Biography and Proposal
by
Bertram Herzog

University of Michigan, Ann Arbor
and
Fraunhofer CRCG
Providence, RI

Bio

This short bio should suffice for my contribution. As a late joiner to this workshop I appreciate being included. Originally I requested to be "a fly on the wall" to observe this workshop. Why? I am charged with organizing a similar "Challenge" but focused on a derivative of Computer Graphics. My attendance is contingent upon presenting a proposal. Having the advantage of reading early submittals I am encouraged by the fact others seem to have similar concerns.

My training was in physics and engineering mechanics beginning in 1945 at Case Institute of Technology (now part of Case Western University) and to be followed by obtaining the Ph.D. at the University of Michigan in 1961. In between I practiced structural engineering.

My first computer experience was using analog computers during my masters degree. Digital computing came through working on the Ph.D. dissertation. In 1963 I took a leave from my professorial position at Michigan to manage a department in Product Research at Ford Motor Company. Circumstances put me in touch with the field of Computer Graphics and Professor Steven A. Coons at MIT. I have been active in that field ever since. Along the way I directed the building of an early computer network (the MERIT network in Michigan), directed a computing center (University of Colorado), consulted, was a founder of a startup in CAD, and returned to Michigan as part of the Information Technology Division. After retirement in 1992 I resumed teaching and consulting. In 1998 I joined the Fraunhofer Center for Research in Computer Graphics as Vice President and Chief Operating Officer. Today I remain as the President's Deputy.

Some long time ago I received a NSF Fellowship without which the Ph.D. might never have been achieved. I am a Fellow of ACM and will receive the SIGGRAPH Outstanding Contributions Award in July 2002.

My experience with systems engineering is thus peripheral but I want to offer a challenge not so much in research directions but a challenge in engineering practice for computer systems engineering.

Challenge

The challenge is to achieve the level of Professional Engineering responsibility in "computer systems" as can be found in traditional product engineering professions We should expect well engineered software. We should expect well engineered hardware. In other words, we should expect good system engineering combined with marketing strategies. May we ask for error-free systems? Certainly. Will we get them? Of course

not. Absolute freedom of error is likely not possible However, the current level of awkward user interfaces and functional application interfaces combined with numerous bugs is not acceptable.[1]

This point is often disputed by defenders of current products – erroneously in my opinion. The cause of the issue, in my judgment, comes from the lack of application of sound engineering principles. Rather than adhering to clear engineering principles we have obtained products with confusing architectures and an excess of features. A lot of this arises from a disproportionate adherence to marketing input without the requisite balance of engineering. The very popular word-processor on which this essay is being prepared epitomizes this malady. All too often the word processor wants to help me with features for which I have no use and for which I can not find an easy way to escape.

There is no point to harangue more on this matter. The need for this challenge is recognized by many. The solution requires a major shift in practice. The solution, however, is not much fun compared to intriguing new research directions. However, not tackling the situation is an abrogation of fundamental product engineering responsibilities. May this workshop at least recognize this product engineering challenge while it is sorting through the many interesting proposals for research challenges.

Attachment

The Future and its Enemies[?]

by

Bertram Herzog
University of Michigan
Ann Arbor, MI, USA
and
Fraunhofer CRCG
Providence, RI, USA

Introduction

It is generally accepted that the current state of information technology, especially as epitomized by its most widely distributed products, is less than optimal if not down-right bad. This essay seeks to offer a proposal for altering the situation and hopes that colleagues at this workshop will add their insights either to reject the initial assumption (unlikely) or, more likely, to refine or replace the proposal.

Today we have enormous computing power at our fingertips and at consumer and disposable prices. This workshop's colleagues, I expect, will provide their visions, inventions, and points of view that will extol a glorious vision of the future and even

[1] This point is pursued in an essay, published elsewhere, entitled "The Future and its Enemies." A copy is attached for the curious.

[?]Postrel, Virginia, The Future and its Enemies, The Growing Conflict Over Creativity, Enterprise and Progress, The Free Press a division of Simon and Schuster, New York, NY, USA, 1998.

greater advances to come. Nevertheless, I feel it is necessary to examine the current situation lest we build the next generation of products on the shaky foundation of current products.

At this point, I want to mention the "Principle of Least Astonishment." Simply put, it states one should not be astonished by the results obtained from a computer. This statement, originally offered in the good old batch processing days surely applies more urgently in today's highly interactive computing world. The principle needs to be applied, in my opinion, as a test for all computer applications and innovations.[2] On that basis alone, I believe current products are shaky. This conclusion is emphatically supported by Norman in his book "The Invisible Computer[3]" wherein he identifies the folly of ever more technopower, more megabytes and more megahertz, rather than attending to user needs or requirements. Norman identifies a solution that favors the notion of limited and specific task-oriented information appliances. I want to propose another, though complementary, solution. The solution will specify architectural considerations and design and manufacturing process matters.

The Complaint or Problem

So what is the complaint? What we deliver as information products are of alarmingly low quality. I use many of the so-called productivity tools on a regular daily basis. I hardly ever write with pen or pencil any more. My lap top computer and its word processor, the spreadsheet program, several database products, and the slide maker might just as well be attached via my umbilical cord. I can more easily do without my automobile than do without my laptop or desktop computer. However, the violations of the Principle of Least Astonishment alone descend upon me with astonishingly high frequency. The surprises are never ending. Worse, however, are the occasions when what should be conceptually and easily satisfied eludes the user trying to ferret out the "how to" from the user interface, from the manual if it exists or is incomplete, or from the help file.

Like so many others I use these products because they help me do my daily work. Like so many others I can use trial and error to solve problems. Electronic spreadsheets are the liberators from the green eye shades and the paper spread sheets. The improved productivity is real. To dampen this enthusiasm we can point out, all too often, that it takes dedicated persistence to climb the learning curve when easy and intuitive methods should be available. The marketers can demonstrate that millions of happy users are using these tools. My conjecture is that these millions each are using less than five percent of what is provided. Do they need more? Would they use more if made readily available? Good questions. We can invoke the 80-20 rule. Eighty- percent probable don't need more. The other twenty- percent would like to use more if they could discover what is available or, if known to be available, if they could find out how to use more. When I claim that most people use less than, say, five percent of the potential I must hasten to add it is not the same five percent.

Early instantiations of word processors were simple and the interface presented itself accordingly. For example, the taxonomy of a document could be: a document consists of sections, or chapters; sections consist of paragraphs; paragraphs could be styled and each style could be named and have its attributes specified. Documents could be presented on pages whose margins were specifiable. Each element of this

[2] Need one mention the blinking interface of VCRs?

[3] Norman, Donald A., The Invisible Computer, The MIT Press, Cambridge, MA, USA, 1998.

taxonomy could be clearly and reasonably independently described and specified. Interaction between these elements could be specified so that widows and orphans could behave in a specified manner – if not always in the expected manner in the execution! A good taxonomy would be implemented using object-oriented principles. Paragraphs would be a member of an object class that would include tables and graphical objects. We need not mention the concept of pointers to embrace the advantages of hypertext. So much for simple principles upon which to build more complex and specialized (for the local knowledge pool and needs) custom applications.

Instead, we now have feature bloat. Word processors, for example, have evolved into feature-laden applications (see Figure 1). Norton cites that Microsoft Word, by 1992, had 311 commands, but five years later, the count had reached 1,033 commands. That is a lot of commands to explore for their possible utility. Each feature useful for some specialized purpose is used to enlarge the pool of non-overlapping five percenters. However, if one of these initially satisfied users wants to venture toward another use then these unprincipled features refuse to bend, intuitively or via documentation, to meet that need. I want to be able to extend a particular use in a smooth and incremental way. Those that disagree with this complaint dismiss it by stating: "Of course, I turn off all those features before I even use the word processor." Unwittingly, they underscore my position on this matter. Further, it should be noted that "turning off" these features is not such an obvious alternative for most users. Why not reverse the situation? Start out with a concise but capable core, which can be extended to provide features, built upon the core, and able to be invoked by those needing the specialized feature. It used to be possible to instruct novices easily in the use of word processors using a simple taxonomy and its associated structure. Now, it is necessary to remember where to find each feature distributed over menus in an unpredictable manner.

The Luddites refuse to use newer versions of the feature-bloated applications. A typical comment is: "I continue to use my older Mac and MacWrite, thank you." This is just fine for the isolated user who is totally content with the increased productivity of an older, less capable, but sufficiently productive application. That same Luddite, however, also reported recently that when seeking help with an older version of a slide making program he was told by the original supplier: "We no longer support that version but you can obtain help from Company X specializing in support for the unsupported version" - at $37/hour, of course.

Apart from isolation, the ultimate result of not keeping up with the latest versions eventually leads to inoperability when the computer fails and a newer model needs to be purchased. The cherished old friendly application will not run and the Luddite must catch up with the latest and greatest after all. The vendor is the apparent victor.

The Architectural Solution

What I seek is a conceptually clear and compact architecture for the application, and its application core, that is able to perform the fundamental operations. Such a core application can then be extended and provided with specialized user features and interfaces to match a particular market segment, see Figure 2. Developing such an application tempts one to classify it as a good engineering accomplishment – a successful software engineering accomplishment. Clearly, we have not yet reached this ideal.

The Process Solution

That the process of creating information technology products is faulty is not subject to argument. To put it more civilly, the process can be improved. But how? Norton and others argue that technologists dominate the industry. Engineers fiddle with a product without considering what a user wants. These critics propose, as an antidote, that marketing should and must control the products. They assert that to find out what a user wants we need marketing people. Only they can find out what users want and thus the marketing people must specify what engineers should produce. At a recent meeting of a software consortium this credo of distinction between engineers (read programmers) and marketing people was religiously upheld.

The implied lesson is that users can describe what they want and engineers can produce the consequent application to achieve the desired result. However, a good dose of technical analysis and knowledge can be and must be used to obtain a far more reasonable and efficient result. Technical efforts in the absence of user input (market studies) are as foolish as using market studies alone to specify technical requirements. I blame the feature bloat, mentioned above, on an excess of real or perceived market driven demands with insufficient technical analysis to define the core technical application capable of meeting those demands in a coherent technical manner. This is a faulty over-reaction which can be construed to be the source of the Feature Bloat mentioned above. The dichotomy of technologists and marketeers is too simplistic. Therefore, it is appropriate to make a proposal.

A Proposal

As a starting point, I borrow freely from the automobile industry. The products delivered by all automakers appear to satisfy the market demands. Further, the product is delivered on time: when new car models are to appear in the showroom they are there.

The software production and delivery process can be modeled with four essential elements:

Marketing

Product Planning

Engineering (or engineering product design)

Manufacturing (including especially quality assurance).

Here neither marketeers nor technologists are ignored. One of the roles of marketing is to measure what customers want. Ultimately, marketing must define market niches and sales strategies to help the sales force deliver the product.

There are two distinctive features I want to emphasize: product planning and product design. Product planning is a separate function in the quartet of marketing, product planning, design, and manufacturing. The role of product planning is to specify the product intended to meet the market demand but with full consideration of the advantages and constraints of design and manufacturing. In software applications the manufacturing process is relatively simple but does include quality assurance and testing – one wonders at times to what degree these two elements are neglected.

Engineering design, for automobiles, involves disciplines such as machine design, electronics and other disciplines as analysis and algorithms should in software design. But it does not end there. Drawings are made manually or now via CAD. These drawings are checked by one or more levels of checkers or supervisors who acknowledge compliance with product plans and engineering practice. It is not news that software engineering generally fails to omit these critical steps of checks and balances. My experience grounded in such practices leads me to be cynically suspicious of our software production methods.

As in the auto industry, the combined roles of engineering and manufacturing require special attention also in software production. One could argue that in the software industry, manufacturing is merely making CDs and manuals and that software development and production is an engineering function. No matter how that is resolved one can agree that technological innovations originate from engineering/manufacturing. Production of software is similarly a combined responsibility. Structured programming, style recommendations and object-oriented design are recognized to help but are all too often ignored in practice. Having designers be the testers and judges of quality. The dismissal of well-established practices is, in my opinion, a major cause of the malady. Practicing software engineers need to be persuaded about the merits of such practices. These practices are especially needed in an industry where change and innovation proceed with greater speed than in any other field of engineering.

A Matter of Attitude or Practice

There is another troublesome behavior or attitude held by information technologists and especially programmers. How often have you heard the statement: "We are researchers. We do not produce code of production quality. We just cobble up an implementation of our ideas to demonstrate their feasibility. We just barely make a demonstrable prototype."

A sense of double outrage engulfs me when I hear that statement – and I hear it all too often. I value my research efforts too much to endanger them by implementing them with sloppily developed code. How many times have research efforts prompted new and cleverer discoveries? Therefore, I must write well-structured and easily revisable code to meet the demands of new research results. Sloppily written programs most likely will produce erroneous results in the first place. A prototype may not be ready for prime time because it is incomplete for full production use. However, it should never be implemented with other than first class code. Unfortunately the believers in the above quotation not only eschew good programming practice but also almost deliberately hack code that inevitably must be thrown away - seemingly, by design! The concept of cobbled code is unprofessional, at least, and possibly evil. The practice of cobbled code can only be deplored and then stamped out.

Conclusion

An immediate conversion from the present bloated and ill behaving application to a conceptually concise and compact core application may ultimately be economically necessary but may not be a marketable idea. Still, to avoid collapse under the sheer weight of bloat demands a strategy of replacement. Before such a replacement can be undertaken we need to recognize the folly of our current practices and move to a more deliberate application building approach. In other words we have identified the enemy of

the future and the enemy is “us.” Ending on a note of optimism: when we throw out the old bloats and replace them with the trim and fit replacements, we know that our children will easily adopt our new products.

I hope this essay will foster some discussion to refine the proposal and to elicit suggestions and directions for the future. If we carry on as we have so far the future is doomed. Let’s convert us from enemies to liberators.