WORKING NOTE AC114

Alan Creak
2 September 1997

CRITICAL PARAMETERS

I explore the topic of critical parameters, advocated by William Newman in various writings, in the context of my own experience and reading.

GENESIS.

The seeds of these notes were sown in a seminar at York given by William Newman in 1996. That moved me to read his article in a compendium of essays[1], which in turn led me to get in touch with him. After an exchange of electronic mail, he sent me two further publications[2, 3] in which his more recent views are presented.

The essay in the book is noticeably different from the later papers in several ways. Most obviously, Newman's thinking has, unsurprisingly, developed, and his ideas are more precise in the papers; he distinguishes critical parameters from requirements, while the distinction is quite unclear in the essay. As well, his advocacy is directed in different ways; in the book, he is particularly concerned to introduce HCI into university computing courses, while in the papers he concentrates solely on persuading practitioners to adopt better HCI design techniques. I don't know whether that represents an evolution of ideas or the vagaries of very small samples, but it introduces two rather different topics.

I'll present a set of ruminations on the subject in an order which has little significance, and finally return to these two concerns.

NORMAL DESIGN.

Vincenti, arguing from extensive experience in the aircraft industry[4], discusses[5] "normal design", which is the activity of designing new versions of known objects. Generally, the intention is to proceed by development rather than by innovation - to produce a better version of something which already exists and has well known properties, not to produce something new and unknown. Most conventional engineering design is of this sort. We certainly do some of this in computing, and have done for a long time; a systematic approach to drawing up a specification document for real-time computing systems developed[6] in the 1970s (coincidentally, also originating in the aircraft industry) is clearly based on strong expectations about programme structure of just this "normal design" type, and the same pattern is still perfectly usable today. The same example illustrates the evolutionary principle in action at different levels, for, while the structure of the programmes designed has developed into a class of techniques such as rate-monotonic analysis[7], the design method itself can be seen as a forerunner of today's object-oriented design techniques. That example is unlikely to be unique - I chose it simply because it's familiar to me. On the other hand, Newman[8] has pointed out that such development is the exception in HCI work. If HCI development is not usually to be classified as normal design, what it ?

Vincenti's alternative to normal design[9] is "radical design". In this mode, the designer is working with unknowns, and the task is not so much to produce a design which will work better as to produce one which will work at all - and, says Vincenti, which will warrant further development. Can HCI practice be described as radical design ? Looking at the characteristics of normal design, one can argue either way. Is HCI at the stage where[9] "how the device should be arranged or even how it works is largely unknown" ? This is the position suggested by Newman's observations. If so, it is perhaps because the enormous variety of possible user interfaces makes it fairly certain that each one we build will be novel. On the other hand, we can see HCI as having the characteristics of normal design, with everything a variant of much the same basic model with accepted requirements. Looking at the interfaces at a more detailed level, it is easy to suppose that there is a "normal configuration"[10] of windows, buttons, menus, clicking, selecting, dragging, etc. Further, this configuration is generally accepted, just as Gilruth is reported[11] as commenting that his critical parameter for aeroplane design was accepted as applying to any sort of aeroplane.

Vincenti describes the normal configuration as "the general shape and arrangement that are commonly agreed to best embody the operational principle", and gives the example[12] of aircraft designers in the late 1930s who would automatically assume that an aeroplane would be "a tractor monoplane with tail aft". Today's interface designers do much the same, following interface style guides appropriate to their chosen look-and-feel flavour, in the belief that maintaining consistency is a valuable component of good design. Perhaps that is why we have a normal configuration, but without the development expected

of a normal design position. If so, we are dealing with a field of endeavour where normal design rules at the widgets-and-gadgets level, with high-level interface design not so much radical as absent, replaced by conformity to look-and-feel conventions. This leads to a question : to what extent are the innovative designs observed by Newman really embodiments of significant new principles, and to what extent are they better seen as (not necessarily unworthy) implementations of functions which might be novel in themselves but are constructed in fairly obvious ways from well known HCI components ? My guess, without a careful survey but with much experience of reading conference reports, is that the second category predominates by a considerable majority.

Reflection suggests that matters are not quite so simple as that. Even the most innovative design must grow from the same roots as existing systems; the most original new aeroplane is going to be built from the same old materials, fabrication techniques, nuts and bolts as something else, if only because we cannot invent everything from scratch at once. We have to know the properties of the materials, fabrication techniques, and nuts and bolts before we dare use them in our new aircraft, so they will have already have been used in some other context. Radical engineering will therefore necessarily happen on top of normal engineering.

Is it no more than a matter of scale or level ? Newman identifies the traffic flow rates at a road junction as a critical parameter[3], but optimising these in isolation can be useless unless properties of the whole interacting network of junctions are taken into account. Similarly, could the phenomena of Project Ernestine be regarded as a failure of individual small-scale critical parameters in the context of a larger system ?

WHAT ARE CRITICAL PARAMETERS ?

Newman states[3] that critical parameters are "defined at the outset of design as performance targets", but "are not always obvious, or easily identified" - despite which, he follows that statement with what looks suspiciously like an instant and easy identification of obvious critical parameters in four separate cases. He draws a distinction between *requirements*, which are the desired ends of the design process, and *critical parameters*, which are figures (or perhaps, more generally, gauges) of merit which should be optimised to approach the requirements. Parker, Roast, and Siddiqi[13] agree with this view of the relationship between requirements and "measures of non-functional properties" - which, in their context of usability, it is reasonable to identify with critical parameters. They remark that "specification of usability requirements remains an under-developed field", and that "traditional methods offer little guidance on the formulation or calculation of appropriate measures of non-functional properties and on the application and verification/satisfiability of these metrics in the design stage".

Vincenti does not make this distinction very clearly; in his book, the requirements *are* (or, at least, include) the critical parameters[11] : "By 1941 the accumulated information and experience enabled Gilruth to publish a refined set of requirements. To arrive at his results, Gilruth reviewed the mass of flight data and pilot opinion to see what measured characteristics proved significant ...". Vincenti also uses the term[14] *engineering criteria* in a sense which is certainly close to that of critical parameters, but he doesn't seem to make a clear distinction between the levels. He states[14] that, without knowing the appropriate engineering criteria, designers have to "guess what their customers want"; so the criteria are in some sense measures of what the customers want. Newman starts from this position; in his earlier paper he lists[15] a set of "requirements" - but these attributes would later be regarded by Newman as critical parameters rather than requirements.

Vincenti's discussion suggests that he agrees with Newman in supposing that the criteria are sometimes fairly obvious, but at other times not. He regards his engineering criteria, along with other topics which can be regarded as engineering design knowledge, as growing from engineering practice[16]; you need a lot of experience with the problem before such things become clear. It is later suggested[17] that criteria and specifications grow from theoretical and experimental research, design practice, and running experience.

Critical parameters are[3] "defined ... as performance targets"; engineering criteria are in some sense measures of what the customers want; requirements are more or less formal statements of how a system shall behave, typically set down before the system is constructed. These three terms are clearly intimately related. I have already suggested that the terms *critical parameters* and *engineering criteria* are very similar in meaning, and I further suggest that we lose nothing if we identify them; I use the term critical

parameters for both henceforth. Critical parameters are then seen as directly or indirectly measurable properties of a system which give some guide to how well the system approaches its requirements in some respect.

Newman gives [15] a list of "requirements" for "mainstream computing" - speed, reliability, ease of reuse, ease of maintenance - then bewails "its inability to find a place for interactive computing". Perhaps this is because they are critical parameters *derived from* the requirements rather than requirements themselves. If the requirements don't include or imply HCI, which is reasonable enough for the topics which Newman regards as mainstream computing (discrete structures, algorithms, data structures, computer hardware, operating systems, distributed systems, programming languages[42]), then no amount of subsequent analysis will introduce it. Further, if explicit HCI outcomes are required, as is reasonable with user interface topics, why aren't they included with the requirements when systems of this type are designed ? It's not unreasonable to add special requirements for special fields; you have to add the requirement that the system be functional for operating systems and distributed systems.

The requirements are taken for granted in normal design, but that doesn't mean that they're correct or complete. Before jet engines were invented, and for an uncomfortably long time afterwards, everyone took it for granted that aeroplanes had propellers; in the same way, doubtless there were accepted requirements for tools, but they were formulated without regard for the detail that people had to use them. The situation changed when the importance of ergonomic considerations was understood. That's just like ignoring HCI; the solution is to add a new normal requirement, which is ergonomics.

Vincenti[18] discusses how, through defining a critical parameter, "an ill-defined problem, containing in this case a large subjective human element" was refined into "an objective, well-defined problem for the designer". It is interesting that the problem studied by Vincenti is also to do with a human interface[40] - and that it wasn't until people began to study the essentially subjective "flying qualities" of aircraft that progress was made. Again, before progress could be made, there was a need to accept that the interaction with people was important. That's where we are with HCIs.

Braha and Maimon[19] draw a laboured analogy between system design and the process of scientific research. Through this paper, though, it becomes clear - as much by considering the examples as from the text - that an essential prerequisite for the design process is to identify clearly what you're trying to do. That's what the aircraft designers did when they took notice of flying qualities.

Judy Brown[20] compares HCI and software engineering approaches to designing interactive software, and concludes that "HCI specialists are user-centered and software engineers are system-centered", where in the context "system" means "computer system". I would argue that to adopt either of these stances is bad engineering, as both ignore a part of the problem. I would wish to begin by working out what the significant system really is by defining the object of the exercise more carefully. The moral of the story is that you can't do that by normal design, which is, in effect, an assumption that you know the object of the exercise already. Radical design is essential, and here I'm suggesting that it is likely to happen if one seriously confronts the problem to be solved without assumptions about system-centred or user-centred models. I'd prefer to regard this as *problem-centred* design.

Where performance is perceived to be important, critical parameters are recognised and used. This is certainly the case in rehabilitation systems, where maximum communication per keystroke is a well-recognised critical parameter (though not everyone agrees that it's the most useful). Stevens and Edwards[21] discuss the question of evaluation in some detail, pointing out some difficulties in evaluation, and showing how several specific factors affecting performance can be measured.

Is the idea of critical parameters itself a critical parameter ? If the definition of some physical quantity which must be optimised is important in developing a piece of machinery, is the definition of an optimal notion of critical parameters important in developing the technique of design ? There are similar features : the notion surfaced from much experience in design, and could be seen as a characteristic of a normal design process.

THE DEVELOPMENT PROCESS.

Developments in knowledge come from blind variation and selective retention[22]; new combinations of parameters are selected for testing without full information (that's the blindness), but taking into account

previous experience. Wirth[34], comparing software development unfavourably with other engineering fields, writes "Good engineering is characterized by a gradual, stepwise refinement of products that yields increased performance", very much in harmony with normal engineering. In a rather long paper (apparently full of promise, but in the end much less exciting than I'd expected - perhaps through my recent immersion in Vincenti, Newman, et al.), Braha and Maimon[19] call it *bounded rationality*. I'd probably call it informed guesswork. Then from results obtained, a favoured combination is chosen for further investigation. A sort of algorithm for identifying possible variations is presented[41] : first, look for similar situations to identify useful knowledge; then try to work out what differences will be made by deviations of the immediate problem from the experience; then pick a likely solution from the results obtained from the previous steps. A broadly similar method is presented by Braha and Maimon, who develop it by pursuing an analogy between what they call *incremental redesign* (Vincenti's normal design) and scientific research. (They also recognise radical design, which they call *innovative redesign*.)

How far do these methods apply to software ? While my memories are certainly unlikely to be representative of software design as a whole, it impresses me that I don't recall anything in software development which looked like optimisation in an engineering sense, with some notion equivalent to a maximum in a merit function. "Engineering is an intensely quantitative activity"[23] - while computing, by and large, isn't. Engineering is helped by generally continuously varying properties, meaning that graphs of A against B are well behaved, and interpolation makes sense. If you don't have that sort of continuous variation, then lots of measurements aren't really a lot of use.

Operating systems are like that in real life, however neat and tidy they seem in textbooks. Years ago, I did some measurements[24] of various parameters of a Tops-10 system under normal load conditions. The results were scattered in the extreme; system behaviour would change moment by moment for no obvious reason from calm and steady to violent thrashing - or we would measure exactly the same parameters for a calm and steady system as we had previously for a violently thrashing system. There were trends in average values, but they were of no help in predicting the actual behaviour. As another example, exhaustive surveys of some aspects of the behaviour of simple cellular automata have been carried out[25]. In this case, systematic variation of a simple bit string over its range resulted in patterns of behaviour of the automata which could be classified into a few characteristic types, and predominant types could be associated with different ranges of the bit string's value - but the association was statistical, and all sorts of behaviours could be found in most of the ranges.

Even when such disorderly behaviour isn't evident, computists don't seem to understand the idea of systematically studying variation of parameters. One of my students[26] was an example. He performed some experiments with genetic algorithms, where there was a rather well defined critical parameter, but in trying to identify how other parameters affected this he carried out experiments at a small number of points widely scattered in the enormous parameter space, and attempted to draw conclusions from the results. Others seem not able to design appropriate experiments; for example, in experiments on a method for reinforcement learning[27] the experimenter used such a complicated system that the significance of an extensive series of measurements is very difficult (well, impossible) to discern. With a simpler (though much less exciting) system, it might have been possible to gain far more insight into the mechanisms of the learning.

Vincenti's ideas seem to be coloured by an expectation of finding optimal points in smooth(ish) functions of continuous variables; judging by the list of examples which he gives[3], Newman's seem rather to be concerned with identifying useful parameters, and using them to compare different methods' efficiencies. And that's rather like what I've been doing, particularly in respect of rehabilitation systems; I look for the best system, not the best parameterisation of one system. Does anyone do the laborious and exhaustive testing for software systems ? (Compare the aeroplane wing research in chapter 2 of Vincenti's book[4].) I've seen occasional attempts to measure things, but I can recall nothing even approaching the systematic approach illustrated there.

Are there really two distinct phenomena here ? The engineer, perhaps like the HCI designer who is similarly dealing with events in the real physical world, is concerned with continuous phenomena; optimising critical parameters makes sense, because we can understand the behaviour of smooth curves. The computist, in contrast, lives in a world of discontinuities, where phenomena are capricious, with behaviour which depends on every bit in the system. We don't know even how to search, and in despair

take refuge in simulated annealing, genetic algorithms, and other such methods. Or we abandon the idea of search and require certainty, which leads us back to formal methods and programme proofs.

In view of the possibility of two different sorts of behaviour, it is interesting to speculate on whether we could do with some radical design in this business of designing design which is my topic in this note. Is it possible to suggest that Newman, in following Vincenti's lead rather closely, is exercising normal design in a situation where radical design might be more appropriate ? Taking a problem-centred approach, we should ask precisely what our requirements are in pursuing the design of development techniques, and then apply the answer to the different sorts of system.

DISTURBING FACTORS.

What factors work against the practice of normal design ? Apart from complexity measures, software is commonly thought of as binary - either it works, or it doesn't. Complexity considerations encourage this view, by implying that the proportionality constant doesn't matter. In HCI systems, there isn't even complexity to hang on to. The complexity isn't so serious an issue in a lot of HCI programming, because much of the computing comes in short bursts on just a few items, with long gaps in between for the operator to think.

Newman[2] lists five categories of *engineering knowledge* (artefacts, artefacts' environments, representations, analysis techniques, critical parameters) which he would expect a software designer to need. He remarks that designers previously studied in fact fell short on the last two of these categories. They did not use any quantitative measures of performance, but preferred less quantifiable attributes such as reliability or simplicity. Newman[2] is surprised at this deficiency, because "programmers are known to take pride in squeezing speed improvements out of programs". The problem is particularly severe in the case of HCI systems; Newman[2] makes out a case that this can be accounted for by comparatively poor sources for the five categories of knowledge he identifies. I'm not sure that his remarks about squeezing speed improvements are relevant any more; they certainly used to be, but "efficiency" is now a fairly dirty word, apart from complexity analysis[2], where - as I mentioned - the proportionality constants are rarely taken into consideration. Programmers no longer need to write tight code, as they can rely on increasing speed of processors to make it run. In programming courses, structured design is stressed as far more important than speed.

This observation fits in with a suggestion[43] that programming is becoming, or has become, detached from the world of the problems which the programmes are intended to solve. Programmers (it is asserted) seek abstraction, while the world, for the most part, remains resolutely and incorrigibly concrete; elegant abstractions are valued more than boring performance. This view caused me to look twice at a paper[44] which I had originally seen as a vote for critical parameters in its (rather colourful) advocacy of the use of software metrics in software design and development. On inspection, it turns out that none of the measures suggested has more than a passing connection with performance - which is all well and good, provided that the performance is considered elsewhere, but I'm not convinced that it is.

I don't think that students nowadays are taught to aim for high performance when programming, beyond avoiding methods which obviously have a high complexity exponent. Common sense isn't common any more; Jon Bentley used to have a regular column of "programming pearls" in CACM[28] which was an excellent source of sensible observations, but that was before CACM turned itself into a sort of comic. A recent CACM paper[29] shows a very different attitude; it seems to be written on the assumption that a programmer composes a programme while writing the code at the keyboard, with no preliminary design stage at all. This is certainly not in the tradition of carefully tailoring the instructions for maximum speed[30].

As an aside, one might also wonder whether development of existing interface methods is likely to be inhibited by the ridiculous "copyright" lawsuits about interfaces which have occurred in past years. If it should happen that your interface could be made more efficient by incorporating some feature of your rival's (which seems pretty likely), who is going to try it ? Such a constraint could force people into novelty when they should be producing better versions of the "normal" interface designs.

I came across an example of this rampant novelty recently in an article[31] which impressed me as one of the worst examples of arbitrary functionality-seeking which I'd read for a long time. That being so, I was astonished to find towards the end of the article a strong endorsement[32] for the identification and use

of critical parameters - but perhaps it doesn't count, because it wasn't HCI. On the other hand, it was middleware, just the sort of trendy area (as evidenced by the rest of the article) where people are eager to do something - anything - quickly.

The appeal to critical parameters nevertheless suggests that people are prepared to take them seriously, but if they can get away without using them, they will. Fields in which they're used (engineering, established software areas) have evolved to the point where normal design is expected, and more or less required if you want to make progress.

Newman's observations suggest that normal design is not much in evidence in HCI, though he asserts that in mainstream computing the normal design pattern is approached more closely. Is this true ? Methods evolve, as I observed in the context of real-time systems - but that is a rather straightforward case in that the major critical parameter (to get the tasks finished before their deadlines) is obvious and hasn't changed. Another example, chosen not entirely at random, is operating systems. In this case, there has been at least one very significant change of emphasis, from efficient use of hardware to efficient use of people. So far as this discussion is concerned, that's probably better regarded as a change in the requirements than as a matter of critical parameters. Indeed, it was precisely my recognising this very significant change in requirements which led me to reorganise my operating systems course[33]; it was the realisation that the task of an operating system is not merely to run the machinery but to provide computing services to people that caused me to pay attention to HCI topics, and eventually to make them in some ways the central feature of my treatment of the subject. I find it curious that, while HCI topics certainly strongly influence the way people write operating systems (an old-fashioned operating system couldn't drive a modern user interface), it does seem to be largely ignored in considering the design of operating systems. Even modern textbooks are likely to treat the user interface as an incidental topic. (If mine is ever published - CUP have been sitting on it and making apologetic noises for over two years now - that will change.)

The change of requirements for operating systems was more economic than enlightened. In the process, perhaps we learnt something - but we didn't necessarily learn it very well. I remember seeing a graph (though unfortunately I don't remember where) which showed the stages of development of operating systems plotted against time for systems written for mainframes, minicomputers, and microcomputers. The three curves were almost identical in shape, though displaced along the time axis; the designers had learnt nothing from the previous experience. It is certainly not the case that computists as a whole have learnt nothing; we now know enough to construct very reliable and effective operating systems if we so desire. One can only conclude that manufacturers don't desire. Wirth has some typically trenchant things to say about it[34].

Even in these comparatively staid and settled fields, then, good engineering practice is far from universal. Why ? I think that the quick answer, and not necessarily the wrong answer, is that manufacturers can get away with it, and still make satisfying profits. Software almost always gets faster and more versatile, but much of this improvement can be attributed to the steady and reliable increase in processor speeds and memory capacities. The software manufacturers have not had to try particularly hard.

HCI has one special characteristic which might predispose it to be done not so well; you have to do laborious and fuzzy experiments to measure the critical parameters with HCI systems (just as engineers do). It's comparatively easy and clean to measure performance on systems which don't involve people, and you can often automate them so that the computer can do them on its own overnight, but for HCI you must have people, who are all different, so you need even more people and it's hard.

CRITICAL PARAMETERS AND ME.

Do I believe in critical parameters ? Yes, though I haven't called them by that name, nor always looked for an explicitly numeric measure. There is some evidence. For a long time, I've urged students to evaluate their software in some way. For conventional software, I've recommended comparing it with another implementation, and the comparisons have commonly been qualitative rather than quantitative. Nevertheless, the attributes on which the comparison is based are critical variables of a sort. The practice of this fine principle has not matched the intention; you can lead a student to produce software, but you can't make him evaluate it. I've urged the use of more recognisable critical parameters in cases like

simulators, where something like a machine is clearly modelled. Examples include neural networks[35], genetic algorithms, reinforcement learning, and so on.

Once the notion of critical parameters has been established, it can be developed in many different ways. One of my activities at York was to work on the application of a formal notation to the analysis of rehabilitation systems[36], with the eventual modelling of systems as one of the aims. This would open the way to simulation of the behaviour of interfaces in some detail, and possibly (depending on how far it was possible to associate specific costs with elementary interface operations) to their evaluation without detailed testing - provided that times and loads for the atomic operations can be established.

In a close approach to one sort of HCI, the use of critical parameters turned out to be thwarted by lack of evidence. Simon Dixon and I, following on from Simon's thesis[37] work on comparing authoring packages, tried to work towards a relationship between the facilities offered by an authoring package and its usefulness. The principle was simple enough : identify the useful features offered by the package (text, sound, video, etc.), find out how these could be used to produce certain desirable educational ends, and assess the differences between the ease with which educational ends could be achieved by different packages. This came to nought because no one knows the important things about the effect of the HCI.

In one way, I can perhaps claim to be a little ahead of the field. I have argued[38] the importance of aiming beyond the current practice in designing interfaces for disabled people, and urged that the design aim should be full human communication rather than mere character transport. I think that counts as radical design. I have not gone on to define critical parameters; that's a lot harder, and indeed requires more experience in the practical application of the idea than anyone has at present.

In summary, I believe in critical parameters, but in practice I've thought of optimisable numeric values more in the context of software with physical-like behaviour. In most other cases, I've thought only of comparisons between values of various parameters; this has the germs of critical parameters, but no real substance.

AN ALTERNATIVE VIEW.

I'm not entirely convinced by Newman's identification of HCI as the exceptional field in which good engineering practice isn't so evident. I suspect that it might be more a matter of easy and exciting fields, where many (often comparatively young and inexperienced) practitioners can have a lot of fun uninhibited by recognised practice, and in so doing amass impressive publication records. I have no hard evidence to support this assertion, but I detect a certain similarity in the attitudes exhibited by adherents of these fields in public fora like conferences and newsgroups. I've certainly found the same phenomena in topics as diverse as neural networks, genetic algorithms, reinforcement learning, and fuzzy systems as well as HCI. In all these areas, I've been exasperated by flashy publications in which more or less spectacular work is presented in a manner which makes a point by exhibiting an example, but defies any deeper analysis because the system studied is far too complex. It is true that these examples differ from the HCI work which Newman investigated in that there is often some defined critical parameter (accuracy of identification, efficiency in performance, speed of learning, etc.), but the broader fault of poor experimental design is common to all.

It is significant that, in the context of computer programmes, the far-too-complex (but entertaining) experiment can be quite easily constructed and run without any need to embark on the detailed study which might make it far easier to construct in such a way that it would run much more effectively - particularly if you need only run it once to get your publication. That's because computer programming is too easy and too cheap; in many cases, you can make a programme that works passably well without taking any care at all. You do not find aircraft engineers building bizarre aeroplanes just for the fun of it. In areas where simple performance of a function is more important than efficient execution, performance might not be regarded as a criterion to be weighted very highly. ("Just wait for the next generation of processors.") Excepting the one-off studies, there are (at least) two sorts of area which fit this pattern : those where increases in software performance don't make much difference to the overall speed - typically areas in which some other system determines the maximum speed, such as communications or interfaces - and those where the benefits of getting a new and different idea running gain you more credit than you would earn by making it efficient - staking claims to commercial areas, or getting in first in active research areas. The examples I mentioned all fall into these classes. The middleware example I mentioned earlier[31] is perhaps another example, though in that case the motive was

perhaps commercial rather than adventurous. Wirth[34] seems to believe that the problem is endemic in modern computing : "Meticulous engineering habits do not pay off in the short run, which is one reason why software plays a dubious role among established engineering disciplines".

Ted Lewis[39] continues the same theme, urging the view that ideas have to be developed quickly for commercial success, and perhaps implying that there won't be time to do careful development until an idea is established : "If a technology (or an idea) does not achieve mainstream status within a decade, it dies". It's therefore only the established ideas which get developed, and, once established, careful development doesn't get many publications.

BACK TO THE ORIGINAL CONCERNS.

I began by noting Newman's two concerns - that HCI matters should be introduced into university courses, and that HCI practitioners should pay more attention to identifying sound development techniques. While thinking about these matters, I have come to the conclusion that, though obviously related, these are quite distinct in nature, and have different (if any) solutions.

I think that the key to the university problem lies in getting the requirements right. Newman considers the possibility[8] that "HCI could simply be declared to be a mainstream computing topic", but it seems unlikely that such an apparently arbitrary action would be fruitful. He continues by asking : "Can mainstream computing and HCI be integrated successfully into a single discipline ?". I think that's a mistaken approach. It is more important to define the discipline correctly from the start. Just as it was found to be important to design aeroplanes so that people could fly them effectively, so it should be recognised that it is important to design computer systems so that people can use them effectively - which automatically and inevitably introduces people into the system from the start, and leads naturally to a consideration of interface factors in identifying system requirements as a normal part of the design process.

To some extent I'm biased by my own experience with my operating systems course, where I followed precisely this path, with (what I believe to be) very rewarding consequences. In practice, university syllabuses are responsive to changing perceptions of what is important, and a conviction that people are part of the system to be studied will work through eventually. Having said that, it is only fair to add that my experience in trying to persuade my department that we should offer courses specifically in HCI has not been encouraging - and I was surprised to find that even at York, with a highly regarded HCI research group, there is no undergraduate HCI course.

For the industrial problem, I can see no such clear-cut potential solution. So long as software purveyors can get away with dazzling naive purchasers by providing more and more useless "functionality", to do so will be easier and cheaper than investing resources in careful and painstaking development. My own hope is that before long customers will begin to question the astonishing "guarantees" common with software, which in effect absolve the manufacturer from any responsibility whatever. Once people insist that manufacturers take responsibility for their products' failures in the same way as honest engineers always have done, I think that there will be many interesting changes. C and C++ will disappear, and perhaps Java will too; formal methods and systematic design techniques will become popular overnight; and real benefits to people, rather than eye-catching gimmicks, will receive attention. And pigs will fly.

REFERENCES.

1 : W. Newman : "The place of interactive computing in tomorrow's computer science", in *Computing Tomorrow*, ed. I. Wand, R. Milner (Cambridge, 1996).

2 : W. Newman : "What application designers know : some thoughts on innovation in interactive systems", *Proceedings of Software Ergonomie '97* (Dresden, March, 1997).

3 : W. Newman : "Better or just different ? On the benefits of designing interactive systems in terms of critical parameters", Preprint, submitted to *DIS '97 : designing interactive systems*, August, 1997.

4: W.G. Vincenti : *What engineers know and how they know it* (Johns Hopkins University Press, 1990).

5 : Reference 4, pages 7 - 8.

6 : K.H. Britton : "Specifying software requirements for complex systems" (*Proceedings of IEEE conference on specifications of reliable software*, 1979), as reprinted in R.L. Glass : *Real-time Software* (Prentice-Hall, 1983).

7 : M.H. Klein, J.P. Lehoczky, R. Rajkumar : "Rate-monotonic analysis for real-time industrial computing", *IEEE Computer* **27#1**, 24 (January, 1994).

8 : Reference 1, page 305.

9 : Reference 4, page 8.

10 : Reference 4, page 209.

11 : Reference 4, page 92.

12 : Reference 4, pages 209 - 210.

13 : H. Parker, C. Roast, J. Siddiqi : "Towards a framework for investigating temporal properties in interaction", *SigCHI Bulletin* **29#1**, 56-60 (January, 1997).

14 : Reference 4, page 211.

15 : Reference 1, page 303.

16 : Reference 4, page 225.

17 : Reference 4, page 235.

18 : Reference 4, page 51.

19 : D. Braha, O. Maimon : "The design process : properties, paradigms, and structure", *IEEE Trans. Sys. Man Cyb. A* **27**, 146-166 (1997).

20 : J. Brown : "Exploring human-computer interaction and software engineering methodologies for the creation of interactive software", *Sigchi Bulletin* **29#1**, 32-35 (January, 1997).

21 : R.D. Stevens, A.D.N. Edwards : "An approach to the evaluation of assistive technology", *ASSETS '96 The second annual ACM conference on assistive technologies* (Vancouver, April, 1996), pages 64-71.

22 : Reference 4, page 48.

23 : Reference 4, page 205.

24 : (For example - records are sparse) G.A. Creak : *Monitoring the DEC-10's performance* (Unpublished Working Note AC40, 29 August, 1984).

25 : A. Wuensche: *Complexity in One-D Cellular Automata: gliders, basins of attraction and the Z parameter* (Cognitive Science Research Paper 321, School of Cognitive and Computing Sciences, University of Sussex, 1994).

26 : D. Norman : *Emergence in artificial intelligence* (M.Sc. Thesis, Computer Science Department, Auckland University, 1995).

27 : L.-J. Lin : "Self-improving reactive agents : case studies of reinforcement learning frameworks", *From Animals To Animats*, Proceedings of the First International Conference on Simulation of Adaptive Behavior (MIT Press, 1990).

28 : J. Bentley : *Programming Pearls* (Addison-Wesley, 1986).

29 : C. Fry : “Programming on an already full brain”, *Comm. ACM* **40#4**. 55-64 (1997).

30 : E. Nather : *The Story of Mel, a Real Programmer*, http://www.datamation.com/PlugIn/humor/jargon/jargon_48.html (May 21, 1983).

31 : P.A. Bernstein : “Middleware : a model for distributed system services”, *Comm. ACM* **39#2**, 86-98 (February, 1996).

32 : Reference 31, page 97, left column.

33 : G.A. Creak : *Operating systems course structure*, unpublished Working Note AC75 (December, 1989).

34 : N. Wirth : “A plea for lean software”, *IEEE Computer* **28#2**, 64-68 (February, 1995).

35 : G.A. Creak : *Some notes on a proposed interface device*, unpublished Working Note AC78 (December, 1990).

36 : G.A. Creak : *Interactors in rehabilitation system design*, unpublished Working Note AC97 (August, 1996)..

37 : S.H. Dixon : *A microcomputer programme for teaching money-handling skills in an experience class* (M.A. Thesis, Department of Education, Auckland University, 1992).

38 : G.A. Creak : "Notes for a seminar : Insights from a System Specification Aid", *Sigcaph Newsletter* **#58** (June, 1997).

39 : T. Lewis : "The big software chill", *IEEE Computer* **29#3**, 12-14 (March, 1996).

40 : Reference 4, page 53.

41 : Reference 4, page 246.

42 : Reference 1, page 302.

43 : K.G. Balke : a letter, *Comm.ACM* **38#5**, 11-12 (May, 1995).

44 : L.O. Ejiogu : "On diminishing the vibrant confusion in software metrics", *Sigplan Notices* **32#2**, 35-38 (February, 1997).