Title
Cautionary Aphorisms for Customer-Oriented System Development

Permalink
https://escholarship.org/uc/item/4bq2s63

Author
Stevens, D.F.

Publication Date
1994-10-01
Cautionary Aphorisms for Customer-Oriented System Development

D.F. Stevens

October 1994
DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, or The Regents of the University of California.

Lawrence Berkeley Laboratory is an equal opportunity employer.
DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.
Cautionary Aphorisms for Customer-Oriented System Development*

David F. Stevens
Information and Computing Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

October, 1994

Prepared for
ACR's 1995 Conference on Improving Productivity in System Development

Mesa, Arizona, 6-10 February, 1995

* This work was supported by the U. S. Department of Energy under contract No. DE-AC03-76SF00098.
Cautionary aphorisms for customer-oriented system development

D. F. Stevens
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Of the three substantive two-word clusters in the title of this note the least important is "system development". Although people who attend conferences like this may be interested in "system development", few other people are. Your customers, in particular, are much more interested in system use, system availability, and system effectiveness, than in system development. Because it is your customers who keep you in business, it therefore behooves you to develop some techniques to keep use, availability, and effectiveness in mind—in other words, to develop a customer orientation. This note illustrates one such technique: the distillation of applicable wisdom concerning people, systems, and the relationships between them into statements made memorable by their wit, brevity, and pungency. In this session I hope to start you on the way to your own collection of useful thoughts. You should note that the preparation of such a collection serves as a working demonstration of the value of re-use as a productivity principle: few of these statements are original with me, and many had origins in other applications.

The cautionary aspect of the truths presented here is emphasized for two reasons: because many of them run counter to much of what you are told in the system development trade press, and because many of them are, in fact, warnings.

We shall consider in turn methodology and procedure, design, schedules, measurement (a recurring favorite of mine), documentation, and quality.

On Methodology and Procedure

MP1: Getting it right is better than doing it right.

MP2: Getting it right is better than getting it in writing.

MP3: Procedures are founded on lack of trust and are generally designed (or used) to shield everybody from accountability.

MP4: The objective of methodology and procedures is control, not productivity.

MP5: Procedures are like railroads: easy to use and efficient...if they go where you want to go and stop where you want to stop.

MP 1, 2, and 4 are concerned with the control aspects of methodology. I claim that we cannot make effective use of procedures and methodology until we recognize that their very purpose is control; that productivity improves (if it does) is but a serendipitous by-product. Understanding this is one of

* This work was supported by the U. S. Department of Energy under contract No. DE-AC03-76SF00098.
the necessary steps in the journey from success in school to success in the real world. Perhaps one of the reasons that re-use is so slow to achieve acceptance is that in school we learned to call it “cheating”.

In school, the correct result is often less important than the correct method. A good example of this is the algebraic method, to which most of us were introduced in our first course in plane geometry. (That is where we learned to disregard what we knew (about the real world everywhere) and to limit our “facts” to what we had proven (about the idealized Euclidean world).) As a reader in college calculus courses I frequently had to give zero credit for the right answer arrived at through a faulty method, and near top credit for the wrong answer, if the approach was correct. When the objective is customer satisfaction, however, rather than a Euclidean proof, the result is more important than the method. The best technique in the world doesn’t compensate for doing the wrong application.

One of the standard features of many methodologies is the formal sign-off procedure. This is an extension of the modern American tendency to replace trust with contracts, responsibilities with entitlements, and accountability with legal shields. Getting it in writing, whether “it” is a requirements statement or a design approval, is often an exercise in anticipatory blame shifting. For lengthy projects, it is also a guarantee of obsolescence, in that the requirements were fixed in yesterday’s environment; it is an implicit statement that the implementor’s intellectual engagement is not wanted, and it is an explicit removal of all flexibility from the process. In all these ways it is inimical to quality and productivity.

Quality and productivity reflect the manner in which you meet the customer’s current needs—neither the requirements you persuaded him to settle for when the product was still undergoing definition nor the requirements he really thought he had then, but the requirements he has now and will have tomorrow, whether he recognizes it or not. Few businesses are so static today that the creations of a traditional lock-in-the-requirements methodology are still what is needed when the products actually become available. Rather than locking in requirements you should strive to lock in a relationship so that whatever your customer learns about his changing environment and changing needs can be incorporated into today’s dynamic design...even if it will be scrapped tomorrow in the light of new information. This of course, demands flexibility on the part of “methodology” and staff alike. A procedure that punishes a developer for providing what the customer needs instead of what he signed for cannot result in satisfactory products.

Procedures are inherently straitjackets, designed to remove doubt, thought, initiative, and creativity from the process. Procedures are great when the job is inherently mindless and routine, but system development should be neither mindless nor routine. Procedures work well when the objective is to stay on a familiar track or within a known framework, but they are useless when you have to operate outside the frame. So choose and apply your methodologies and procedures with care and remember that neither the train nor the track is as important as the destination.

On Design and Implementation

DI 1: Know your customers.
• Users know more about their working environment than designers and implementers do.

• Provide expected consistency.

• Speak to the user in user language. Listen to the user in user language.

DI 2: Know your platform.

• Stay within striking distance of the state of the art.

• The limit of user patience is somewhere around 3.0 seconds (and much less in some circumstances).

DI 3: Systems work as implemented, not as designed.

• Progressive improvement is better than postponed perfection.

• Eliminate rough edges; solve your own implementation problems.

• Don’t invite users to try things that don’t work.

• Half a good idea is more irritating than none.

DI 4: Satisfaction is not sufficient.

• Even good systems can be improved. (Don’t expect the user to quit complaining just because it’s a good system.)

• Even bad systems can be used. (Don’t assume that use indicates satisfaction.)

Many of these precepts are so obvious that you must be wondering why they are included. Well, it’s because they are still so frequently ignored or violated. Designers, for instance, still tend to consider that their colleagues are reasonable surrogates for real people, so rather than trying out their new ideas on the marketing staff or the clerks, they try them out on the designer next door. The result is systems with—for example—fill in sequences that make logical sense but do not conform to the order in which information is usually given over the phone. (Anyone who has tried to use a keyboard arranged in alphabetical order knows that pure logic can fail when it is opposed by common expectations.) Or calendaring systems that require all meeting attendees to acknowledge the addition of a new invitee. Or error-handling systems that prevent you from correcting the error until you’ve acknowledged the error message.

The question of consistency is a particularly tricky one. Jonathan Gruder (CACM, October, 1989) points out that while it is a criterion beloved of designers, it can drive users crazy if they happen to
be on a different wavelength.\(^1\) (We must always remember Emerson.\(^2\)) For example, convenience might dominate consistency when deciding on mode of input (mouse, keyboard, touch screen, etc.), and brevity might dominate consistency of vocabulary when designing an expert interface. In all cases, however, it is only user-perceived consistency that matters for user satisfaction: consistency of design is insufficient if it does not result in desirable, perceived consistency in use, and is actively damaging if it interferes with ease of use.

With respect to language, the admonition to speak in user language is a familiar one, but its companion is equally important and far more often ignored. It does little good to speak in user language if you then require the user to speak to you in computerese. A large component of the success of the graphical user interface is that it allows the user to eschew technobabble, and thus continue to think and work in his own language.

It is as important to know (the limitations of) your platform as it is to know (the desires and capabilities of) your customers. More than one apparently excellent design has failed miserably in implementation because the platform didn't match the expectations of the designers in speed or features. You must remember that the user patience is a fleeting thing, rarely lasting even as long as 3 seconds—and that's for complex operations: the trivial should be instantaneous: any perceptible delay is intolerable.

We are now well launched into the information era. Most of the users of the systems you are about to design have some experience with computers and systems in other venues. Many of them are experienced game players. Successful games have attractive interfaces, tailored to the specific situation. These users know what is possible; they are going to be increasingly dissatisfied if your systems fail to match the power and convenience available to their pre-school and teen-aged children.

We have alluded briefly to the too-frequent disconnect between design and implementation. Napoleon's expression of this principle is perhaps the most colorful: "No battle plan ever survived contact with the enemy." Those who would be designers must always remember that systems work as implemented. As John Gall has noted (in Systemantics), systems develop goals of their own that are not necessary consonant with the goals of their designers. He should have gone on to note that realization of the design may not even be the principal goal of the developers.

I remember one computer-aided tutorial that requested the student to press <CR> after each of the first 23 steps, but asked specifically that <CR> not be pressed after the twenty fourth. Those who preferred to proceed by the experimental method saw the system crash when they pressed <CR>. This is a rather blatant instance of the attractive nuisance. Other examples abound, especially those that serve as placeholders for features or functions required by the design but not realized in the

\(^1\)"Focusing on consistency may encourage the false hope that good design can be found in properties of the interface."

\(^2\)"A foolish consistency is the hobgoblin of little minds."

-4-
implementation. There may be some cases where this is unavoidable, but presenting "active" menu items or soft keys that serve only to tell the would-be users that "This function is not available" does not seem to share this necessity.

System developers have a responsibility to eliminate rough edges and to avoid burdening users with their implementation problems. They fail in this last most frequently in their succession planning, often requiring users to go through explicit conversion exercises. Rough edges, of course, are interface awkwardnesses with which a user can inflict self-damage. The most notorious of these was probably the UNIX `rm *` command: five keystrokes to wipe out a life's work!

Rough edges often come from paying too much attention to the familiar half of "something is better than nothing", namely that something usable today is better than something better promised for tomorrow. The problem is that half a good loaf can leave you less satisfied than before. The half you see whets your appetite for more, and the dashing of your higher expectations can be more frustrating than complete absence.

The last word in this section is that satisfaction is a slippery beast. You can't assume that use implies satisfaction: people will use bad systems if that is all they have. And you can't assume that satisfaction implies the absence of dissatisfaction. "Satisfaction" is a relatively modest aspiration, after all; you can't expect the user to quit complaining just because it's a good system. As the idea of quality takes hold, you can expect that all competitive systems will provide delight. (Just remember: today's delight is tomorrow's expectation, and next week's insufficiency.)

On Schedules

S1: Hofstadter's Law

S2: Development time starts with the realization of need, not the project.

S3: More accurate estimates aren't necessarily better estimates.

S4: Political estimates take precedence over technical estimates.

Perhaps the most useful thing one can say about schedules is the sooner you fall behind, the more time you'll have to catch up. Because you will fall behind—Hofstadter's Law\(^3\) takes care of that. Brooks (in *The Mythical Man Month*) had the most definitive comment on scheduling and what to do about it when he noted that adding more people to a late project makes it later. The problem is that schedules are too often established through political imperative rather than through any sensible technical process. To some extent, that is the way it should be, because the political imperative is coupled to the cumulative dissatisfaction that begins building as soon as the need is felt, thus contributing to the creation of the project, whereas the technical process, by definition, doesn't come into play until after the project has been created. The effects of political imperative are most visible in two important queue server disciplines that are generally overlooked in the technical literature.

\(^3\)"It always takes longer than you think, even when you take Hofstadter's Law into account."
You are familiar with FIFO and LIFO; the two I refer to are BIFO and FISH: BIFO, of course, is "biggest in, first out"; you ignore it at your peril. (FISH ("first in, still here") often co-exists with BIFO, for obvious reasons.)

The comments on accuracy of estimates could go equally well in the next section (on measurement), but it probably applies more strongly to schedules than to other attributes of system development. Rewarding accurate estimation is an invitation to over-estimate and then execute the Parkinsonian expansion. This should not be interpreted as an attack on accuracy in estimation, but as a recognition of the possibility of the cynical manipulation of measure to advance one's own interest to the detriment of that of the customer.

**On Measurement**

**M1:** Measurement isn't management.

**M2:** Measurement is often obfuscatory.

In America today one often reads something to the effect that we cannot manage what we cannot measure. The easily led allow that reasonable (incorrect; but reasonable) statement to be perverted into an unspoken belief that the very act of measuring something constitutes management. In one sense, of course, it does, for measurement changes behavior so as to encourage the optimization of the attribute measured. But unless the entire suite of measures is chosen, and the goals are adjusted, with some specific objective in mind, we are counting angels on pinheads. A measurement program can be a valuable part of your management effort, but you should be quite clear in your own mind that management is in what you choose to measure and what you do with the measurement and a whole lot about judgment in general; it does not reside in the measurements themselves.

Measurement is a tool, and like any tool it can be abused. One of the most prevalent abuses is the use of measurement to obfuscate what is really happening, rather than to clarify it. I have written at length on this topic and do not wish to repeat those earlier papers in detail; suffice it to say that many of the most popular measures are obfuscatory—not necessarily from evil intent, but because the concepts are fuzzy, the application is inconsistent, and—most importantly—we really don't know what they mean. We have seen lines of code, function points, complexity, software "work", and other measures created in the effort to turn software development from an "art" into a "science", where by "science" is meant a known discipline with known ways of finding solutions to whatever problems arise. Metrics are a diversionary tactic used to hide the fact that we don't know what we're doing, and those who do it well can't really explain how they do it. There is no royal road through measurement to management of systems development.

**On Documentation**

**D1:** It's easier to fix the manual than the problem.

**D2:** You don't turn an inferior product into an acceptable one by documenting its idiosyncrasies.
D3: If it needs documenting it’s not user friendly.

D4: People who write programs don’t think like people who use computers.

D5: Automated advice always answers the wrong questions.

One of the more persistent system development myths is that good documentation is the key to system usability. (I have contributed to the early strength of this myth, in fact, but I believe I know better now.) In fact, the inessentiality of documentation is a better indication of usability. The point is not to avoid documentation, but to write systems that don’t need it. The nub of the problem has been noted before, and here restated as D4: systems developers don’t think like users. This is as true of those who write conventional documentation as it is of those who write programs. As phrased by one frustrated user, “The people who write computer documentation are the same people who index the yellow pages and make announcements in bus stations.” (The phenomenon is not limited to computer systems; the interested reader is directed to The psychology of everyday things, by Donald Norman, wherein the unintuitive design of such commonplace devices as doors is taken severely to task.)

Some recent systems have attempted to finesse the problem by providing on-line “help” systems. But here, too, we run into the cultural, psychological, and philosophical differences between system writers and system users. On line help is written from the point of view of the system, and will direct you to a course of action that results in some acceptable—to the system—situation. But that is (a) almost invariably a situation other than the user had in mind, and (b) not the way the way the user wants to get there. Users are more interested in following their own lines of reasoning than in following pre-programmed advice, however sound it may be.

On Quality

Q1: Value exists only in the mind of the customer.

Q2: Defects are in the eye of the beholder.

Q3: Just because it works doesn’t mean it’s doing the right things.

These are three variations on the same theme, namely that it is not up to the system developer to define quality. You may have a “quality” process (by some definition) that delivers a “quality” product (matches specifications, on time under budget, works the first time) that doesn’t meet the user’s needs. (Remember the Edsel and the “new” Coca Cola.) The user’s decision is the one that counts, and that’s all that needs to be said.

In Conclusion

I have not told you how to be successful in system development, but I have indicated how you should measure success; to paraphrase Dean Witter: one customer at a time. I wish you luck; you’re going to need it.