DOI:10.1145/2811280

Vinton G. Cerf

On (Computing) Artifacts

The term *artifact* has more than one interpretation. *Moiré patterns* are artifacts produced when two very similar patterns, usually straight-lined, are overlaid and one

desires to exercise that software. There are times when using so-called computer tools such as text processing programs that I wish I could go back to typewriters that put the keys where I position the paper rather than trying to coerce an uncooperative text processing program to center, indent, or place text where I want it on the page. Of course, I would lose an enormous amount of flexibility including the possibility of adjusting text font and size automatically or reformatting the document on the fly.

For any significant piece of software, design for usability takes a good deal of thought, especially when considering how its functionality is to be made apparent to a user with a visual, aural, or motor problem requiring the user interface to adapt or be adaptable to accommodate. It seems to me these considerations often do not get the attention they should in the early stages of program/application design. Stanford University has the *d.school*, which is shorthand for the Institute of Design. There are similar initiatives elsewhere; I recently discovered a related program at UC San Diego.

If we are really serious about making software accessible and usable, we are going to have to recognize this is a problem related to the design of artifacts and that our intuitions about design must be rooted in real knowledge of how software is used by people needing assistive consideration. Testing of interface ideas by people using assistive techniques regularly seems a critical part of solving this problem. Blind-folding an otherwise sighted programmer and asking her to try out her code does not produce an experience adequate to the task (although it may well be, er, eye-opening).

Just as Herb Simon recognized so many decades ago, we really need to derive deep understanding of the nature and behavior of the artifacts we create before we can successfully design them to perform in predictable or at least manageable ways.

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

is slightly rotated. *Pixelation*, for example, occurs when resolution is lacking in a display. The interpretation here, however, focuses on the creation of man-made things (for example, software, tools, and computers).

Nobel Prize winner Herbert Simon wrote *The Sciences of the Artificial* 46 years ago. The third edition, published almost exactly 20 years ago,^a still holds a great deal of meaning today. Computers and computer programs are *artifacts*. That is, they are man-made constructs and, in theory, they should be more understandable than natural phenomena for which there are no guaranteed laws or rules, only approximations that help us understand or predict behavior.

Computers and, especially, computer programs may be among the least well understood artifacts known to mankind. Computers themselves may have sufficiently constrained behaviors that we may be fairly confident as to how they behave (instruction sets, I/O behavior). Linking them in networks may make their aggregate behavior less clear but that is partly because this behavior is derived from *software* that seems to have no serious constraints on the variations it can manifest. Moreover, most software today is mapped from some programming language into machine language, so the mapping may not always produce results we can anticipate even if the program appears to be straightforward as expressed in its higher-level form.

Making things even more complicated, most software runs in the context of other software such as an operating system that mediates access to the resources of the computer. In a networked environment, programs that express *protocols* for communication may themselves produce anomalous (that is, unexpected) behavior because of our inability to adequately analyze the potential state space the protocols can generate. When two computers interact over the Internet, they can be thought of as two bags full of varied software that might never have interacted before and thus produce unpredictable results. When all these interactions are mixed together in the global Internet, it just seems as if there are an uncountable number of ways in which things might unfold.

If we care about predictable behavior or at least behavior that might be constrained into a predictable envelope, we are going to have to think hard about *design*. Design is one way to think about bounding behavior—“constrained by design” seems like an attractive phrase.

Users are generally not too fond of unpredictable behavior. Perhaps there are some forms of entertainment in which unpredictability has value, but for programs we rely on daily, most of us would prefer they work as advertised, for the user interfaces to remain relatively stable, and to be alerted if it appears things are not going to unfold as expected.

The more I think about software *usability*, the more I am inclined to focus on the underlying design of the software and, especially, the means by which users can communicate their functional

^a Hardcover ISBN: 9780262193740 (Sept. 1996),
Paperback ISBN: 9780262691918 (Sept. 1996)