Examples

The following examples provide the intuitive basis for understanding a range of issues related to user modifiability. Each illustrates a particular point and may help to predict the dimensions along which users will/can/want to modify software to suit their own needs.

Framing the question differently poses different problems. For example, if it turns out that the users' assumptions are based on previous experience?

How important is the user's current context? What about what have they learned?

Contend that users who learn one software program learn both what it is and also, what it is not. What it can do, and what it can't do. Well-defined primitives may suggest new possibilities (e.g. object-oriented programming allows you to frame problems in terms of objects in the real world), but at the same time, those primitives limit the ways in which you can think about the problem. While it is true of very powerful primitives that you can frame any problem in the language, it is not the case that it is intuitive or easy to do so. So the choice of primitives affects the way the user thinks about things, implicitly or explicitly.

Do you change users (i.e. teach them to frame problems in a way that maps onto the way the software is partitioned, the "primitives")? Or do you change the software (i.e. make the software conform to the dimensions along which the user frames the problem)? Or do you do both?

What are the kinds of things that can be predicted? How can we find out the mismatch between the user's heuristics and the software's internal model?

Would like the changes to the user to be in the "generally useful" category, e.g. problem-solving skills.

Range of possibilities:

	User behavior	Software behavior
User changes	data from user data from software data from teacher	data from user data from software data from teacher
Software changes	data from user data from software data from teacher	data from user data from software data from teacher

Do you want to have the user adapt? The software adapt? Or both. What does it mean to *modify* software?

The examples are:

1. Lucy’s change in her rules
2. The use of rulesets
3. Buggy
4. Writing styles
5. Stimulus classes
6. Franklyn’s program fragments
7. Programming by example: identify general and specific
8. Editor tutor
9. Repair Theory

Buggy

Background

Example

Students have theories about subtractions -- their errors are not random. As a result, if you can identify the theories, you can figure out why the student made the error and fix the theory.

Lessons

Stimulus Class Formation

Background

Mackay, W. E.

Sidman, M.

Example

Can look at old doctoral dissertation as an example of implicit training, with out direct Sr.

Lessons

The method of presenting information affects the associations users make, even if not done explicitly.

These associations affect the user's assumptions about the system, their predictions about the behavior of the software, and the ways in which they are likely to try to change the software to meet their particular needs.

Writing process

Background

This data was collected from 17 participants in a seminar on computer-supported cooperative work on March 9, 1987.

Example

The class had a discussion about different writing strategies. We discovered that we were pretty different, even though we were all academics trying to accomplish similar tasks. We identified two dimensions:

Writing artifacts:	Notes, outlines, and the final text.
Continuity:	Does the writer throw out the previous version when moving to the next artifact ("fresh start") or copy it and use it in the next stage?

Seventeen people responded as follows:

	Fresh start	Copy	Total
Text only	2	n/a	2
Outline, text	0	1	1
Outline, notes, text	1	2	3
Notes, text	2	2	4
Notes, outline, text	5	2	7
Total	10	7	17

Some people just sit down and write; they do not use notes or outlines (!) and so do not face the issue of whether or not to use the previous artifact (notes or outline) when moving to the next stage.

Of the remaining people, eight make a fresh start when moving from one artifact to another and seven base subsequent work directly on the previous work. The most common strategy was to use all three artifacts: notes, outline and text. Two people said they make a set of notes, then put them aside and write the text. Five people said they started with notes, then formed an outline of the paper, then wrote the text. One person started with an outline, then gathered notes, then wrote the text. Nobody went from an outline to the text. (Despite what they teach you in fifth grade.)

One person expanded the outline into the text of the paper. Three people start with an outline, fill in with notes, and then create the text. Two people start with notes and then rearrange them into the text. Two people start with notes, rearrange them into an outline, then then smooth it into text.

Lessons

People, even those with very similar goals and backgrounds, have very diverse strategies for solving problems and accomplishing tasks. They may all be unique (which would be rather discouraging if one is trying to make predictions) or they may fall into definable categories, from which it may be possible to predict behavior.

Asking a user about their preferred writing process may reveal information about how they already define the system (or the tasks they want to accomplish) which may help us predict the ways in which they want to modify the software, which in turn may help us create software that can be modified along those dimensions.

Problem Solving: programming

Background

Franklyn Turbak and Mike Eisenberg: Study of how students learn to program in 6.001. (In Empirical Studies of Programmers.)

Also, look at Soloway's work on Plans.

Example

Students are given sample programs from which they are expected to generalize programming principles. (The principles are also explained on a more abstract level.)

They found that they could often identify the student's programming misconceptions by the ways in which they chose to copy program fragments (from sample programs) into their programs.

Lessons

Problem-solving: users often go from concrete examples to the general. Hard to generalize.

Programming-By-Example

Background

J. Carroll - CACM article.

Example

You could imagine a system in which the user is asked to create a form and then engage in a dialog with the system to identify which sections of the form are general (e.g. "name", "date"), and which are specific to this particular form (e.g. "Wendy", "24 March").

The software, given some clues as to the domain, should be able to guess which are specific and which are general, particularly, after experience over time. (See Majors, "SW that grows with the user"). More importantly, the user should be able to engage in a dialog with the software to clarify things -- the system says "I think this is what the general form should look like, and here's your example of how it would be filled out" and the user ought to be able to tell the system whether or not the system is right.

Lessons

This isn't a lesson as much as an example of a user-modifiable system that does more than just let the user create macros (which are "by rote" recitations of steps to accomplish particular tasks).

By incorporating the users' knowledge, it is less important for the system to have to predict things.

Look at Rich Pito's work with OLC.

dLens: Diagnostic Tool

Background

Example

You can use a tool like Lens to discover the kinds of activities the organization is engaged in and what it considers important. It can be a diagnostic tool for finding out what the organization does.

Lessons