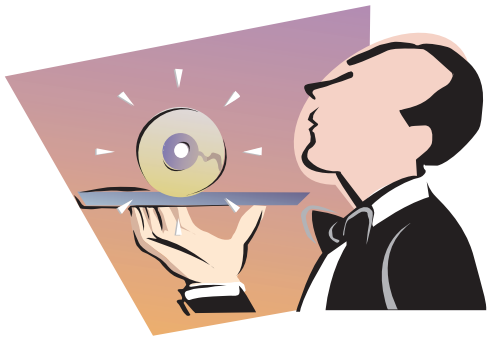
# 14 Principles of Polite Apps



## Software should respond to your obvious needs, not just your commands. Use these 14 principles to create accommodating software.

**by Alan Cooper**

**WHAT YOU NEED**
Visual Basic 6.0

Clifford Nass and Byron Reeves, professors at Stanford University, study people's response to computers. By cleverly repurposing classic experiments in social psychology, they've observed some remarkable behavior. They published their findings in a book titled *The Media Equation* (see the Resources sidebar) and demonstrated conclusively that humans react to computers in the same way they react to other humans.

Nass and Reeves say "people are not evolved to twentieth-century technology," and "Modern media now engages old brains … Consequently, any medium that is close enough will get human treatment, even though people know it's foolish and even though they likely will deny it afterward." To our human minds, computers behave less like rocks and trees than they do like humans, so we unconsciously treat them like people, even when we "... believe it is not reasonable to do so."

In other words, humans have special instincts that tell them how to behave around other sentient beings, and as soon as *any* object exhibits sufficient cognitive friction, those instincts kick in and we react as though we were interacting with another sentient human being. This reaction is unconscious and unavoidable, and it applies to everyone. With profound and amusing irony, Nass and Reeves used many computer science grad students, skilled enough to have coded the test programs themselves, as test subjects. These subjects were highly educated, mature, rational individuals, and they all strongly denied being emotionally affected by cognitive friction, even though the objective evidence was incontrovertible.

Cognitive neuroscientist Steven Pinker, from MIT, corroborates this thesis in his remarkable book, *How the Mind Works*. He says, "People hold many beliefs that are at odds with their experience but were true in the environment in which we evolved, and they pursue goals that subvert their own well-being but were adaptive in that environment."

One important implication of the research is profound: If we want users to like our software, we should design it to behave like a likeable person. If we want users to be productive with our software, we should design it to behave like a good human work mate. Simple, huh?

Nass and Reeves say that software should be "polite" because politeness is a universal human behavioral trait—that is, actions considered polite might vary from culture to culture, but the trait is present in all cultures. Products with high cognitive friction, such as software, should follow this simple lead and also be polite. The manners of many high-tech products suggest that as long as you say "Please" and "Thank you," it's okay to be rude, but that is emphatically not what politeness is all about.

If the software is stingy with information, obscures its process, forces the user to hunt for common functions, and is quick to blame the user for its own failings,

**RESOURCES**

- **How the Mind Works**, by Steven Pinker (W.W. Norton & Company, 1997, ISBN: 0393045358). I absolutely love this wonderful, eye-opening, literate, amusing, readable book. —A.C.)
- **The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places**, by Byron Reeves and Clifford Nass (Cambridge University Press, 1996, ISBN: 1575860538)
- **Accidental Empires: How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can't Get a Date**, by Robert X. Cringely (Addison-Wesley, 1991, ISBN: 0887308554)

the user dislikes the software and has an unpleasant experience. This happens regardless of "Please" and "Thank you." Regardless, too, of how cute, how representational, how visually metaphoric, how content-filled, or how anthropomorphic the software is.

On the other hand, if the interaction is respectful, generous, and helpful, the user likes the software and has a pleasant experience. Again, this happens regardless of the composition of the interface; a green-screen command-line interface is well liked if it can deliver on these other points.

What exactly does it mean for software to be friendly or polite? What does it mean for software to behave more like humans? Used car salesmen wear handsome clothes, smile broadly, and are filled with impressive information, but does that make them likeable? Humans are error-prone, slow, and impulsive, but it doesn't follow that software with *those* traits is good. Human beings have many other qualities that, although present only conditionally, make them well suited to the service role. Most software fills the service role.

Most good software engineers are at a disadvantage in the politeness realm. Robert X. Cringely says that programmers are expressive and precise in the extreme, but only when they feel like it. Their mode of communication is so precise that they can seem almost unable to communicate. Call a nerd Mike when he calls himself Michael and he likely won't answer, because you couldn't possibly be referring to him.

You can see how the concepts of politeness, or even humanness, can be a stumbling block when we ask programmers to interpret such fuzzy concepts. They struggle with this idea of making computers behave more like humans, because they see humans as weak and imperfect computing devices.

I asked my friend Keith Pleas, who is well-known in the engineering community as an articulate, expert programmer sensitive to user interface issues, about making software more human. Keith interpreted adding humanness as adding imprecision to the interaction. He replied:

> *Would a computer "lie" to you? Would a computer say you have "about $500" in your checking account? Would a computer give you a different answer than it just gave someone else? If we enhance the humanness, some of the computer-ness will be reduced, at least in comparison.*

Keith's response is natural from the programmer's point of view. True, the computer would never give you an approximate bank balance, but then the computer wouldn't differentiate between taking 1/10th of a second to say you have "about $500" in your account, versus taking 17 minutes to say you have "exactly $503.47." A really polite, more human program would immediately say you have "about $500," and then inform you it will give you a more precise figure in a few additional minutes. Then it would be *your* choice whether to invest more time for additional precision. That's commensurate effort.

Humans have many wonderful characteristics that make them "polite," but those definitions are fuzzy and imprecise. Here's my list of what improves the quality of interaction, either with a human or a high-tech, software-based product rich in cognitive friction. Polite software:

- is interested in me
- is deferential to me
- is forthcoming
- has common sense
- anticipates my needs
- is responsive
- is taciturn about its personal problems
- is well-informed
- is perceptive
- is self-confident
- stays focused
- is fudgable
- gives instant gratification
- is trustworthy


**Polite Software Is Interested in Me.** A friend would ask about me, and would be interested in who I am and what I like. He would remember my likes and dislikes so he could please me in the future. Any supportive service provider would make an effort to learn to recognize the face and name of her customers. Some people appreciate being greeted by name and some don't, but everyone appreciates being treated according to his own personal tastes.

Most software doesn't know or care who's using it. In fact, none of the *personal* software on my *personal* computer seems to remember either me or anything about me. This is true in spite of the fact that it is constantly, repetitively, and exclusively used by me and no one else. Larry Keeley jokes that the automatic-flush urinal in an airport bathroom is more aware of his presence than his desktop computer.

Every bit of that software should work hard to remember my work habits, and particularly, everything that I say to it. To the programmer writing the program, it's a just-in-time information world, so whenever the program needs some tidbit of information, it simply demands that the user provide it. But the thoughtless program discards that tidbit, assuming it can merely ask for it again if it ever needs it. Not only is the computer better suited to doing the remembering, but it is impolite for it to forget.

For example, there are 11 people named Dave in my e-mail program's name and address directory. I rarely communicate with most of them, but they include my best friend Dave Carlick, to whom I send e-mail all the time. When I create a new e-mail and type an ambiguous "Dave" in the "To:" block, I expect the program to have learned from my past behavior that I mean Dave Carlick. If I want to send

## ABOUT THIS FEATURE

This article is excerpted from **The Inmates are Running the Asylum—Why High-Tech Products Drive us Crazy and How to Restore the Sanity** by Alan Cooper (Macmillan Computer Publishing USA, 1999, ISBN: 0672316498). Reprinted with the publisher's permission.

something to another Dave (David Fore, for example) I'll type in Dave F, D4, David Fore, or something else to indicate my out-of-the-ordinary choice. Instead, the program behaves stupidly, always putting up a dialog box and making me choose which of the 11 Daves I mean. The program just doesn't care about me, and treats me like a stranger even though I'm the only human it knows.


**Polite Software Is Deferential to Me.** Any good service person defers to her client. She understands the person she is serving is the boss, and whatever the boss wants, the boss should get. When a restaurant host shows me to a table in a restaurant, I consider his choice of table to be a suggestion, not an order. If I politely demur and choose another table in an otherwise empty restaurant, I expect to be accommodated immediately. If the host refuses, I am likely to walk out and choose another restaurant where *my* desires take precedence over the host's.

Impolite software supervises the assumed-to-be-incompetent human's actions. It's okay for the software to express its *opinion* that I'm making a mistake, but it is not okay for it to judge my actions. Likewise, it's all right for software to *suggest* that I cannot "Submit" my entry until I've entered my Social Security number, but if I go ahead and "Submit" without it anyway, I expect the software to do as it's told. The very word "Submit" and the concept it stands for are a reversal of the deferential role. The software should submit to the user, and any program that proffers a "Submit" button is *de facto* impolite. (World Wide Web sites, take notice.)


**Polite Software Is Forthcoming.** At the airport, if I ask an airline employee at which gate I can find Flight 79, I would expect him to not only answer my question, but to volunteer the extremely useful collateral information that Flight 79 is also 20 minutes late.

If I order food at a restaurant, it should be obvious that I also want a knife, fork, and spoon, a glass of water, salt, pepper, and a napkin.

Most software won't do this. Instead, it only narrowly answers the precise questions we ask it, and is typically not forthcoming about other information, even if it is clearly related to my goals. When I tell my word processor to print my document, it never tells me that the paper supply is low or that 40 other documents are queued up before me, but a helpful human would.


**Polite Software Has Common Sense.** Although any good restaurant happily lets you tour its kitchen, when you first walk in the front door the hostess' simple common sense directs you to the dining room instead. Most software-based products don't seem to differentiate between kitchen and dining room, putting controls for constantly used functions adjacent to never-used controls. You can commonly find menus offering simple, harmless functions along with deadly, irreversible ejector-seat–lever functions that should be used only by trained professionals. It's like seating you at a dining table right next to the grill. The earlier "about $500" example is a good illustration of putting common sense to work in an interface.

Horror stories abound of customers permanently offended by irrationally rational computer systems that repeatedly sent them checks for $0.00 or bills for $8,943,702,624.23. Most of the customer service nightmares have gone away through the judicious isolation of customers from computer systems, but most employees still have to interact with them. The employees are paid for doing so, so they tend not to complain too loudly, and they typically have no one to complain to; the customer service department is normally not for them.


**Polite Software Anticipates My Needs.** My assistant knows I require a hotel room when I travel to another city to a conference. She knows this even though I don't explicitly tell her so. She knows that I like a quiet, nonsmoking room, too, and requests one for me without any mention on my part. She anticipates my needs.

My Web browser spends most of its time idling while I peruse various Web sites. It could so easily anticipate my needs and prepare for them instead of just wasting time and effort. Why can't it use that idle time to preload links that are visible? Chances are good that I will soon ask the browser to examine one or more of those links. It's easy to abort an unwanted request, but always time-consuming to wait for a request to be filled. If the program were to anticipate my desires by getting prepared for my requests during the time it would otherwise be idling, waiting for my commands, it could be much more responsive without needing a faster modem.


**Polite Software Is Responsive.** When I'm dining in a restaurant, I expect the waiter to respond appropriately to my nonverbal cues. When I'm engaged in intense conversation with my tablemates, I expect the waiter to attend to other duties. It would be inappropriate for the waiter to interrupt our discussion to say, "Hello, my name is Raul, and I'll be your waitperson for the evening." On the other hand, when our table conversation has ended and I am swiveling my head and trying to make eye contact with Raul, I expect him to hustle over to my table to see what I want.

My computer normally runs in a video mode that gives me 1024-by-768 pixels on screen. When I do presentations, I am required to change temporarily to 800-by-600 pixel mode to accommodate the lower resolution of my video projector. Many of the programs that I run, including Windows 95, react to the lowered resolution by changing their window size, shape, and placement on the screen. However, I invariably and quickly change my computer back to 1024-by-768 pixel mode. But the windows that changed to accommodate the lower resolution don't automatically change back to their previous settings for the higher-resolution screen. The information is there, but the program just doesn't care about responding to my obvious needs.


**Polite Software Is Taciturn About Its Personal Problems.** In saloons, salons, and psychiatrist offices, the barkeep, hairdresser, and doctor are expected to keep mum about their problems, and to show a reasonable interest in yours. It might not be fair to be so one-sided, but that's the nature of the service business. Software, too, should keep quiet about its problems and show interest in mine. Because computers don't have egos or tender sensibilities, they should be perfect for the role of confidant, but they typically behave the opposite way.

Software is always whining at me with confirmation dialog boxes

***Continued on page 69.***

*Continued from page 64.*

and bragging to me with unnecessary little status bars. I don't want or need to know how hard the computer is working. I'm not interested in the program's crisis of confidence about whether to purge its recycle bin. I don't want to hear its whining about not being sure where to put a file on disk. I don't need to hear the modem whistling or see information about the computer's data transfer rates and its loading sequence, any more than I need information about the bartender's divorce, the hairdresser's broken-down car, or the doctor's alimony payments.

Two issues lurk here. Not only should the software keep quiet about its problems, but it should also have the intelligence, confidence, and authority to fix its problems on its own.


**Polite Software Is Well-Informed.** On the other hand, we all need more information about what's going on. That same barkeep helps me by posting his prices in plain sight on the wall, and writing on the chalkboard what time the pregame party begins on Saturday, along with who's playing and the current Vegas spread.

Shopkeepers need to keep their customers informed of issues that might affect them. I don't want my butcher to tell me on November 21 that he is out of Thanksgiving turkeys. I want to know well in advance that the supply is limited and that I need to place my order early.

When I search a topic on the Web using a typical search engine, I never know when link rot will make the engine's findings useless. I'll click on the URL of something I'd like to see, only to get a nasty "404 Link Not Found" error message. Why can't the engine periodically check each link to see if it still exists? If it has rotted away, the useless entry can be purged from the index so I won't waste my time waiting for it.

Programs constantly offer me choices that, for some reason, are not currently available. The program should know this and not put them in front of me.


**Polite Software Is Perceptive.** The concierge at a hotel I frequent in New York noticed my interest in Broadway shows. Now, whenever I visit, the concierge—without my asking—puts a handy listing of the current Broadway shows in my room. She was perceptive enough to notice my interest, and this allows her to anticipate my desires and provide me with information I want before I even think about it. It takes little effort for the concierge to exploit the value of her acute perceptions, yet it draws me back to this hotel again and again.

Whenever I use an application, I maximize it to use the entire screen. I then use the Windows Start bar to change from one program to the other. But the applications I run don't seem to notice this fact, especially new ones. I frequently have to tell them to maximize themselves even though they should be able to see that my preference is clear and unequivocal. Other users keep their applications in smaller windows so they can see icons on their desktop. This is just as easy for software to spot and anticipate.


**Polite Software Is Self-Confident.** I expect the service people I interact with to have courage and confidence. If they see me emerge from the men's room with my fly unzipped, I want someone to tell me quickly, clearly, and unobtrusively before I walk into the ballroom to give my speech. It takes some courage to do this, but it's courage appreciated. Likewise, if my assistant can't book me the flight I want, I expect him to confidently book something close to the one I want without bothering me with details.

If I tell the computer to discard a file, I don't want it to come back to me and ask, "Are you sure?" Of course I'm sure, otherwise I wouldn't have asked. I want it to have the courage of its convictions and go ahead and delete the file.

On the other hand, if the computer has any suspicion that I might be wrong—which, of course, is always—it should anticipate my changing my mind and be fully prepared to undelete the file. In either case, the product should have confidence in its own actions, and not weasel and whine, passing the responsibility off onto me.

I have often worked on a document for a long time, pressed the Print button, and then gone to get a cup of coffee while it prints out. Then I return to find a mindless and fearful dialog box quivering in the middle of the screen asking me, "Are you sure you want to print?" This insecurity is infuriating, and the antithesis of polite human behavior.


**Polite Software Stays Focused.** When I order salad in a good restaurant, I get a good salad. In a bad restaurant, I get the third degree along with it: "Spinach, Caesar, or mixed greens? Onions? Croutons? Grated cheese? Parmesan or Romano? Full serving or dinner size? French, Italian, oil and vinegar, or Thousand Island? Dressing on the side? Served before or after the main course?" Even the most demanding gourmet just doesn't care that much about the salad to be subjected to such a grilling, but interactive systems behave this way all the time. Adobe's Photoshop program is notorious for peppering the user with lots of obnoxious and unnecessary little questions, each in a separate dialog box.

Impolite software asks lots of annoying questions. Choices are generally not all that desirable, and being offered them is not a benefit, but an ordeal.

Choices can be offered in different ways, too. They can be offered in the way that we window shop. We peer in the window at our leisure, considering, choosing, or ignoring the goods offered to us. Alternatively, choices can be forced on us like a hostile interrogation by a customs officer at a border crossing: *"Do you have anything to declare?"* with the full knowledge that we can dissemble as much as we like, but the consequences for getting caught can be more than just embarrassing. We don't know the consequences of the question. Will we be searched or not? If we know that a search is unavoidable, we would never lie. If we know there will be no search, we would be tempted to smuggle in that extra carton of Marlboros.


**Polite Software Is Fudgable.** When manual information processing systems are translated into computerized systems, something is always lost in the process. Manual systems are typically computerized to increase their capacity, not to change their functionality. But manual systems are typically flexible, which is not a function that can easily be isolated. While an automated order-entry system can handle millions more orders than a human clerk can, the human clerk has the ability to *work* the system.

In an automated system, the ability to work the system disappears. There's almost never a way to jigger the functioning to give

or take slight advantages.

In a manual system, when the clerk's friend from the sales force calls on the phone and explains that getting the order processed speedily means additional business, the clerk can expedite that one order. When another order comes in with some critical information missing, the clerk can go ahead and process it, remembering to acquire and record the information later. Typically, this flexibility is absent in computerized systems.

Computerized systems have only two states: nonexistence or full-compliance; no intermediate states are recognized or accepted. Any manual system has an important but paradoxical state—unspoken, undocumented, but widely relied upon—of *suspense*, wherein a transaction can be accepted while still not being fully processed. The human operator creates that state in his head or on her desk or in his back pocket.

For example, a digital system needs both customer and order information before it can post an invoice. While the human clerk can go ahead and post an order in advance of detailed customer information, the computerized system rejects the transaction, unwilling to allow the invoice to be entered without it.

I call this human ability to take actions out of sequence or before prerequisites are satisfied "fudgability." It's typically one of the first casualties when systems are computerized, and its absence is a key contributor to the inhumanity of digital systems. It's a natural result of the implementation model. The programmers don't see any reason to create intermediate states because the computer has no need for them. Yet humans need to be able to slightly bend the system.

One of the big benefits of a fudgable system is the reduction of mistakes. By allowing many small temporary mistakes into the system and entrusting the human to correct them before they cause problems downstream, much bigger, more permanent mistakes are avoided. Paradoxically, most of the hard-edged rules enforced by computer systems are imposed to prevent just such small mistakes. These inflexible rules cast the human and the software as adversaries, and because the human is prevented from fudging to prevent big mistakes, he soon stops caring about protecting the software from really colossal problems. When inflexible rules are imposed on flexible humans, both sides lose. It's invariably bad for business to prevent humans from doing what they want, and the computer system usually ends up having to digest invalid data anyway.

Fudgability is one of the few human politeness traits that can be difficult to build into a computer system. Fudgability demands a much more capable interface. In order to be fudgable, systems have to reveal their process to the moderately skilled observer. The clerk can't move a form to the top of the queue unless the queue, its size, its ends, the form, and its position can be easily seen. Then the tools for pulling a form out of the electronic stack and placing it on the top must be present. These have to be made as visible as they are in a manual system, where it can be as simple as moving a sheet of paper. Physically, fudgability requires extra facilities to hold records in suspense, but an undo facility has similar requirements. The real problem is that it admits the potential for fraud and abuse.

Fudging the system can be construed as fraud. It's technically a violation of the rules. In the manual world, fudging is tacit and winked at. It is assumed to be a temporary, special case, and the fudger will tidy up all such accounts before leaving for the night, vacation, or another job. Certainly, all such examples are cleaned up before the auditors are allowed in. If this process of temporary rule suspension were well known, it might encourage people to use the technique to the point of abuse.

Especially if fudging has been documented in the company manual, investing it with respectability, those with weaker characters might see in it a way to avoid doing accurate and complete work, or they might see in it a way to defraud the company of money. It's not fiscally responsible for the company to support fudging.

But fudgability has a powerful effect on the way users regard the system. All the reasons for not having a fudgable system are rational and logically defensible—and probably legally defensible, too. Unfortunately, this idealized state of affairs is simply not an accurate description of the way the world works. Everyone in all areas of business utilizes the fudgability of manual systems to keep the wheels of business—of life—greased and turning easily. It's vital that automated systems be imbued with this quality despite the obstacles.

The saving grace for abuse is that the computer also has the power to easily audit all of the user's actions, recording them in detail for any outside observer. The principle is a simple one: Let the user do whatever he wants, but keep detailed records of those actions, so that full accountability is easy.


**Polite Software Gives Instant Gratification.** Computer programming is all about deferred gratification. Computers do nothing until you've put enormous effort into first writing a program. Software engineers slowly internalize this principle of deferred gratification, and they tend to write programs that behave in the same way. Programs make users enter all possible information before the programs do even the tiniest bit of work. If another human behaved that way, you'd actively dislike them.

We can make our software significantly more polite by assuring that it works for and provides information to the user without demanding a lot of up-front effort.


**Polite Software Is Trustworthy.** Friends establish trust with one another by being dependable and by a willingness to give of themselves. When computers behave erratically and are reluctant to work for users, no trust is generated. Whereas I trust the bank teller because she smiles at me and knows my name, I always count my cash at the ATM because I simply don't trust the obtuse machine.

Our software-based products irritate us because they aren't polite, not because they lack features. As this list of characteristics shows, polite software is usually no harder to build than impolite software. It simply means that someone has to envision interaction that emulates the qualities of a sensitive and caring friend. None of these characteristics is at odds with the other, more obviously pragmatic goals of business computing. Behaving with more humanity can be the most pragmatic goal of all. VBPJ

## About the Author

Alan Cooper is known as "The Father of Visual Basic" for his invention of the visual programming interface that became VB. Reach Alan at alan@cooper.com.