Human-Computer Interaction – INTERACT '84 / B. Shackel (ed.)
Elsevier Science Publishers B.V. (North-Holland)

GENERATIVE USER-ENGINEERING PRINCIPLES FOR USER INTERFACE DESIGN

Harold Thimbleby
Department of Computer Science
University of York
YORK, YO1 5DD, UK

Generative user-engineering principles are assertions about interactive system behaviour and have equivalent colloquial forms. Current work shows that they are a promising contribution to the design of acceptable user interfaces, because they effectively bridge the conceptual gap between designer and user. In colloquial form a generative user-engineering principle can be used to help clarify requirements in participative design, or to explicate documentation. In rigorous form, generative user-engineering principles provide a constructive higher order consistency on user interfaces.

1 INTRODUCTION

Currently, the best designed interactive systems are partly specified formally and partly developed through ad hoc accretion. Many such accretions are details for handling abnormal boundary conditions, which are usually afterthoughts to the formal specification. Despite 'user-engineering principles', no overall consistency emerges, and certainly no consistency which can be applied down to details. This is a disaster for the purposes of introducing an interactive system to new users, or for interactive systems which are intended for non-computer-expert use. This paper suggests that 'generative' principles should be employed as theorems over the system specification: this overcomes the bottom-up piece-meal approach of applying conventional user-engineering principles.

If, in addition, we require the user interface to be compatible with external requirements (and not merely internally consistent), the generative principles must be expressible in a form accessible to users. In a colloquial form a generative principle could be used in both requirements specification and in documentation. In one case, the principle would permit the user to contribute to higher-order requirements, and in the other case, even if the user has not participated in the system design, he could be given an unusually coherent view of it. The colloquial form must be judiciously chosen; in [1] the colloquial form is termed a 'metaphor' and excellent recommendations are given for their presentation.

We shall use the term 'generative user engineering principle' (abbreviated 'guep'), to distinguish a principle which has a colloquial form and can express constraints on the overt behaviour of a system. Necessarily a guep has to be compatible with human factors guidelines and this can be achieved weakly post hoc, because simply stating a system property makes a system easier to use. The longer term releases the basic term 'generative principle' to be used for specificational purposes where there may need be no human factors requirements at all.

Ideally, user interfaces may be designed by collaboration between user, ergonomist and computer scientist (using the three terms generally). However, each view has limitations which restrict communication and curtail the emergence of an overall design approach. This paper proposes using gueps as a 'thematic' requirement of the user interface behaviour expressible in colloquial task-oriented terms. With them, the user may have reasonable and well founded expectations; the ergonomist can check compatability of a system against them; and the computer scientist may rephrase them as theorems over the formal specification. In addition, gueps, as generative principles, impose a high degree of internal consistency.

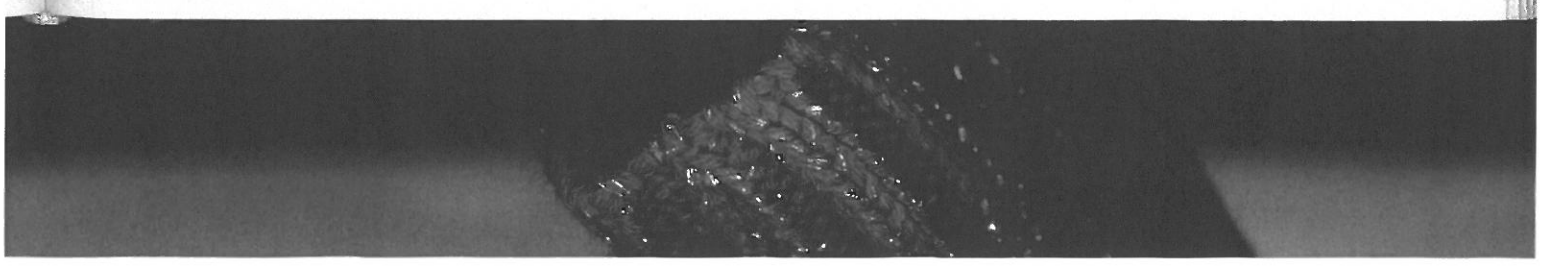
The idea is not new, and appears to have been first used under the JOSS system, where gueps were termed 'rubrics'[2]. We could view the user model as an abstraction of required system behaviour, and under suitable definitions it could then be viewed as a guep.

1.1 Hypotheses

I am making a strong hypothesis that gueps exist and that they are useful for both designer and user. A weaker hypothesis is that only certain pre-evaluated forms of guep are properly effective, and therefore the approach is not applicable for novel user interfaces, where a set of suitable principles has not already been established. A still weaker hypothesis is that gueps have negligible effect on the user at the interface but are nonetheless useful in specification (as generative principles). Directions for research are discussed in §7.

2 GUEP VERSUS USER-ENGINEERING PRINCIPLE

User-engineering principles which are sufficiently constructive to be used in design are numerous and are highly task-specific[3]. In fact, many such principles appearing in the literature have been derived from very small experiments and it is dubious whether such hypotheses merit promotion to principles. It is unlikely that constructive principles can be generalised or used in combination reliably, certainly not without considerable design experience.



For example, it is not clear when a mouse outperforms key selection for selecting items from a known menu, though experiments suggest that unanticipated cursor positioning is faster using a mouse. Principles interact too strongly with user skill levels, task dimensions (frequency, openness), hardware (e.g. resolution, response times), psychosocial issues (e.g. consequences of user performance and error), and may even be in mutual conflict. Really, user-engineering principles, at this level, can be used no more constructively than as suggestions for particular design features.

A user-engineering principle is an informal requirements statement, which is yet too sophisticated for the user and can still be interpreted by the designer with considerable freedom. A typical user-engineering principle from [4] illustrates this: "User orientation must be maintained throughout a session. Information detailing the user's status should be displayed." However some principles operate at a higher level (e.g. be consistent, reduce modes) and could be formalised as requirements over the entire specification. These two particular principles happen to be relatively trivial in relation to a formal specification, but nonetheless are too sophisticated for computer-naive users.

Distinctively generative user-engineering principles meet four criteria: (i) they can be expressed formally (ii) they have a colloquial form (iii) in common with user-engineering principles (which can be instantiations of gueps), they embody certain ergonomic guidelines and (iv) they are constructive rather than descriptive.

3 EXAMPLES

3.1 Gueps for Concealed Information

Information in user interfaces may be concealed by a variety of means, usually with the intention of making the dialogue more rapid (e.g. because less needs to be said), catering for varying levels of user skill or for folding complex information into uncluttered forms.

Typical mechanisms for concealing information range from the straightforward absence of the information (as in default command operands), forms of abstraction (as in macros), to symbolic expressions (as in regular expressions). The purpose of the relevant guep is to express constraints on these kinds of mechanisms, and hence improve the chances of designing a consistent user interface (and, additionally, give the user an explicit key to the particular form of consistency). The distinction between gueps and user-engineering principles is now perhaps a little clearer: a plausible user-engineering principle might suggest that "defaultable operands should occupy a consistent position" but this is very specific; it has a limited domain (in this case certainly to linear command strings) and it is not at all clear that one can be used effectively in formal design, an area where human factors expertise is generally absent.

In comparison a plausible guep would be that (**a**) "only complete instances of syntactic categories may be abstracted (i.e. replaced by names, marks, abbreviations or stubs) or concealed (i.e. eliminated from the operand set altogether)". This might be supplemented by the following: (**b**) "information may be totally concealed locally, but there must be a convention to indicate this is occurring"; (**c**) "information concealing/revealing never occurs as a side-effect" (e.g. a search would not unfold data which contains the searched-for items so the user is left at the same level of abstraction); (**d**) "the body of an abstraction is consistently available as an operand"; (**e**) "the abstraction mechanism is conservative and invertible". In fact, these gueps might be part of a stronger set which implies the commutativity of the operations required to implement them.

For clarity we shall introduce the terminology: a **fold** is an instance of an abstraction or concealment and **folding** is the act of abstracting away or concealing information. Similarly, **unfolding** is the inverse act, of exposing the concealed information. If abstraction requires naming, unfolding is not normally considered to lose the name, nor the extent of the abstract's body — in other words unfolding is normally invertible.

The gueps above may be formalised in terms of the chosen formal specification method, thus their correct application may be established mechanically. As required by the user, a more colloquial expression of these gueps is that only entire objects are folded, one is always made aware of any folding, and objects must be folded by some direct act (this last point not only encourages faith in the predictable nature of the user-interface, but expresses the fact that the user is in control of the abstraction level of the interface and the skill level it consequentially requires).

Since these concepts may be unfamiliar, they are applied to two application areas:

3.1.1 Linear Command Languages

We may view a default as the user folding system-known information: however the system normally echoes the command which may remain displayed for some time, perhaps until it scrolls out of sight. In this case, the echoed form conceals information from the user about the history of the system's actions. Adhering to (**b**), the echoed form should minimally indicate that a default has been assumed. Adhering to (**e**), the user should be able to determine what default has been taken.

Macros are a very direct way of permitting the user control over abstraction. If we adhere to the suggested guep (**a**), the system should impose syntactic constraints on the bodies of macros. (Most macro interfaces fail to do this.)

3.1.2 Menu Based Systems

Single-level menus often grow to such size that the entire range of options cannot be displayed. Unfortunate systems exist which flaunt (**b**) and therefore surprise their users when they select non-displayed choices.

If rest of the menu is folded (for example, by providing an option 'others'), adherence to (**e**) suggests that a function must exist to return to the enclosing menu. Of course, if the menu is arranged as a tree, (**e**) suggests that this 'go back' function is uniformly available.

3.2 "What You See is What You Get"

The well worn maxim "what you see is what you get" is an ideal example. In simple English, this phrase specifies certain properties of a user interface which, with a little explanation, may be used for the user to develop hypotheses about system behaviour. The phrase may also be placed on a more formal base as follows [5]:

A display can be considered a function **content->view**, with **view** perhaps as **position->symbol** (= **pixel***). Operations on **contents**, such as edits are **content** transformations (**content->content**) with corresponding display transformations (**view->view**) to provide feedback on the outcome of the edits. The "what you see is what you get" simply requires the display function to be a morphism of the **contents** and **view** transformations. The user can then equally view commands acting on his data (**content**) or on what he can see displayed (**view**): this may be non-trivial to specify because in most applications the display function is a projection with no inverse, so some 'obvious' (easy to use) **view** transformations are complex **content** transformations. Indeed, most user interface problems stem from the non-uniform conventions for the display transformation: obscure quoting and meta conventions are introduced (cf §3.4). Consequently, a user interface which adheres to the "what you see" guep may well be more consistent and easier to use, but it is very likely to be much harder to implement.

It should be noted that the accepted interpretation of "what you see" by the computer-science community is different from the semantics presented above. Here, we have simply taken a workable phrase and shown how it might be utilised — after all, users who are presented with this guep need not know of its other (weaker) connotations. The non-standard semantics are explored in [6] and the usual interpretation is discussed further in §6.2.

3.3 "It Can Be Used With Your Eyes Shut"

Being able to use a system 'with your eyes shut' clearly implies a very predictable interface. But if the user really did shut his eyes, to be able to make effective use of the system, the interface would have to be mode free and indicate diagnostics in a non-visual mode (e.g. audibly) and rapidly, to be synchronised with the occurrence of the error, rather than at the end of a unit utterance in a dialogue. In a keyboarded application this requires per-character interaction and a postfix (or function key) command structure. Already we have quite definite technical constraints consistent with the view of the interface the user may have been encouraged to elaborate from exactly the same principle. See [7] for further discussion.

3.4 Gueps for Quoting Mechanisms

Any general purpose system will require quoting mechanisms. For example, a text editor must allow a user to issue commands and to enter text to a file which, for example, could document those commands. When the commands are entered as documentation, they should not be executed and therefore they have to be quoted, that is, interpreted in some non-command mode. In addition, a text editor would normally provide a command which allows the user to search for textual patterns: the patterns will use metacharacters which may be overloadings of textual characters. To search for a textual character which is overloaded requires the character to be quoted.

Perhaps quoting mechanisms most frequently arise where different input/output devices have different resolutions or character sets. Users are confused by the consequent overloading e.g. "^K" denoting 'control-shift-K', not 'up-arrow' 'K': the problem would not arise if there was a displayable form for 'control-shift-K' compatible with the keyboard legends. (In this particular case quoting is implicitly determined by context.)

Few systems impose any consistency over quoting mechanisms, and a number actually have misleading documentation in this respect†. This is probably because quoting is tedious to specify and usually slips past the formal specification. (A set theoretic approach is used in [8].) This often results in an unnecessary

† For example, **lex** [16] documentation asserts that \ followed by any character denotes itself. So * denotes a star, \\ a backslash, but \n a newline! An alternative quoting mechanism provided by **lex** is to place literal text between quote marks. There is no simple relation between the two quoting mechanisms: "n" ≠ \n, but "*" ≡ *. The latter mechanism cuts across the phrase structure (e.g. "ab"* ≡ "a"("b"*) where * is a postfix operator) and, indeed, is subordinate to certain operators (e.g. ["]"] is ill-formed).

profusion of unrelated quoting mechanisms, some of which cut across the phrase structure of the interface.

A guep is needed which might express some of the following constraints: there is one quoting mechanism which is permanently available (e.g. there is a dedicated 'quote' key); quoted composite objects become atomic; the quoting mechanism superordinates all other functions.

4 CHARACTERISING GUEPS

Gueps will be most practical as a contribution to design method, rather than as a contribution to catalogues of acceptable interface techniques. To this end we must recognise what characterises gueps.

For the user, gueps explain in straight-forward language higher-order properties of the user interface which affect its 'feel'. For example, a guep may verbalise a perceptual-motor routine, or a group of related routines (cf §5.1). Or the principle might define a second level grammar [9] or predicates over the formal specification:- there are many possibilities and until further research indicates a particularly promising form it will be better to treat 'generative user-engineering principle' as a generic term. For the designer, the principles constrain the interface to avoid what has been termed 'interaction uncertainty' [10] or 'under-determination' [11] — as the user interface is powerful enough to exhibit any behaviour, the user has less confidence that it will exhibit any particular behaviour.

Thus gueps constrain the machine and explain the interface — in a manner which was not previously accessible to the user, and once verbalised may be used by the user constructively, probably using an acquired or taught inference procedure.

5 THE CHARACTERISTICS OF SYSTEMS MOST EFFECTIVELY DESCRIBED

Necessarily, gueps have an effective domain, and it is constructive to characterise that property of interactive systems which would make them more suitable for applying gueps.

It seems clear that the purpose of a guep is partly to make a property of the system 'second nature' (autonomic) to the user. This can only occur when the user's reasoning can leave symbolic processing and, instead, have rapid feedback of progress and so on. In short, the interface should be 'manipulative'. The condensed observations in the following section are largely due to Ben Shneiderman, whose excellent paper should be consulted[12]. Other positive comments have been made about manipulative interfaces in the literature, e.g. [13].

5.1 Manipulative Interfaces

User interfaces which support direct manipulation use a fixed and commensurate representation for application objects (often a 'real world' view or reasonable icon). The actions the user may perform affecting the objects are atomic; consequently the user does not have to reason about command composition to achieve logically immediate goals. As all atomic actions are rapid, they may be used incrementally and reversibly.

An example of direct manipulation in an interface is the use of four cursor control keys (i.e. keys labelled 'up', 'down', 'left' and 'right') instead of textual commands (such as 'move 23,47'), requiring numerical coordinates or offsets. Manipulative interfaces provide immediate feedback on each atomic action (e.g. 'per-character' interaction). On typing the key labelled 'up', the cursor would move up; whereas with the command form, the cursor cannot be moved until the multi-keystroke command 'move how much' is completed. To determine 'how much' cognitive processing (e.g. counting) is required, in distinction to the (iterative) perceptual-motor processing used with atomic commands.

A manipulative interface supports layered learning: a manipulative interface may be used immediately by novices (probably after a hands-on demonstration), and with growing experience the user will be able to use it more and more effectively without abandoning his initial perceptions.

With manipulative systems, casual users can retain essential concepts readily, yet expert users can work rapidly. Users experience less anxiety because the system is comprehensible, because their actions are invertible. Since objects have a fixed and commensurate representation it is plain whether the users' actions further their immediate goals. Indeed, sophisticated diagnostics are rarely necessary. Using a direct manipulation system is self-motivating.

5.2 Self-Demonstrating Interfaces

If a system uses direct manipulation, it may be demonstrated to the user by the system itself. This is obviously easiest to do on a graphics terminal using light pen or touch screen for input (as a finger or light pen might be displayed on the screen, demonstrating the possible user actions explicitly). A self-demonstration approach cannot be so direct using a textual command system; and the prejudice that 'a system is not easy to use if it needs help' presumably arises because non-manipulative, non-self-demonstrable, systems are manifestly harder to use.

5.3 Passive Interfaces

A user interface is passive if the manifest

action and object worlds are commensurate (for example, the commands and their operands must be mode-free); passivity requires an interface to be uniformly manipulative. A manipulative system is a candidate to be well described by a guep, but for the principle to be uniformly applicable, the user interface must in addition satisfy passivity[14].

Passivity is essentially a restraint on the system not to do too much in anticipation of the user (or worse, occasionally do too much): this anticipation may occur either during the system design or during a particular dialogue. Simply increasing passivity in the obvious way, by decreasing hidden activity, variation and complexity but keeping the dialogue style constant may be too restrictive — instead a different, more manipulative style should be chosen and passivity is thereby increased without loss of effectiveness.

6 A WARNING AGAINST PSEUDO-GENERATIVE PRINCIPLES

There is a class of principle which superficially meets the criteria for gueps. Two outstanding examples are (1) the 'desk-top model', that is, the interface should simulate a user's desk-top displaying appropriate icons mapping onto objects which might be found on office desk tops, such as calculators and spread sheets. And (2) the "what you see..." principle as conventionally interpreted.

6.1 The Desk-Top Model

It will expose the misconception in the desk-top view of man-machine interaction if you consider the horseless-carriage period at the beginning of this century. The initial approach to designing cars followed the obvious 'generative' principle of being compatible with the functional predecessor. This resulted in machines which were no doubt familiar and 'easy to use' but which pushed carriage technology to limits, e.g. in suspension, steering, coachwork and so on. The principle (not that it was ever espoused as such) led to the exposure of limitations in contemporary technology — far from constraining it. So far as I am aware the horseless-carriage period achieved no standardisation in the user interface, not even the development of the steering wheel. Similarly, the desk-top approach in office automation leads inevitably to display processing requirements (e.g. to scroll A4 text in real time in arbitrary direction) which are certainly an impetus to technological development but may not be a valid approach to designing a user interface *per se*.

6.2 What You See is What You Get

As conventionally interpreted, "what you see..." implies equivalent resolution for hardcopy and display devices (when there is usually an order of magnitude difference). It may also be taken to imply real-time formatting and screen updates. Again, the principle primarily encourages systems research rather than either an improved user interface or research towards one. See [6].

7 DIRECTIONS FOR RESEARCH

7.1 In Formal Specification

Gueps may be more defined in terms of formal requirements (e.g. as assertions about the specification). Whether algebraic (cf [15], which formalises such questions as, "Is it the case that pictures are not transparent or even translucent? I.e. if two pictures overlap does the bottom one have no effect on what one sees through the top one?") or constructive methods (such as VDM) are most suitable remains to be determined. Some of the gueps appear to require an explicit time metric and this is not satisfactorily modelled by any existing formal technique.

7.2 In User Interface Design Methodology

It is not clear how gueps may be used prescriptively in general, and a design methodology is wanting. For example, some gueps are very general and allow considerable choice over design.

7.3 In User Interface Evaluation

The gueps may alternatively be selected for their psychological effectiveness. Psychological experiments might establish whether gueps have any significant effect on end users — which is possible because the interface is better designed or because the users know more about the form of the design. Whether the effect, for example, is limited to accelerating the time to reach criterion; what gueps the users establish for themselves in the absence of explicit gueps; whether, conversely, semantic-free gueps have any effect.

These three research areas need to interact, to avoid unintelligible gueps which are helpful in design, or helpful gueps for users which are too vague for formal expression. It is also possible that there exist principles (i.e. certain forms of user-interface consistency) which are actually detrimental to usage, in which case the stronger hypothesis of §1.1 is falsified in an unhedged form.

8 SUMMARY

This paper has been an exploratory introduction to 'generative user-engineering principles', and has not been intended to be superficially rigorous when considerable effort is still required to find useful definitions. The intention has been to demonstrate the potential for higher-order guiding principles which may be used throughout design and help bridge the gap between designers and users.

Generative user-engineering principles are easily understood and used both by user and designer. They can be expressed in such terms that

- The user interface may be designed top-down, using gueps as guidelines to select appropriate low-level features.
- The user has a sound basis on which to construct an understanding of the system, even before using it.
- The user may generalise his knowledge reliably. The user is confident as to what has happened (e.g. after an error), and does not need debugging skills.
- The user is encouraged to use his skills fully. Gueps can help enhance the view that what the user does is real and not abstract. This is especially motivating.
- The designer can use gueps to meet clearly defined user expectations, often with specific techniques.
- A manipulative and passive style of interaction is encouraged.

Having once suggested a basis for the user model, the designer is under an obligation to ensure its coherent implementation through careful system design, which should maintain the model as understood by the user — this approach will entail evaluation and retrospective refinement.

I believe that generative principles, plus attention to detail, already provide a constructive approach to the top-down design of effective, acceptable, interactive systems. At the very least, even if a guep is ergonomically unsound, an interactive system explicitly designed around it will have more internal consistency and be more clearly documented than the average interactive system available today.

9 ACKNOWLEDGEMENTS

Michael Harrison, John Long, Ian Pyle, Andy Whitefield and others made major comments on earlier drafts of this paper for which I am very grateful.

10 REFERENCES

[1] Carroll J. M. & Thomas J. C., Metaphor and the Cognitive Representation of Computing Systems, IEEE Transactions on Systems, Man, and Cybernetics, **SMC-12**, 107-116 (1982)

[2] Baker C. L., JOSS: Rubrics, P-3560, RAND Corp. (1967).

[3] Smith S. L., User-System Interface Design for Computer-Based Information Systems, ESD-TR-82-132, MITRE Corp. (1982).

[4] Engel S. E. & Granada R. E., TR 00.2720, IBM Poughkeepsie Laboratory (1975).

[5] Harrison M. D. & Thimbleby H. W. Formalising User Requirements for a Display Editor, University of York, To appear.

[6] Thimbleby H. W., 'What you see is what you have got' - a user engineering principle for manipulative display? 70-84 in Balzert E. H. (ed) Software Ergonomie (Teubner, Stuttgart, 1983)

[7] Thimbleby H. W. Character Level Ambiguity: Consequences for User Interface Design, International Journal of Man-Machine Studies, **16**, 211-225 (1982)

[8] Sufrin B. Formal Specification of a Display Editor, PRG-21, Oxford University Computing Laboratory (1981)

[9] Green T. R. G & Payne S. J. Higer-Order Rules in the Perception of Grammars, Memo 544, MRC Social and Applied Psychology Unit, Sheffield University (1983)

[10] Hansen W. J., Doring R. & Whitlock L. R. Why an Examination was Slower On-line than on Paper, International Journal of Man-Machine Studies, **15**, 507-519 (1978)

[11] Thimbleby H. W. Dialogue Determination, International Journal of Man-Machine Studies, **13**, 295-304 (1980)

[12] Shneiderman B. Direct Manipulation: A Step Beyond Programming Languages, IEEE Computer, **16** 57-69 (1983)

[13] Brooks F. P. The Computer 'Scientist' as Toolsmith — Studies in Interactive Computer Graphics, in Gilchrist E. B. (ed) IFIP Conference Information Processing '77, Toronto, 625-634 (1977)

[14] Thimbleby H. W. Interactive Technology: The Role of Passivity, in Bensel C. K. (ed) Proceedings Human Factors Society, Boston, **23**, 80-84 (1979)

[15] Guttag J. & Horning J. J. Formal Specification as a Design Tool, in 7th ACM Symposium on POPL, Las Vegas (1980)

[16] Lesk M. E. Lex — A Lexical Analyser Generator, Computer Science Technical Report No. 32, Bell Laboratories, Murray Hill, New Jersey (1975)