Work-arounds, Make-work, and Kludges

Philip Koopman, ***Carnegie Mellon University***
Robert R. Hoffman, ***Institute for Human and Machine Cognition***

Paradigms are often defined partly in terms of what they are not, or in terms of what they are reacting against. The paradigm of human-centered computing is no exception. In response to an essay in the Jan./Feb. 2002 Human-Centered Computing column "The State of Cognitive Systems Engineering,"[1] we had a lengthy discussion on the question, What is a *user-hostile* system? The following quote is from that essay:

> The road to user-hostile systems is paved with user-centered intentions on the part of the designers. Even smart, clever, well-intentioned people can build devices that are fragile and hostile, devices that force the human to adapt and build local kludges and work-arounds. Worse still, even if one is aware of this trap, one will still fall into it.

We decided that the terms *kludge* and *work-around*, and also the related concept of *make-work*, have yet to be clearly defined for the intelligent systems community. Human-centered systems are different from user-hostile systems as well as from systems based on a designer-centered approach.[2] In this essay, we try to clarify the senses of these three terms and suggest ways we might study work-arounds, make-work, and kludges as an integral part of human-computer systems—rather than as embarrassing necessities that are best swept under the computing research rug.



Editors: Robert R. Hoffman, Patrick J. Hayes, and Kenneth M. Ford
Institute for Human and Machine Cognition, University of West Florida
rhoffman@ai.uwf.edu

Work-around

We all have had to create and use a work-around at some point to get a user-hostile computer to do our bidding. Documented evidence attests to the pervasiveness of work-arounds.[3] Sometimes the work-around is inspired by a friend's hint; sometimes it's spelled out in a help page on the Web; sometimes it's discovered by flailing around until something just happens to work, more or less. Although work-arounds might seem inherently ad hoc, in many situations they make the difference between success and failure.

The *Concise Oxford Dictionary* definition of *work-around* is "a method for overcoming a problem or limitation in a program or system." (Neither the *Oxford Unabridged Dictionary* nor the *Oxford Dictionary of Computing* lists "work-around," but, ironically, the *Concise Oxford Dictionary* does. Go figure.) This definition doesn't go far enough to pin down the concept so that it might be more amenable for study. Does the "problem" being overcome mean a design defect or something as simple as a component failure due to wear-out? Does "limitation" mean an unimplemented (but clearly imaginable and clearly useful) feature, an intentionally precluded action, or an inability to use the tool to cope with an unanticipated operating environment? And is the work-around method written down, foreseen by designers, or made up on the fly?

Wikipedia, a Web-based encyclopedia (www.wikipedia.org), defines *work-around* as

> A bypass of a recognized problem in a system. A workaround is typically a temporary fix that implies that a genuine solution to the problem is needed. Frequently workarounds are as creative as true solutions, involving out-of-the-box thinking in their creation. A workaround is not a permanent solution. Typically they are considered brittle in that they will not respond well to further pressure from a system beyond the original design. In implementing a workaround it is important to flag the change so as to later implement a proper solution. Placing pressure on a workaround may result in later failures in the system. For example, in computer programming workarounds are often used to address a problem in a library, such as an incorrect return value. When the library is changed, the workaround may break the overall program functionality, since it may expect the older, wrong behavior from the library.

1094-7167/03/$17.00 © 2003 IEEE
Published by the IEEE Computer Society

Whatis.com, a Web-based glossary (http://whatis.techtarget.com), defines *work-around* as

> A method, sometimes used temporarily, for achieving a task or goal when the usual or planned method isn't working. In information technology, a work-around is often used to overcome hardware, programming, or communication problems. Once a problem is fixed, a work-around is usually abandoned.

This last statement seems to point to an empirical base that, as far as we know, doesn't exist. Be that as it may, all three definitions seem to capture the usual intuitive notion and are all right as far as they go. However, they suggest trying to unpack the different senses of *work-around* might be useful.

Les Gasser defined three alternative senses of what he termed *adaptations*: *fitting*, *augmenting*, and *working around*.[4] Fitting jobs and work schedules to accommodate computing-resource limits is an adaptation you can make to avoid a computing-system failure due to overload. Augmenting involves removing anomalies from the computing environment to avoid triggering problems, such as scrubbing data and training users to avoid making certain types of mistakes. Working around involves users altering input data or procedures to compensate for system shortcomings, or using backup systems.

When a work-around is not readily available, people might even change their goal to something that they know the system can do. For example, if sending a file as an email attachment doesn't work, a potential work-around is putting it on an FTP or Web server and just sending a pointer to the intended recipient. However, we look at this solution simply as a work-around taking place at the next level up from a person's interaction with a computer. Gasser reported the common practice of using informal processes based on personal relationships to expedite organizational processes, with computer-based workflow activity backfilled after the actual task has been performed. Gasser gives the example of a purchasing manager handwriting a requisition to avoid delay and then having the requisition entered into a computer after the fact for tracking purposes.

We define the general sense of *work-around* as follows:

> When a path to a goal is blocked, people use their knowledge to create and execute an alternate path to that goal.

A block can occur when you don't know whether a path to your goal even exists (that is, you're working on a system with invisible and therefore unknown functionalities). A block can also occur when a known path is confusing, laborious, broken, or otherwise hostile. In this case, you might create a new path that you perceive as either more user friendly (for example, intuitive or transparent) or simpler and hence better able to expedite the work. In both cases, the method you create might or might not actually be simpler than that included in the existing functionality.

We propose grouping four alternative senses of *work-around*, hinging on the nature of the block:

A block can occur when you don't know whether a path to your goal even exists. A block can also occur when a known path is confusing, laborious, broken, or otherwise hostile.

- Completing tasks despite design flaws in a computational tool
- Completing tasks despite component failures
- Extending functionality to complete a new task
- Intentionally evading designed limits in an effort to overcome them

This set is based on the notion of a system problem or limitation as expressed in the dictionary definitions given earlier and on Gasser's emphasis on the intent of the person exercising the work-around. Of course, you can use the mechanisms for executing a work-around for more than one of these purposes, and we'll discuss them as we go.

(We've heard of a fifth meaning of "work-around," as a personality trait. Trainers might attribute this to their students, asserting that some individuals seem prone to ignore what they're told. We don't take seriously the idea that "working-around" is a personality trait, on par with, say, sociability.)

Sense 1: Completing tasks despite design flaws

Work-around (1): A procedural change in computer system use intended to compensate for a design flaw, typically a software behavior that is perceived to be a flaw.

Perhaps the most common experience of a work-around with computers is getting buggy programs to work well enough to accomplish a task or achieve a goal. Software companies often issue work-arounds to help users cope with bugs. While you can argue that almost any nontrivial software will have defects, desktop office automation software gets a lot of press for being plagued with bugs. An example that came up on a simple Web search was Microsoft Security Bulletin MS02-027, which states in part, "Customers using IE should implement the work-around detailed in the FAQ."[5] This particular work-around is a set of steps the user performs to reconfigure Internet Explorer.

Some definitions (for example, in Whatis.com) state that a work-around is a temporary compensation until a defect can be corrected. This is the meaning we also find in Eric Raymond's Jargon File.[6] MS02-027 is an example. It involves disabling Gopher service until a patch is released that eliminates a buffer overrun security vulnerability. Also, some work-arounds are only partial solutions. For example, MS02-027 actually disables a service and leaves users on their own if they must use Gopher for something, but it does improve the security of using IE for other tasks.

Sometimes work-arounds are put in place not because the software itself is defective or is perceived to be defective but because the software requirements or operating environment are inadequate. The designer's path through a system might be so confusing, laborious, or otherwise hostile that people create their own paths. These paths might or might not be "right," but are at least perceived to be friendlier (for example, intuitive or transparent), simpler, or less time-consuming.

Generally you can think of a work-around in Sense 1 as a situation in which a user tries to follow a set of well-defined steps to accomplish a particular goal, but something goes wrong that blocks the user from accomplishing those steps. For example, the user might encounter some mind-

boggling complexity in the steps' description or a software design flaw that prevents completing those steps in a particular set of circumstances. For instance, Stephan Poelmans describes work-arounds as deviations from a defined workflow system, whether to overcome problems or to improve user experience.[7]

Sense 2: Completing tasks despite component failures

Work-around (2): A procedural change to using a computer system intended to compensate for a hardware or component failure.

We can expect most computer systems of reasonable complexity to suffer component failures, sometimes regularly. Work-arounds in such cases involve people diagnosing a failure or recognizing a pattern of system misbehaviors and then executing an alternate procedure to accomplish a goal. The simplest work-around strategy in the face of a component failure is to have a backup system (computerized or manual) you can use in a pinch, even though it might have limited functionality.[4]

Component failure work-arounds are similar to design defect work-arounds but differ in a few key ways. First, a completely redundant backup system is often helpful for component failures, but it might well have the same design defects as the primary system and thus might not help with design defects. Having heterogeneously designed systems is a way to provide at least some work-around capability for both design and component failures—for example, having both a Windows and a Unix desktop computer, or installing multiple Web browsers from competing vendors.

One good thing about component failures compared to design failures is that it might be possible to fix a component failure very quickly, whereas corrections of design problems can take a long time to create, and work-arounds for them might require trial and error. So, in some cases a simple adaptation strategy of rescheduling work hours to wait for a repair might be possible instead of using a bona fide work-around.

Sense 3: Extending functionality

Work-around (3): A new procedure that uses a computer system in a way not originally envisioned to accomplish a task.

Some work-arounds are necessary because the computer or software as originally designed simply doesn't address the problem or task at hand. This can be because the task you're trying to accomplish is "new" and there hasn't been time to make software changes. In some cases you can foresee this, and you can design work-arounds for existing software as part of a business process change. But at other times, this might not be practical. Working-around in Sense 3 is a common activity. An example would be to use spreadsheet software to compose an essay outline.

In the area of task analysis, Michael Albers argued that goal-driven approaches (that is, ad hoc activities) rather than a preplanned task analysis can be necessary when there are failures or exceptional operating conditions.[8] This reflects the fact that in many instances users are left on their own to determine a work-around because the great number of potential failure and use extension scenarios exceeds currently known techniques for analysis and documentation.

Users are left on their own to determine a work-around because the number of potential failure and use extension scenarios exceeds the currently known techniques for analysis and documentation.

Sense 4: Users intentionally misleading their computers

Work-around (4): A procedural deviation to circumvent an intentionally designed-in limit or constraint on computer system operation.

Sometimes we use work-arounds to evade a computer system's designed-in behavior. This can be because we're trying to use a computer to do something it wasn't intended to do, such as holding down the shift key to defeat a music CD copy protection scheme.[9] This differs from the other types of work-arounds in that the user is trying to do something that the system designers specifically intended the user to not do, or intended to prevent the user from doing, rather than accommodating to a design defect or gap in design requirements.

Some in the field of computer-aided design define *work-around* in this way, as a sort of subversion of a task:

> In subversion [as a constraint negotiation strategy], the user modifies the task approach to take advantage of known weaknesses in the tool, overriding the spirit but not the mechanism by which the constraint is imposed (also known as a "workaround").[10]

Indeed, in the context of CAD, a circuit design arrangement that exploits a loophole in automated design-rule checkers will likely result in a chip design that doesn't work. To increase productivity in safety-critical systems and other critical applications, people might use work-arounds to similarly evade safety features, but they do so at a cost of increased risk. Donald Day conducted a study of work-arounds in this sense of subversion, based on results from a questionnaire concerning the factors that affect flexibility when working under software-imposed constraints.[10] Approximately 200 professionals in computer-aided systems engineering completed the questionnaire. While most respondents reported that they didn't feel especially encumbered by their software tools, most also reported that they absolutely reserved the right to override or work around software-imposed constraints, assuming sufficient justification exists (for instance, to adhere to overriding professional standards or to deal with an obvious software design fault). Those who felt "controlled" by their software tended to report lower levels of satisfaction with it and a higher likelihood of being subversive. Conversely, those who felt that they were less controlled by their software tended to report greater levels of satisfaction with the software and a greater likelihood of conforming to the constraints that the software imposed.

By implication, Day is saying that evading design features is a constructive activity in some circumstances. Such instances abound in the area of graphics software. Alison Black studied how novice graphic designers get "sucked in" to the conventions and limitations of graphics software

that was created without benefit of a user needs analysis.[11] She noted the work-arounds that users created to cope with awkward and inefficient commands in graphics software (for example, the problem of filling in enclosed objects that are created piecemeal).

Gasser interviewed approximately 60 professionals who worked at several manufacturing firms to study how they adapt to computing resources.[4] He reported several instances in which users "gamed" their computer systems by entering data known to be inaccurate, or otherwise not following expected normal system use. But the practitioners were able to get their systems to yield usable results, ones that sometimes were more "accurate" than those derived by "following the rules." An example of this in everyday life is booking a conference room for a critical meeting for a time block starting a half hour early on the expectation that any preceding meeting will run later than scheduled. Here's an example from Gasser's report:

> Engineering analysts working [at] a large firm which designed and constructed chemical plants, learned which analyses could go wrong in the package of complex analytical programs they used, and how to correct for untenable results. For example, they routinely input temperature coefficients for pipes carrying hot fluids as though the pipes were intended to operate cold, causing the analysis program to disregard certain heating stress calculations. Over years of experience with this program, the engineers had learned that entering the "correct" information would lead to erroneous results and the pipes would not work properly in the final design. They worked around the technical problems of the program by "running the hot pipes cold" when using it.[4]

The positive value of work-arounds is a recurrent theme in all the empirical work we found. One more example is a study by Gary Parish and William Sollfrey,[12] who examined work-arounds' effects on unmanned spacecraft, including performing reliability calculations that incorporated the expected benefits of work-arounds. They found that even the relatively primitive satellites of that era benefited from work-arounds, with 18 life-extending work-arounds recorded for 25 satellites studied. Their definition of the term emphasized procedural and software changes and specifically excluded built-in redundancy.

Kludge

Webster's Ninth Collegiate Dictionary defines *kludge* as a system (especially computers) made of components that are poorly matched or were originally intended for some other use. This source indicates that the word has unknown origins but cites 1962 as the earliest recorded use. This might refer to a paper in which Jackson Granholm,[13] his tongue firmly in his cheek, suggested several rules of applied "kludgemanship," exemplified by a number of ways in which hackers could create a variety of clever kludges. For example, when using magnetic tape as a storage medium, the "kludgeman" would use both odd and even parity and as many widths as he could find reels for.

Kludge and work-around are related in that both originated because an alternative solution that is either more elegant or more appropriate is for some reason unavailable.

We've seen the word spelled both "kluge" and "kludge" and have heard it pronounced both as "klooge"(as in "stooge") and "kludge" (as in "fudge"), with the former being the pronunciation most people prefer. Granholm traces the word to the German "kluge" (pronounced "kloo-ga"), meaning smart, witty, or clever. We wouldn't be surprised if the term was in fact an acronym for "*k*nowledge and *l*earning *u*sed to derive *g*ood *e*ffects." Nor would we be surprised if the term comes from the last name of an absent-minded German professor who was notorious for making electromagnets from hairpins, old tractor batteries, and fence wire. Raymond asserts that the term came into use in the 1940s in reference to a device called the Kluge Paper Feeder, a "fiendishly complex assortment of cams, belts, linkages … devilishly difficult to repair … but oh, so clever."[6]

The general difference between a kludge and a work-around is that a kludge is a set of design artifacts or elements, in one of these forms:

- A fix that is awkward or clumsy but is at least temporarily effective
- An overall design that is of questionable elegance or downright ugly

On the other hand, a work-around is generally considered to be a procedural variation that a person creates and uses. Raymond's Jargon File sees a kludge as an initial design approach and a work-around as a temporary bug fix.[6] On the other hand, most other sources, and our own experience, indicate that using "kludge" rather than "work-around" is more appropriate for temporary fixes. In any event, we feel that the distinction of procedure versus designed artifact is useful, one that accords with most documented definitions. This distinction seems largely consistent with various documented usages in which kludge applies only to hardware and work-around applies to both procedures and software.

Kludge and *work-around* are related in that both originated because an alternative solution that is either more elegant or more appropriate is for some reason unavailable.[13] These concepts are also related in that they are both, apparently, permanent residents of the technology landscape:

> Hardware and software products are sometimes the result of adding a new and basically incompatible design to an original design rather than redesigning the product completely. Users often have a different opinion than the designers do. To the extent that information technology products are combinations of elements originating from a variety of design philosophies and constraints, almost any product is bound to contain some element of kludginess.[14]

Make-work

Make-work activities are repetitive, boring, time-consuming activities that someone must engage in to accomplish something that could not be accomplished using a shortcut, or that one should be able to easily accomplish but cannot. An example is when a display of visualized scientific data relies on screen sectoring to permit the presentation of multiple data types or fields. Sectoring, an attempt to overcome screen real-estate limitations, places a great burden on the user, requiring repetitive

point-click-and-drag minimization and maximization to view individual data fields. Another example is having to cut, paste, cut, then "paste as" to get a clean chart transfer from Excel to Corel Draw.

Opportunities

Now that our semantic deboning of the triad has played itself out, you must be asking just the sorts of questions that HCC advocates would raise. We began this essay with a reminder that human-centered systems differ from user-hostile systems in that user-hostile systems are often based on a designer-centered approach.[2] In this regard, Granholm was right on the money when he pointed to some of the reasons why kludges and work-arounds are necessary:

> It is sad, but true, that a kludge cannot be designed under just any old organizational structure. One of the most helpful atmospheres in which a kludge may arrive at full flower is that of complete, massive, and iron-bound compartmentalization. It is a good idea if the I/O men, say, are not only not allowed to talk to the mainframe designers, but also that they have, in fact, never met them. Interface cross-talk should, by all means, be done by edict and directive, and not by memo and design note. After all, someone has to hew the line on design philosophy, or people will go off in all directions.[13]

His sarcasm rings true. Day made a similar point in a literal tone:

> User dissatisfaction with some constraint management techniques may be attributable in part to a mismatch between builders' and users' views of appropriate and necessary design practices.[10]

Rather than simply bemoaning this state of affairs, we could see it as an opportunity. This involves adopting the goal of Day's[10] and Gasser's[4,15] empirical studies:

> Instead of studying how to eliminate problems … we are attempting to describe and explain the dynamics of computer use over time. This leads us to focus on how circumstances persist and evolve, rather than why they exist in the first place.[4]

Might not research on work-around, kludging, and make-work (WKM) activities reveal ways we could achieve human-centering? Might it not suggest new methods for studying or evaluating human-centeredness?

Also wide open for research are the social and organizational aspects of work-arounds. When and why do organizations choose to ignore subversive, work-around activities? Individuals aren't the only ones who create work-arounds. As Day pointed out, teams and project managers also create them. Why? When? Inspired by Parish and Sollfrey,[12] can we quantify the effects of work-arounds on system dependability and usability?

There's no denying that WKMs will exist as an enduring part of the computing experience. But much software culture today is based on the notion of trying to achieve perfect software, which of course is an in-your-face manifestation of designer-centered design. Perhaps it's time to start treating WKMs as first-class citizens in terms of research topics, development efforts, and teaching. Gasser captured this point beautifully:

Much software culture today is based on the notion of trying to achieve perfect software, which of course is an in-your-face manifestation of designer-centered design.

> A fundamental assumption of design-oriented literature is that managers, designers, and system proponents define what are "rational" actions in dealing with a computing system —rational procedures for using a system, getting problems fixed, etc. From this viewpoint, workers in our study who worked around formal systems were "escaping" from the constraints of the system, and acting "irrationally" with respect to system goals and managers' expectations. But we have discovered that, far from acting irrationally, the informal practical actions of participants actually make systems more usable locally. Informal fitting, augmenting, and working around are essential and locally rational parts of system use. Appropriate and rational action is defined by the demands of the work situation and the institutional arrangements surrounding computing, not by the ideologies of managers or the presumed necessities of system structure.[4]

This clearly places WKM behavior in the legitimate arena of "adaptive design," in which users help to finish the system design.[16] Researchable topics abound for studying the deliberate, systematic creation and deployment of WKMs. It might be useful to explore ways to build compensation mechanisms into software to make the user-plus-software system more resilient—that is, ways to make work-arounds easier for users to create, document, and share. For example, an online help system might notice that a user is repeatedly getting a particular error message and might offer advice on work-arounds relevant to that error. Help systems today assume that the user is making a mistake and offer canned tutorial advice, but typically do not consider that the real problem might be a software defect, limitation, or user hostility. Users must search the Web or other databases to find work-arounds, if they're published at all.

Although foreseeing all possible failure modes might be impossible, it's probably useful to catch and represent likely failure modes at design time. The only way to do that is to empirically study the "envisioned world"[17] and evaluate software for usefulness and usability. Indeed, safety-critical systems use Failure Mode Effects and Criticality Analysis[18] to represent the effects of hardware component failures and subsystem functional failures. We can extend this to look at the ways that users are blocked and frustrated as they try to complete sequences of steps.

In addition to possible practical applications, research along the lines we propose might lead to a scientific understanding of WKM behavior and a formal representation of work-arounds and kludges.[19]

WKMs are here to stay, perhaps even after computer scientists take notions of human-centering to heart. Intelligent and human-centered systems do not blame or punish users for making mistakes. Perhaps it's time for researchers and technologists to start including WKMs more explicitly in their agendas. *IEEE Intelligent Systems* should be the home for that agenda. ■

Acknowledgments

Phil Koopman's contribution to this article was supported in part by the General Motors Collaborative Laboratory at Carnegie Mellon University. Robert Hoffman's contribution was supported through his participation in the Advanced Decision Architectures Collaborative Technology Alliance, sponsored by the US Army Research Laboratory under cooperative agreement DAAD19-01-2-0009.

References

1. R.R. Hoffman, G. Klein, and K.R. Laughery, "The State of Cognitive Systems Engineering," *IEEE Intelligent Systems*, vol. 17, no. 1, Jan./Feb. 2002, pp. 73–75.
2. R.R. Hoffman et al., "A Rose by Any Other Name … Would Probably Be Given an Acronym," *IEEE Intelligent Systems*, vol. 17, no. 4, July/Aug. 2002, pp. 72–80.
3. R. Kling and W. Scacchi, "Recurrent Dilemmas of Computer Use in Complex Organizations," *Proc. 1979 Nat'l Computer Conf.*, AFIPS Press, vol. 48, 1979, pp. 107–116.
4. L. Gasser, "The Integration of Computing and Routine Work," *ACM Trans. Office Information Systems*, vol. 4, no. 3, July 1986, pp. 205–225.
5. "Microsoft Security Bulletin MS02-027," 28 Feb. 2003, Microsoft, www.microsoft.com/technet/security/bulletin/MS02-027.asp.
6. E. Raymond, *The New Hacker's Dictionary*, MIT Press, 1991.
7. S. Poelmans, "Workarounds and Distributed Viscosity in a Workflow System: A Case Study," *ACM SIGGROUP Bull.*, vol. 20, no. 3, Dec. 1999, pp.11–12.
8. M. Albers, "Goal-Driven Task Analysis: Improving Situation Awareness for Complex Problem-Solving," *Proc. 16th Ann. Int'l Conf. Computer Documentation*, ACM Press, 1998, pp. 234–242.
9. J.A. Halderman, "Analysis of the MediaMax CD3 Copy-Prevention System," tech. report TR-679-03, Princeton Univ., Oct. 2003, http://ncstrl.cs.princeton.edu/expand.php?id=TR-679-03.
10. D. Day, "User Responses to Constraints in Computerized Design Tools: An Extended Abstract," *Software Eng. Notes*, vol. 21, no. 5, Sept. 1996, pp. 47–50.
11. A. Black, "Visible Planning on Paper and on Screen: The Impact of Working Medium on Decision-Making by Novice Graphic Designers," *Behavior and Information Technology*, vol. 9, no. 4, 1990, pp. 283–296.
12. G. Parish and W. Sollfrey, *Preliminary Analysis of the Effect of Work-Arounds on Space System Performance and Procurement Requirements: A Proposal*, Rand Corp. report N-1260-AF, 1980.
13. J. Granholm, "How to Design a Kludge," *Datamation*, vol. 8, Feb. 1962, pp. 30–31.
14. L. Swanson and M.J. Warden, "Kludge," Whatis.com, 2003; http://whatis.techtarget.com.
15. E.D. arses, "The Constraint Satisfaction Approach to Design: A Psychological Investigation," *Acta Psychologica*, vol. 78, 1991, pp. 307–325.
16. D.D. Woods and S.W.A. Dekker, "Anticipating the Effects of Technological Change: A New Era of Dynamics for Human Factors," *Theoretical Issues in Ergonomic Science*, vol. 1, no. 3, 2000, pp. 272–282.
17. S.W.A. Dekker, J.M. Nyce, and R.R. Hoffman, "From Contextual Inquiry to Designable Futures: What Do We Need to Get There? *IEEE Intelligent Systems*, vol. 18, no. 2, Mar./Apr. 2003, pp. 74–77.
18. C. McCollin, "Working around Failure," *Manufacturing Engineer*, vol. 78, no. 1, Feb. 1999, pp. 37–40.
19. C. Martin, *Functional Fault Simulation for Distributed Embedded Systems*, master's thesis, Electrical and Computer Eng. Dept., Carnegie Mellon Univ., 2001.

Philip Koopman is an associate professor at Carnegie Mellon University. He is affiliated with the Department of Electrical and Computer Engineering, the Institute for Software Research International, and the Institute for Complex Engineered Systems. Contact him at the ECE Dept., Carnegie Mellon Univ., 5000 Forbes Ave., Pittsburgh, PA 15213; koopman@cmu.edu.

Robert R. Hoffman is a research scientist at the Institute for Human and Machine Cognition. Contact him at the IHMC, 40 Alcaniz St., Pensacola, FL 32501; rhoffman@ihmc.us.

Addendum

The September/October 2003 installment of this column was titled "The Borg Hypothesis." Jane Abbate, historian of technology at the Chemical Heritage Foundation, points out to us that the term *cyborg* was first introduced in 1960 by Manfred Clynes and Nathan Kline ("Cyborgs and Space," *Astronautics*, vol. 5, pp. 26–27, 74–76). They had a similar mission to that described in our essay: preparing humankind for space travel.

But this early vision of a cyborg future did not include the crucial concept of fitting the human body with intelligent technologies, as opposed to merely adding cybernetic mechanisms such as artificial limbs or even artificial eyeballs. The notion that control theory ala Norbert Wiener (that is, types of feedback loops) is sufficient for intelligence is at least arguable. The point of our essay was to consider the integration of intelligent technologies into humans and the implications this might have for human evolution.

—Robert R. Hoffman

To be continued …