Consistency, Standards, and Formal Approaches to Interface Development and Evaluation: A Note on Wiecha, Bennett, Boies, Gould, and Greene

JONATHAN GRUDIN
University of California, Irvine

Wiecha et al. [5] describe ITS, a system for interface construction. Application developers use ITS to compose style rules that are applied during compilation. These rules result in a certain interface conformity, described as "consistency," and are said to reduce the development effort considerably. I have no reason to dispute the authors' central claim, that their interfaces are useful, usable, and easy to develop. This is a response to their secondary claim, that their interfaces owe their virtue to consistency, which is incorrect.

In [2], I illustrated a range of problems with the concept of interface consistency. In some circumstances it is difficult to define consistency or to determine the appropriate features or dimensions on which to be consistent. Consistency is a trade-off against other goals: at times, it is not the best design strategy. Random or careless design is to be avoided, but the best overall strategy is generally to use all available means to obtain the best possible understanding of the prospective users and their work situations. Consistency and other design rules are best seen as guidelines that may have to be violated for the benefit of users.

ITS generates interfaces from design rules. Wiecha et al. argue for establishing a set of design rules to cover all allowable interaction techniques and then prohibiting exceptions (p. 233). They claim that resulting interfaces will be both good and consistent or perceived to be consistent. The generality of this claim must be evaluated carefully, with an eye to possible limitations. I

Author's address: Information and Computer Science Department, University of California, Irvine, CA 92717; email grudin@ics.uci.edu.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1992 ACM 0734-2047/92/0100-0103 $01.50

will show that not all design cases can be prespecified or translated into rules that contemporary systems can act upon. Furthermore, interfaces generated by ITS need not be consistent in the commonly accepted meaning of the word. Finally, not all good interfaces are perceived to be consistent. The issue of consistency is at the heart of the theory of interface design, and progress requires that we move past the oversimplifications of Wiecha et al.

In making their case the authors critique a menu design example from [2]. The example, described in detail below, demonstrates that a good design need not be consistent. Unfortunately, without mentioning it, Wiecha et al. omitted part of the example and altered other aspects. The simplified result does not convey the complexity of the design problem and allowed the authors to conclude that "consistency can mean a hierarchy of default rules and subrules for special cases. In our view, users will in fact perceive the consistency . . . even if that behavior differs from the default rule." [5, p. 216].

In the next section, the original example from [2] is considered in greater depth. Again, we see that the design it describes cannot be characterized as "consistent." Furthermore, it can be handled by explicit, computer-applied rules only to a degree and only under very close human supervision. More generally, Wiecha et al.'s notion of "perceived consistency" proves under examination to be untenable, an effort to simplify a reality that is resolutely complex. Design optimization is an ongoing struggle to fit technological pegs into societal holes that do not quite line up, a process that cannot be smothered under a blanket called "consistency," convenient though that would be for all of us.

Formal tools such as ITS have a role in interface development if designed and used with a sensitivity to their limitations. Consider artists, whose hands never leave their tools but whose attention is elsewhere, divided between the object being created and the inspiration in the world beyond. Similarly, a principal focus of good developers is to increase their knowledge of the application domain and the people working therein. A tool can free them to do this or it can less helpfully constrain their use of what they learn.

An Example: Default Selections in "Pop-Up" Menus

Developers have designed menu defaults for decades, initially in the context of full-screen menus. When a menu appears, one item can be preselected. This "default" selection can be invoked by simply pressing a mouse button or a key (e.g., ENTER or RETURN). A developer can specify the default items or can provide a process by which a default is chosen while the progam is running. A good set of defaults reduces the menu scanning and navigation required of users.[1]

[1] Other types of menus, such as pull-down menus and ready-only menus listing options that must then be typed, do not have preselected default items. However, every menu favors one item—the item closest to the cursor, most likely to be seen first. Thus, designers always face the same issue: which items to place in favored positions. Users, however, experience a substantial difference between a default selection that can be invoked without looking at the display (by pressing a key or button) and an item that is easily selected but that does require visual attention or navigation.

Design approaches have included defaulting to the first item on the list, the item thought to be most frequently chosen overall, or the item chosen most recently from the menu by the present user. The last option was widely adopted for full-screen menus of the 1980s. It did not please all of the people all of the time, but it was successful. Users often have habitual paths through hierarchical menu structures, and in some circumstances perform one action repeatedly.

Examining another case, Macintosh pull-down menus have no explicit default selection, but the first items in a menu are easier to select, so developers are encouraged to place frequently accessed items there [1].[2] To repeat an operation requires repeating the same sequence of actions.

The menus described in [2] were pop-up menus from an Interleaf desktop publishing system. Depressing one of three mouse buttons brings up a menu at the mouse-controlled pointer location. The default menu item appears in reverse video. A user can move to another item before releasing the button to carry out the selected operation or can move the pointer off the menu to select nothing.

A nice feature of this system is that the default can be selected extremely rapidly by just tapping the button, without even waiting for the menu to appear. Of course, this could be a dangerous habit if default selections are not uniformly safe.

Three coexisting defaulting strategies. The design did not consistently use any of the defaulting strategies described above. In many situations, it defaulted to the last item selected. This is shown in Figure 1, row 1. To italicize text requires four steps after bringing up the menu: moving the cursor to Fonts, Moving to Italic when the Fonts menu appears, moving to On when the Italic menu appears, and releasing the mouse button. Once this has been done, popping up the menu again brings up the entire set of menus, as shown on the right. Italic On is now the default and can be applied with a single button click.

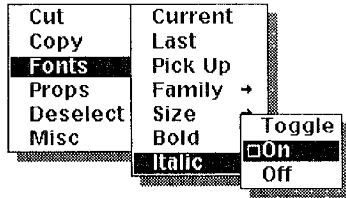
Row 2 shows a different defaulting strategy: Alternation. First, Copy is selected from the pop-up menu (a). The next time the menu is popped up, it defaults to Paste (b). After that, it returns to Copy (c). Since Paste usually follows Copy, this again is very efficient.[3] Alternation is also used where it is less obvious but quite elegant. A system utility allows one to crop and save a bitmap image of part of the display. First one positions a "window" over the desired portion. This window can be moved and can be resized from the lower right corner. The default operations alternate between Move and Resize. This is startling at first, but it leads immediately to very natural, efficient operation.

[2] This can lead to tradeoffs through conflicts with other recommendations, such as the recommendation to group related operations.

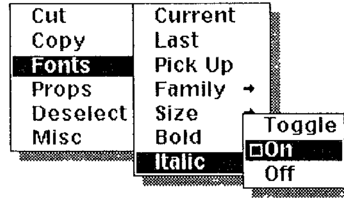
[3] Multiple copies are created by repeatedly clicking the button, with each Copy-Paste pair creating an additional copy.

INITIAL SELECTION

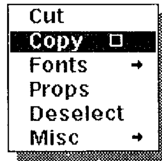
SUBSEQUENT MENU DEFAULT(S)



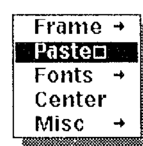
1. a) 'Italic On' selected in 3 steps.



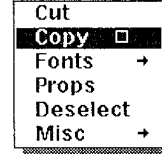
b) 'Italic On' is then the default.



2. a) 'Copy' selected in 1 step.

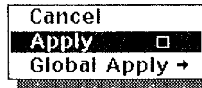


b) 'Paste' is then default



c) Then 'Copy.'

3. a) 'Global apply' in 2 steps



b) 'Apply' and then 'Cancel' defaulted.

Fig. 1. Menu default patterns vary with aspects of the task.

Row 3 shows a third defaulting strategy. A user wants to apply a property to all instances of a certain component; for example, to change all headers to a 12-point font. The first pop-up menu defaults to Apply, which would change only the currently selected header, so the user selects Global Apply. Then a confirmation step is required, moving the next menu pick from Cancel to Confirm (a). Following this, if the user applies another property change (e.g., to center all headers in the page), the first menu defaults to Apply again, and the final confirmation menu still defaults to Cancel (b). In short, the entire sequence must be repeated as before.

The third example violates a rule that covers the first two: "default to the expected action." A user virtually always confirms a change, yet that step is forced by defaulting to Cancel; similarly, a Global Apply is more likely to be followed by another Global Apply than by an Apply, yet this step is also forced each time. This inconsistent behavior is desirable because Global

Apply can be "irreversible" or very difficult to undo. The potential destructiveness of Global Apply is not obvious to new users, because its simplest and most common uses are easily reversed; for example, globally changing headers from a 12-point font to a 14-point is reversed by globally changing from 14 to 12. But imagine that half of the headers are in 10-point font and half in 14-point. Globally Applying a 12-point font will change them all, leaving no record of which were 10 and which 14. With no Undo capability, restoring the original settings can be difficult or impossible.

A fourth strategy. These three approaches are not exhaustive. They do not address the design of defaults for menus that are accessed following unrelated actions. For example, say that the font size menu is brought up following some typing. The default is the existing font size value, although it is unlikely a user will choose that because it results in no change. However, under these circumstances the system cannot guess what the user wants, so the default is to display the current state, which is both informative and nondisruptive if enacted. A similar problem, with the same solution, arises when a user changes the font size of some text from 10 to 12 and then brings up the font menu again with the same text selected. It is unlikely that the user wants to leave it at 12, but there is no way to know whether the user will return to 10, continue up to 14, or choose something else.

Where rules collide. This case is further complicated when only two choices exist; for example, the font is boldface and a user accesses the style menu Bold. The user almost surely wants to turn off Bold, but the designers have a problem: to be consistent with the fourth strategy requires "display the current information" by defaulting to Bold On, whereas consistency with the second strategy, "Default to the expected action," requires a Bold Off default. There is no easy way out: either solution creates an inconsistency. The Interleaf developers did find a middle ground: they display the current information, but allow users who expected Alternation to change the future system behavior to Alternation by including a third option, "Toggle," shown in the first row of Figure 1.[4]

An infinite number of coexisting menu default strategies. In principle, any number of defaulting strategies could be useful. In practice, a large number will be useful for many users. Consider the alternation strategy, applied to two-step operations. In some application domains, three-step operations are common. In fact, many simple macro languages are used to create stereotyped "default operation sequences" of precisely this kind. Such macros are often left to "end users" to design, either because the practice is particular to that user or because the developer has too little awareness of the use of the system to provide it.

[4] The reader might think that the author is an Interleaf developer. Not so, and the interface did have some questionable features, including a few apparently unmotivated divergences from the patterns described here.

Implications for Tools Based on Explicit Rules

The implications of the menu default illustration for rule-based tools differ according to whether the tools are semiautonomous or highly subservient in the development process.

Semiautonomous tools. These are tools that examine an interface and identify design errors or "inconsistencies," or that generate an interface from the code or a description of the code. One would like to automate the development or verification of seemingly low-level design decisions such as menu defaulting, but unfortunately, it cannot be done in the case described.

There is no way of knowing from the code which operations will be used in alternating sequences, whether operations will be destructive and difficult to undo, and so on. Some answers can be worked out logically, but often the only way to discover these things is through the observation of actual use. The Move-Resize alternation is a strong but subtle phenomenon. The destructiveness of operations such as Global Apply is not always obvious.[5]

The correct choice of menu defaulting is dependent not only on the context of use but also on other system capabilities. Introducing a feature that allows an easy recovery from an accidental Global Apply would change the optimal design. If such system changes are more indirect than an Undo command, existing interface generators or verifiers would not catch them. (In theory such verification could be achieved, but to do so would require advances in artificial intelligence.)

Semiautonomous tools can perhaps help with design features that are less complex than menu defaulting, and they can serve in an advisory role, but their use risks encouraging suboptimal design (however, see the discussion below on when suboptimality is acceptable).

Passive tools. A tool entirely under the control of a development team, in the sense that artists' tools are under the artists' control in the metaphor introduced earlier, is a different matter. For example, developers could identify half a dozen defaulting approaches through experience and observation. For a particular menu, they could empirically determine "This is a Number 2," and a tool could translate that into the appropriate detailed design—say, Alternation. If the developers, through observation, discovered a need for a seventh defaulting approach, they could modify the tool by building it in.

To my understanding, this is what ITS does. ITS is an implementation tool more than a design tool. It promises to be useful in cases for which design rules can be stated explicitly and simply. It must be carefully guided by a design team. The developers cannot specify rules such as "If an operation is potentially destructive, provide a confirmation step," because systems cannot reliably make such determinations. However, the developers can determine

[5] Wiecha et al. [5] changed this example from Global Apply to Delete; the obvious destructiveness of Delete hides the potential subtlety of case identification. They also eliminated the Alternation case by changing Copy to be "Default to the last item picked." The intent was perhaps to avoid a detailed digression, but the result was misrepresentation by oversimplification.

that a feature is potentially destructive and specify that a confirmation step be created. Unfortunately, there are cases for which the rules governing optimal design are unknown. These include the keyboard layout examples in [2], not discussed by Wiecha et al., and may also include similar problems such as the organization of menu items, in which a complex interplay of physical, perceptual, cognitive, and workplace factors enter into each design problem. In these situations formal systems such as ITS have no appreciable role.

Although ITS is potentially useful, the claim that it will provide "consistent" interfaces is a harmful error. This claim fosters the common confusion between consistent and good interfaces. Most people agree that some consistent interfaces are bad, but the fact that some good interfaces are not consistent encounters resistance.

"Perceived Consistency" Is Not a Useful Concept

Wiecha et al. [5, p. 216] argue that the menu defaulting scheme described above is consistent because "Consistency can mean a hierarchy of default rules and subrules for special cases." But *anything* can meet this definition of consistency. Consider English spelling. Nonnative speakers (or anyone who recalls grade school) know that it is wildly inconsistent. Yet it can be entirely described by a hierarchy of default rules and subrules for special cases: 'i' before 'e'; except after 'c'; and when pronounced like 'a' as in 'neighbor' and 'weigh'. A finite (but very large) number of "subrules for special cases" later, and we are through.

Wiecha et al. claim that "users will in fact perceive the consistency in behavior of dangerous operations even if that behavior differs from the default rule." This is demonstrably false. As noted above, beginning users may not be aware that an irreversible operation such as Global Apply is dangerous. Some are surprised by the system's behavior, expecting it to default to Global Apply the second time. They perceive its behavior to be inconsistent. Because the operation *is* dangerous, however, it is a good design decision. Similarly, alternations such as the Move-Resize behavior on bitmap cropping are a startling surprise when first encountered, perceived as being inconsistent, but users quickly see their utility. So good design does not require behavioral consistency *or* perceived consistency from the perspective of beginning users.

Furthermore, "perceived consistency" as used by Wiecha et al. is not necessarily either good or a form of consistency. Experienced users often perceive whatever they are accustomed to as being consistent. It is consistent with their experience, but that may be all—it need not be internally consistent or a good design. Native speakers do not notice most inconsistencies in their own spoken language. Living abroad, I have been in many conversations where those learning a new language point out inconsistencies that escape the notice of native speakers, who take the inconsistencies for granted. They perceive "design consistency" where there is none.

Wiecha et al. do not restrict the number of "rules and subrules for special cases" that define consistency. As noted, we can thus generate rules by which

English spelling qualifies as consistent. But a "common sense" notion of consistency limits the number of rules to the smallest number "necessary" [4]. To have one spelling pattern for each sound is fine, but two spelling patterns for one sound is not, even if a rule explains it ('i' before 'e,' except after 'c'). But how many are "necessary"? In theory such a determination requires a complete knowledge of all contexts of use, a complete knowledge of human psychology, and so forth. Developers who design for subtle but important distinctions (as in recognizing the destructiveness of Global Apply or the surprising utility of Move-Resize alternation) can appear to new users to violate this aspect of consistency, and developers who address competing goals, for example by providing shortcut alternatives for high-frequency operations, do violate consistency on behalf of experienced users [3]. Their interfaces might be optimal but they are not consistent.

Interfaces produced by ITS are not necessarily consistent, but they are predictable if one knows the rules that generate them (as are English spelling and English pronunciation). This is potentially a positive accomplishment; it eliminates some sources of careless error and communicates information about design alternatives to developers within and across projects. Is the tool flexible enough to accommodate the complexity of many design decisions? Can a new rule or subrule be added easily whenever a need is found? These can be organizational issues as much as tool issues. A positive use of such a system would be to discourage local optimizations that conflict when integrated into a larger system; a negative use would be to enforce adherence to an overly restrictive set of design alternatives. In any case, such tools will not eliminate the need for skilled interface developers to examine design choices in real work contexts, in order to identify the rules that apply and to develop new ones when necessary.

Design Optimization, Consistency, and Standards

Where do we get the false impression that every good design must somehow be "consistent"? Many bad interfaces are traced to inconsistencies, so perhaps we assume that good interfaces must be consistent, although this does not logically follow. It is usually a criticism to say that a person "behaves inconsistently," so perhaps it seems bad for an interface to behave inconsistently, too. But of course good human behavior is actually highly variable, adapted to the situation at hand. Our choice of words shifts according to whether we are addressing children, peers, casual acquaintances, or authority figures. We probably could not categorize all social situations in advance, yet we adjust our personal "interface" to each situation when we encounter it. The human-computer interface developer requires the same flexibility and the same awareness of the context of a dialogue.

One concern is that after acknowledging that there is foolish or bad consistency and that some optimal designs are not consistent at all, we are not left with a strong consistency guideline.

And unmotivated inconsistency does underlie many bad designs: In the absence of knowledge about users' requirements or work practices, the best strategy can be to choose a rule to follow or a dimension on which to design

consistently [2]. The design will probably not be optimal, but it may be good enough.

This leads us to examine the concept of design optimization. A common goal is to optimize for a "typical user," about whom a limited amount can be said. With no more information than this, internal interface consistency can be beneficial. It usually facilitates learning, although not always [2]. But there are situations in which other factors override consistency in importance, such as the desire to be fast and efficient in the example of Alternation or to avoid catastrophes in the example of Confirmation [3]. These trade-offs even affect interface design optimization for the vague "typical user."

However, optimization for individual use is just one of the factors influencing design. Other factors include optimizing the design for marketability, for ease of implementation, or for collective benefits that outweigh drawbacks for individual users. Consistency can further such goals, perhaps at the expense of usability. We know that the QWERTY typewriter keyboard layout is suboptimal for individual users in isolation, but the associated costs of changing are so great that we settle for suboptimality.

Formal or de facto interface standards are a statement that further optimization on behalf of individual users will not be accepted, that the standard is good enough. This provides certain advantages, including consistency (at least external consistency across systems; one can of course standardize on an internally inconsistent interface).

For many years, developers provided one or another consistent approach to menu defaulting, such as "menus will always default to the last item selected on that menu." Consistency provides many advantages. For developers, it is easier to implement and requires no judgment. A program can verify compliance. Users learn it quickly. However, users of those menus also paid a price in reduced efficiency. We can't have it both ways.

ACKNOWLEDGMENT

This note benefited from conversations with Donald A. Norman, comments from Tom Erickson and Jakob Nielsen, and useful editorial suggestions by Brad Myers.

REFERENCES

1. APPLE COMPUTER, INC. *Human Interface Guidelines: The Apple Desktop Interface*. Addison-Wesley, Reading, Mass., 1987.
2. GRUDIN, J. The case against user interface consistency. *Commun. ACM. 32*, 10 (1989), 1164–1173.
3. GRUDIN, J., AND NORMAN, D. A. Language evolution and human-computer interaction. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (Hillsdale, N.J. 1991), Lawrence Erlbaum Associates, 1991, pp. 611–616.
4. REISNER, P. What is inconsistency? In *Proceedings of the INTERACT'90 Conference on Human Computer Interaction*. North-Holland, Amsterdam, 1990, pp. 175–181.
5. WIECHA, C., BENNETT, W., BOIES, S., GOULD, J., AND GREENE, S. ITS: A tool for rapidly developing interactive applications. *ACM Trans. Inf. Syst. 8*, 3 (1990), 204–236.