Human-Aware Computer System Design [*]

Ricardo Bianchini, Richard P. Martin, Kiran Nagaraja, Thu D. Nguyen, Fábio Oliveira
Department of Computer Science, Rutgers University, Piscataway, NJ 08854
{*ricardob, rmartin, knagaraj, tdnguyen, fabiool*}*@cs.rutgers.edu*

Abstract

In this paper, we argue that human-factors studies are critical in building a wide range of dependable systems. In particular, only with a deep understanding of the causes, types, and likelihoods of human mistakes can we build systems that prevent, hide, or at least tolerate human mistakes by design. We propose several research directions for studying how humans impact availability in the context of Internet services. In addition, we describe validation as one strategy for hiding human mistakes in these systems. Finally, we propose the use of operator, performance, and availability models to guide human actions. We conclude with a call for the systems community to make the human an integral, first-class concern in computer system design.

1 Introduction

As computers permeate all aspects of our lives, a wide range of computer systems must achieve high dependability, including availability, reliability, and security. Unfortunately, few current computer systems can legitimately claim to be highly dependable. Further, many studies over the years have empirically observed that human mistakes are a large source of unavailability in complex systems [7, 13, 15, 16]. We suspect that many security vulnerabilities are also the result of mistakes, but are only aware of one study that touches on this issue [15].

To address human mistakes and reduce operational costs, researchers have recently started to design and implement autonomic systems [9]. Regardless of how successful the autonomic computing effort eventually becomes, humans will always be part of the installation and management of complex computer systems at some level. For example, humans will likely always be responsible for determining a system's overall policies, for addressing any unexpected behaviors or failures, and for upgrading software and hardware. Thus, human mistakes are inevitable.

In this paper, we argue that human mistakes are so common and harmful because computer system designers have consistently failed to consider the human-system interaction explicitly. There are at least two reasons for this state of affairs. First, dependability is often given a lower priority than other concerns, such as time-to-market, system features, performance, and/or cost, during the design and implementation phases. As a result, improvements in dependability come only after observing failures of deployed systems. Indeed, one need not look past the typical desktop to see the results of this approach. Second, understanding human-system interactions is time-consuming and unfamiliar, in that it requires collecting and analyzing behavior data from extensive human-factors experiments.

Given these observations, we further argue that dependability and, in particular, the effect of humans on dependability should become a first-class design concern in complex computer systems. More specifically, we believe that human-factors studies are necessary to identify and understand the causes, types, and likelihoods of human mistakes. By understanding human-system interactions, designers will then be able to build systems to avoid, hide, or tolerate these mistakes, resulting in significant advances in dependability.

In the remainder of the paper, we first briefly consider how designers of safety-critical systems have dealt with the human factor in achieving high dependability. We also touch on some related work. After that, we propose several research directions for studying how human mistakes impact availability in the context of Internet services. We then describe how *validation* can be used to hide mistakes and *guidance* to prevent or at least mitigate the impact of mistakes. Finally, we speculate on how a greater understanding of human mistakes can improve the dependability of other areas of computer systems.

[*]This research was partially supported by NSF grants #EIA-0103722, #EIA-9986046, and #CCR-0100798.

2 Background and Related Work

Given the prominent role of human mistakes in system failures, human-factors studies have long been an important ingredient of engineering safety-critical systems such as air traffic and flight control systems, e.g., [6, 18]. In these domains, the enormous cost of failures requires a significant commitment of resources to accounting for the human factor. For example, researchers have often sought to understand the mental states of human operators in detail and create extensive models to predict their actions. Our view is that system designers must account for the human factor to achieve high dependability but at lower costs than for safety-critical systems. We believe that this goal is achievable by focusing on human mistakes and their impact on system dependability, rather than attempting a broader understanding of human cognitive functions.

Our work is complementary to research on Human-Computer Interaction (HCI), which has traditionally focused on ease-of-use and cognitive models [17], in that we seek to provide infrastructural support to ease the task of operating highly available systems. For example, Barrett *et al.* report that one reason why operators favor using command line interfaces over graphical user interfaces is that the latter tools are often less trustworthy (e.g., their depiction of system state is less accurate) [1]. This suggests that HCI tools will only be effective when built around appropriate infrastructure support. Our vision of a runtime performance model that can be used to predict the impact of operator actions (Section 5) is an example of such infrastructural support. Further, our validation infrastructure will provide a "safety net" that can hide human mistakes caused by inexperience, stress, carelessness, or fatigue, which can occur even when the HCI tools provide accurate information.

Curiously, we envision guidance techniques that may even appear to conflict with the goals of HCI at first glance. For example, we plan to purposely add "inertia" to certain operations to reduce the possibility of serious mistakes, making it more difficult or time-consuming to perform these operations. Ultimately however, our techniques will protect systems against human mistakes and so they are compatible with the HCI goals.

Our work is related to several recent studies that have gathered empirical data on operator behaviors, mistakes, and their impact on systems [1, 16]. Brown and Patterson have proposed a methodology to consider humans in dependability benchmarking [4] and studied the impact of *undo*, an approach that is orthogonal (and complementary) to our validation and guidance approach, on repair times for several faults injected into an email service [3]. To our knowledge, however, we were the first group to publish detailed data on operator mistakes [15].

Our work is also related to no-futz computing [8]. However, we focus on increasing availability, whereas no-futz computing seeks to reduce futzing and costs.

3 Operator Mistakes

In order to build systems that reduce the possibility for operator mistakes, hide the mistakes, or tolerate them, we must first better understand the nature of mistakes. Thus, we believe that the systems community must develop common benchmarks and tools for studying human mistakes [4]. These benchmarks and tools should include infrastructure for experiment repeatability, e.g. instrumentation to record human action logs that can later be replayed. Finally, we need to build a shared body of knowledge on what kind of mistakes occur in practice, what their causes are, and how they impact performance and availability.

We have already begun to explore the nature of operator mistakes in the context of a multi-tier Internet service. In brief, we asked 21 volunteer operators to perform 43 benchmark operational tasks on a three-tier auction service. Each of the experiments involved either a scheduled-maintenance task (e.g., upgrading a software component) or a diagnose-and-repair task (e.g., discovering a disk failure and replacing the disk). To observe operator actions, we asked the operators to use a shell that records and timestamps every command typed into it and the corresponding result. Our service also recorded its throughput throughout each experiment so that we could later correlate mistakes with their impact on service performance and availability. Finally, one of our team members personally monitored each experiment and took notes to ease the interpretation of the logged commands and to record observables not logged by our infrastructure, such as edits of configuration files.

We observed a total of 42 mistakes, ranging from software misconfiguration, to fault misdiagnosis, to software restart mistakes. We also observed that a large number of mistakes (19) led to a degradation in service throughput. These results can now be used to design services that can tolerate or hide the mistakes we observed. For example, we were able to evaluate a prototype of our validation approach, which we describe in the next section.

We learned several important lessons from this experience: First, although we scripted much of the setup for each experiment, most of the scripts were not fully automated. This was a mistake. On several occasions, we only caught mistakes in the manual part of the setup just before the experiment began. Finding human subjects is too costly to risk invalidating any experiment in this manner. Second, infrastructural support for viewing the changes made to configuration files would have been very helpful. Third, we used a single observer for all of our experiments, which in retrospect, was a good decision because it kept the human recorded data as consistent as possible across the experiments. However, on several occasions, our observer scheduled too many experiments back-to-back, making fatigue a factor in the accuracy of the recorded observations. Fourth, our study was time-consuming. Even seemingly simple tasks may take operators a long time to complete; our experiments took an average of 1 hour and 45 minutes each. We also ran 6 warm up experiments to allow some of the novice operators to become more familiar with our system; these took on average 45 minutes each. Combining the different sources of data and analyzing them were also effort-intensive. Finally, enlisting volunteer operators was not an easy task. Indeed, one of the shortcomings of our study is the dearth of experienced operators among our volunteer subjects.

Despite these difficulties, our study (along with [1, 3]) proves that performing human-factor studies is not intractable for systems researchers. In fact, these studies should become easier to perform over time, as researchers share their tools, data, and experience with human-factor studies.

3.1 Open Issues

While our initial study represents a significant first step, it also raises many open issues.

Effects of long-term interactions. The short duration of our experiments meant that we did not account for a host of effects that are difficult to observe at short time-scales. For example, the effect of increasing familiarity with the system, the impact of user expectations, systolic load variations, stress and fatigue, and the impact of system evolution as features are added and removed.

Impact of experience. 14 of our 21 volunteer operators were graduate students with limited experience with the operation of computing services; 11 of the 14 were classified as novices, while 3 were classified as intermediates (on a three-tier scale: novice, intermediate, expert).

Impact of tools and monitoring infrastructures. Our study did not include any sophisticated tools to help with the service operation; we only provided our volunteers with a throughput visualization tool. Operators of real services have a wider set of monitoring and maintenance tools at their disposal.

Impact of complex tasks. Our experiments covered a small range of fairly simple operator tasks. Difficult tasks such as dealing with multiple overlapping component faults and changing the database schema that intuitively might be sources of more serious mistakes have not been studied.

Impact of stress. Many mistakes happen when humans are operating under stress, such as when trying to repair parts of a site that are down or under attack. Our initial experiments did not consider these high-stress situations.

Impact of realistic workloads. Finally, the workload offered to the service in our experiments was generated by a client emulator. It is unclear whether the emulator actually behaves as human users would and whether client behavior has any effect on operator behavior.

3.2 Current and Future Work

Encouraged by our positive initial experience, we are currently planning a much more thorough study of operator actions and mistakes. In particular, we plan to explore three complimentary directions: (1) survey and interview experienced operators, (2) improve our benchmarks and run more experiments, and (3) run and monitor all aspects of a real, live service for at least one year. The surveys and interviews will unearth the problems that afflict experienced operators even in the presence of production software and hardware and sophisticated support tools. This will enable us to design better benchmarks as well as guide our benchmarking effort to address areas of maximum impact. Running a live service will allow us to train the operators extensively, observe the effects of experience, stress, complex tasks, and real workloads, and study the efficacy of software designed to prevent, hide, or tolerate mistakes.

We have started this research by surveying professional network and database administrators to characterize the typical administration tasks, testing environments, and mistakes. Thus far, we have received 41 responses from network administrators and 51 responses from database administrators (DBAs). Many of the respondents seemed excited by our research and provided extensive answers to our questions. Thus, we believe that the challenge of recruiting experienced operators for human-factor studies is surmountable with an appropriate mix of financial rewards and positive research results.

A synopsis of the DBAs' responses follows. All respondents have at least 2 years of experience, with 71% of them having at least 5 years of experience. The most common tasks, accounting for 50% of the tasks performed by DBAs, relate to recovery, performance tuning, and database restructuring. Interestingly, only 16% of the DBAs test their actions on an exact replica of the online system. Testing is performed offline, manually or via ad-hoc scripts, by 55% of the DBAs. Finally, DBA mistakes are responsible (entirely or in part) for roughly 80% of the database administration problems reported. The most common mistakes are deployment, performance, and structure mistakes, all of which occur once per month on average. The current differences and separation between offline testing and online environments are cited as two of the main causes of the most frequent mistakes. These results further motivate the validation and guidance approaches discussed next.

4 Validation

In this section, we describe validation as one approach for hiding mistakes. Specifically, we are prototyping a *validation* environment that allows operators to validate the correctness of their actions before exposing them to clients [15]. Briefly, our validation approach works as follows. First, each component that will be affected by an operator action is taken offline, one at a time. All requests that would be sent to the component are redirected to components that provide the same functionality but that are unaffected by the operator action. After the operator action has been performed, the affected component is brought back online but is placed in a sand-box and connected to a validation harness. The validation harness consists of a library of real and proxy components that can be used to form a virtual service around the component under validation. The harness requires only a few machines and, thus, has negligible resource requirements for real services. Together, the sand-box and validation harness prevent the component, called *masked* component, from affecting the processing of client requests while providing an environment that looks exactly like the live environment.

The system then uses the validation harness to compare the behavior of the component affected the operator action against that of a similar but unaffected component. If this comparison fails, the system alerts the operator before the masked component is placed in active service. The comparison can either be against another live component, or against a previously collected trace. After the component passes the validation process, it is migrated from the sand-box into the live operating environment without any changes to its configurations.

Using our prototype validation infrastructure, we were able to detect and hide 66% of the mistakes we observed in our initial human-factors experiments. A detailed evaluation of our prototype can be found in [15].

4.1 Open Issues

Although our validation prototype represents a good first step, we now discuss several open issues.

Isolation. A critical challenge is how to isolate the components from each other yet allow them to be migrated between live and validation environments with no changes to their internal state or to external configuration parameters, such as network addresses. We can achieve this isolation and transparent migration at the granularity of an entire node by running nodes over a virtual network, yet for other components this remains a concern.

State management. Any validation framework is faced with two state management issues: (1) how to start up a masked component with the appropriate internal state; and (2) how to migrate a validated component to the online system without migrating state that was built up during validation but is not valid for the live service.

Bootstrapping. A difficult open problem for validation is how to check the correctness of a masked component when there is no component or trace to compare against. This problem occurs when the operator action correctly changes the behavior of the component for the first time.

Non-determinism. Validation depends on good comparator functions. Exact-match comparator functions are simple but limiting because of application non-determinism. For example, ads that should be placed in a Web page may correctly change over time. Thus, some relaxation in the definition of similarity is often needed, yet such relaxation is application-specific.

Resource management. Regardless of the validation technique and comparator functions, validation retains resources that could be used more productively when no mistakes are made. Under high load, when all available resources should be used to provide a better quality of service, validation attempts to prevent operator-induced service unavailability at the cost of performance. This suggests that adjusting the length of the validation period according to load may strike an appropriate compromise between availability and performance.

Comprehensive validation. Validation will be most effective if it can be applied to all system components. To date, our prototyping has been limited to the validation of Web and application servers in a three-tier service. Designing a framework that can successfully validate other components, such as databases, load balancers, switches, and firewalls, presents many more challenges.

4.2 Current Work

We are extending our validation framework in two ways to address some of the above issues. First, we are extending our validation techniques to include the database, an important component of multi-tier Internet services. Specifically, we are modifying a replicated database framework, called C-JDBC, which allows for mirroring a database across multiple machines. We are facing several challenges, such as the management of the large persistent state when bringing a masked database up-to-date, and the performance consequences of this operation.

Second, we are considering how to apply validation when we do not have a known correct instance for comparison. Specifically, we are exploring an approach we call *model-based* validation. The idea is to validate the system behavior resulting from an operator action against an operational model devised by the system designer. For example, when configuring a load balancing device, the operator is typically attempting to even out the utilization of components downstream from the load balancer. Thus, if we can conveniently express this resulting behavior (or model) and check it during validation, we can validate the operator's changes to the device configuration. We are currently designing a language that can express such models for a set of components, including load balancers, routers, and firewalls.

5 Guidance

In this section, we consider how services can prevent mistakes by *guiding* operator actions when validation is not applicable. For example, when the operator is trying to restore service during a service disruption, he may not have the leisure of validating his actions since repairs need to be completed as quickly as possible. Guidance can also reduce repair time by helping the operator to more rapidly find and choose the correct actions.

One possible strategy is to use the data gathered in operator studies to create models of operator behaviors and likely mistakes, and then build services that use these models together with models of the services' own behaviors to guide operator actions. In particular, we envision services that monitor and predict the potential impact of operator actions, provide feedback to the operator before the actions are actually performed, suggest actions that can reduce the chances for mistakes, and even require appropriate authority, such as approval from a senior operator, before allowing actions that might negatively impact the service.

5.1 Future Work

Our guidance strategy relies on the system to maintain several representations of itself: an operator model, a performance model, and an availability model.

Operator behavior models. To date, operator modeling has mostly been addressed in the context of safety-critical systems or those where the cost of human mistakes can be very high. Rather than follow the more complex cognitive approaches that have evolved in these areas (see Section 2), we envision a simpler approach in which the operator is modeled using stochastic state machines describing expected operator behavior.

Our intended approach is similar in spirit to the Operation Function Models (OFMs) first proposed in [12]. Like the OFMs, our models will be based on finite automata with probabilistic transitions of operator actions, which can be composed hierarchically. However, we do not plan on representing the mental states of the operator, nor do we expect to model the operator under normal operating conditions.

An important open issue to be considered is whether tasks are repeated enough times with sufficient similarity to support the construction of meaningful models. In the absence of a meaningful operator model for a certain task, we need to rely on the other models for guidance.

Predicting the impact of operator actions. Along with the operator behavior models, we will need a software monitoring infrastructure for the service to represent itself. In particular, it is important for the service to monitor the configuration and utilization of its hardware components. This information can be combined with analytical models of performance and availability similar to those proposed in [5, 14] to predict the impact of operator actions. For example, the performance (availability) model could estimate the performance (availability) degradation that would result from taking a Web server into the validation slice for a software upgrade.

Guiding and constraining operator actions. Using our operator models, we will develop software to guide operator actions. Guiding the operator entails assisting him/her in selecting actions likely to address a specific scenario. These correspond to what today might be entries in an operations manual. However, unlike a manual, our guidance system can directly observe current system state and past action history in suggesting actions.

Our approach to guide the operator uses the behavior models, the monitoring infrastructure, and the analytical models to determine the system impact of each action. Given a set of behavior model transitions, the system can suggest the operator actions that are least likely to cause a service disruption or performance degradation. To do so, the system will first determine the set of components that are likely to be affected by each operator action and the probability that these components would fail as a result of the action. The system will then predict the overall impact for each possible action along with the likelihoods of each of these scenarios.

To allow operators to deviate from automatic guidance yet allow a service to still protect itself against arbitrary behaviors, we will need *dampers*. The basic idea behind the damper is to introduce inertia representing the potential negative impact of an operator's action in case the action is a mistake. For example, if an action is likely to have a small negative (performance or availability) impact on the service, the damper might simply ask the operator to verify that he indeed really wants to perform that action. On the other hand, if the potential impact of the operator's action is great enough, the system may require the intervention of a senior or "master" operator before allowing the action to take place. In a similar vein, Bhaskaran *et al.* [2] have recently argued that systems should require acknowledgements from operators before certain actions are performed. However, the need for acknowledgements in their proposed systems would be determined by operator behavior models only.

6 Discussion and Conclusion

The research we have advocated in this paper is applicable to many other areas of Computer Science. In this section, we motivate how some of these areas may be improved by accounting for human actions and mistakes.

In the area of Operating Systems, little or no attention has been paid to how mistakes can impact the system. For example, when adding a device driver, a simple mistake can bring down the system. Also, little attention has been given to the mistakes made when adding and removing application software. Addressing these mistakes explicitly would increase robustness and dependability.

In the area of Software Engineering, again historically there has been little direct investigation into why and how people make mistakes. A small body of work exists in examining common types of programming errors, yet little is understood about the processes that cause there errors. An interesting example of work in this direction is [10], in which the authors exploit data mining techniques to detect cut-and-paste mistakes.

Finally, in the field of Computer Networks, the Border Gateway Routing Protocol suffered from severe disruptions when bad routing entries were introduced, mostly as a result of human mistakes [11]. Again, addressing human mistakes explicitly in this context can significantly increase routing robustness and dependability.

In conclusion, we hope that this paper included enough motivation, preliminary results, and research directions to convince our colleagues that designers must consider human-system interactions and the mistakes that may result explicitly in their designs. In this context, human-factors studies, techniques to prevent or hide human mistakes, and models to guide operator actions all seem required. Failure to address humans explicitly will perpetuate the current scenario of human-produced unavailability and its costly and annoying consequences.

References

[1] BARRETT, R., MAGLIO, P. P., KANDOGAN, E., AND BAILEY, J. Usable Autonomic Computing Systems: the Administrator's Perspective. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC'04)* (May 2004).

[2] BHASKARAN, S. M., IZADI, B., AND SPAINHOWER, L. Coordinating Human Operators and Computer Agents for Recovery-Oriented Computing. In *Proceedings of the International Conference on Information Reuse and Integration* (Nov. 2004).

[3] BROWN, A. *A Recovery-oriented Approach to Dependable Services: Repairing Past Errors with System-wide Undo*. PhD thesis, Computer Science Division-University of California, Berkeley, 2003.

[4] BROWN, A., AND PATTERSON, D. A. Including the Human Factor in Dependability Benchmarks. In *Proceedings of the DSN Workshop on Dependability Benchmarking* (June 2002).

[5] CARRERA, E. V., AND BIANCHINI, R. Efficiency vs. Portability in Cluster-Based Network Servers. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming (PPoPP)* (June 2001).

[6] GOVINDARAJ, T., WARD, S. L., POTURALSKI, R. J., AND VIKMANIS, M. M. An Experiment and a Model for the Human Operator in a Time-Constrained Competing-Task Environment. *IEEE Transactions on Systems Man and Cybernetics 15*, 4 (1985).

[7] GRAY, J. Why do Computers Stop and What Can Be Done About It? In *Proceedings of 5th Symposium on Reliability in Distributed Software and Database Systems* (Jan. 1986).

[8] HOLLAND, D. A., JOSEPHSON, W., MAGOUTIS, K., SELTZER, M. I., STEIN, C. A., AND LIM, A. Research Issues in No-Futz Computing. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)* (May 2003).

[9] KEPHART, J. O., AND CHESS, D. M. The Vision of Autonomic Computing. *IEEE Computer 36*, 1 (Jan. 2003).

[10] LI, Z., LU, S., MYAGMAR, S., AND ZHOU, Y. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)* (Dec. 2004).

[11] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Understanding BGP Misconfiguration. In *Proceedings of the ACM SIGCOMM '02 Conference on Communications Architectures and Protocols* (Aug. 2002).

[12] MITCHELL, C. M. GT-MSOCC: A Domain for Research on Human-Computer Interaction and Decision Aiding in Supervisory Control Systems. *IEEE Transactions on Systems, Man and Cybernetics 17*, 4 (1987), 553–572.

[13] MURPHY, B., AND LEVIDOW, B. Windows 2000 Dependability. Tech. Rep. MSR-TR-2000-56, Microsoft Research, June 2000.

[14] NAGARAJA, K., GAMA, G., MARTIN, R. P., JR., W. M., AND NGUYEN, T. D. Quantifying Performability in Cluster-Based Services. *IEEE Transactions on Parallel and Distributed Systems 16*, 5 (May 2005).

[15] NAGARAJA, K., OLIVEIRA, F., BIANCHINI, R., MARTIN, R. P., AND NGUYEN, T. D. Understanding and Dealing with Operator Mistakes in Internet Services. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '04)* (Dec. 2004).

[16] OPPENHEIMER, D., GANAPATHI, A., AND PATTERSON, D. Why do Internet Services Fail, and What Can Be Done About It. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS'03)* (Mar. 2003).

[17] RASMUSSEN, J. *Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering*. North-Holland, New York, 1986.

[18] WALDEN, R. S., AND ROUSE, W. B. A Queueing Model of Pilot Decisionmaking in a Multitask Flight Management Situation. *IEEE Transactions on Systems, Man and Cybernetics 8*, 12 (Dec. 1978).