. K. Landauer. Research Methods in Human-Computer Interaction. In *Handbook of Human-Computer Interaction, 2e*, M. Helander et al. (ed), Elsevier, 1997, 203–227.

[9] H. Madeira and P. Koopman. Dependability Benchmarking: making choices in an n-dimensional problem space. *1st Workshop on Evaluating and Architecting System dependabilitY (EASY '01),* Göteborg, Sweden, 2001.

[10] R. A. Maxion and A. L. deChambeau. Dependability at the User Interface. *25th Intl. Symp. on Fault-Tolerant Computing*. Pasadena, CA, June 1995.

[11] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? *4th USENIX Symposium on Internet Technologies and Systems (USITS' 03)*. Seattle, WA, March 2003.

[12] Standard Performance Evaluation Corporation. *SPECmail2001*, http://www.spec.org/osg/mail2001/.

[13] T. K. Tsai, R. K. Iyer, and D. Jewitt. An Approach towards Benchmarking of Fault-Tolerant Commercial Systems. *26th Int'l Symp. on Fault-Tolerant Computing (FTCS-26)*. Sendai, Japan, June 1996.

[14] M. Vieira and H. Madeira. Recovery and Performance Balance of a COTS DBMS in the Presence of Operator Faults. *2002 Int'l Conf. on Dependable Systems and Networks*. Washington, D.C., June 2002, 615–624.

[15] M. Vieira and H. Madeira. Definition of Faultloads Based on Operator Faults for DBMS Recovery Benchmarking. *2002 Pacific Rim Int'l Symp. on Dependable Computing (PRDC2002)*. Tsukuba, Japan, 2002.

[16] A. Whitten and J. D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. *9th USENIX Security Symposium*, August 1999.

[17] J. Zhu, J. Mauro, and I. Pramanick. System Recovery Benchmarking. *2002 Workshop on Dependability Benchmarking,* in *DSN 2002 Supplement*. Washington, D.C., June 2003, F-27–28.

[18] J. Zhu, J. Mauro, and I. Pramanick. Robustness Benchmarking for Hardware Maintenance Events. *2003 Int'l Conf. on Dependable Systems and Networks*. San Francisco, CA, June 2003.