New interface designs for small molecule crystallography



Horst Puschmann



#### Some general considerations in User Interface Design



Horst Puschmann



#### Introduction







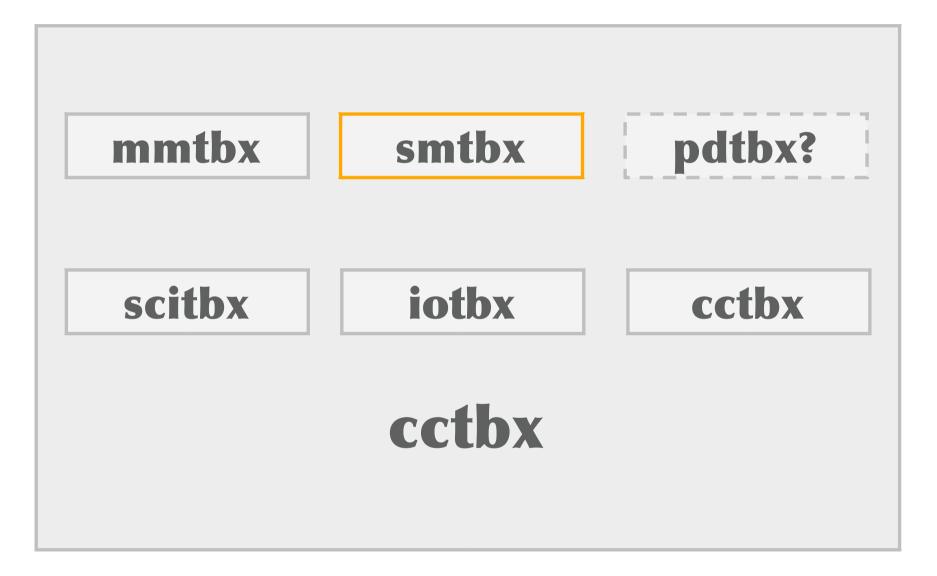
- Modern Crystallographic Program
  - » Based on the cctbx
  - » Toolbox not a program!

Modern User Interface
 » An accessible GUI











(I should have done)

#### **Some Reading:**

#### **Donald A Norman:**

The Design of Everyday Things (Basic Books, 1988/2002)

#### Joel Spolsky:

User Interface Design for Programmers (Apress, 2001)

#### **User Interfaces**

# **History of Interfaces**

- First programs weren't interactive at all.
- Command line: Now what do I do? Whatever you want! Need to memorize all the frequently used commands, or continuously look things up.
- Then 'questions and answer' model appeared.
   Combination of a manual with the program itself.
- More and more choices made this very messy.
   Menus appeared.

## **Disagreement from the start...**

- From early 1970's (when interactive systems first appeared): each generation of interface designers collectively changes its mind about whether users need to be ...
  - » *guided through* a program or whether they should be
  - » *left\_alone* to control the program as they see fit.

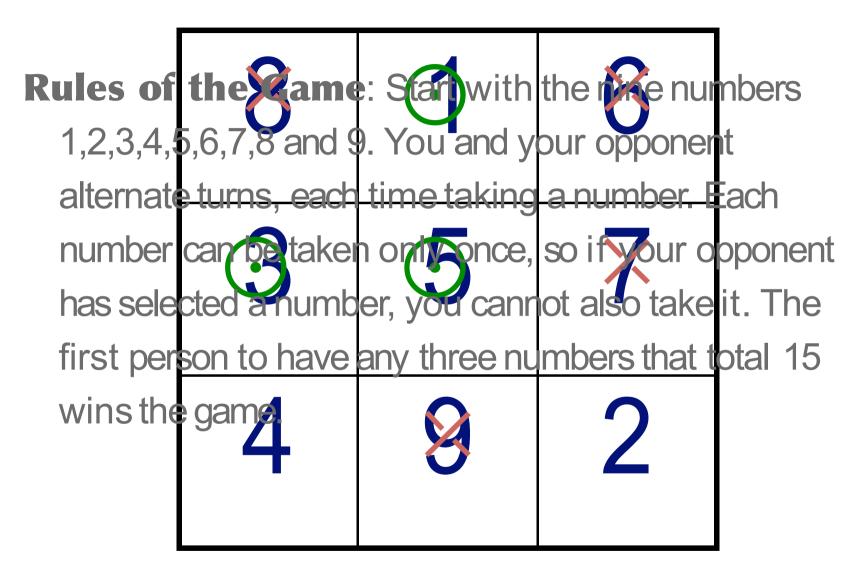
# **GUI? Not interesting! Why bother?**

- Programmers generally *hate* UI programming.
- Good programmers usually think they don't have the skills to do good GUI design.
- Programmers often assume that all their users know as much as they do.
- Scientific programmers often assume that only fully qualified scientists who have a complete understanding of the underlying scientific theory will be using the product.

## **Command line gives you** everything!

• This is true, but you also need to know everything.

#### **The Game**



#### The Listing File

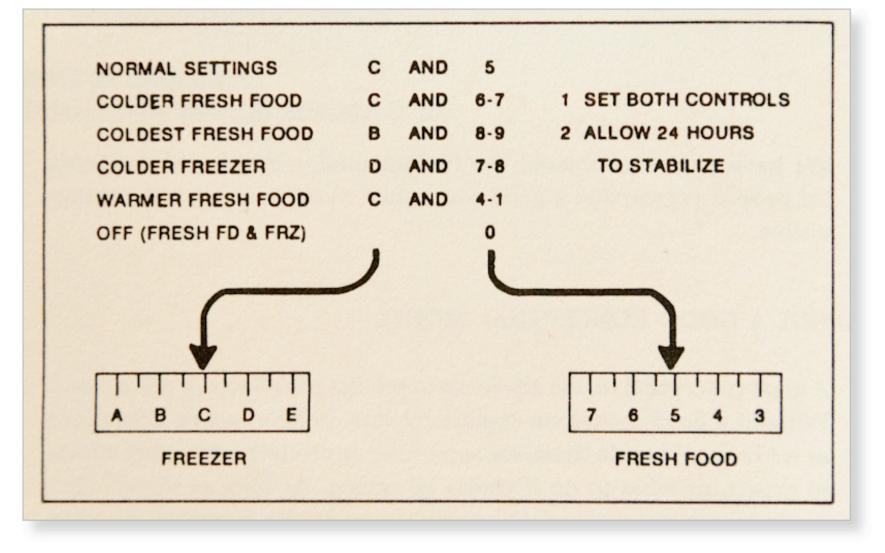
01	3	-0.18442	1.61018	0.71265	11.00000	0.02053
02	3	0.20459	0.96059	0.81032	11.00000	0.03684
03	3	0.24860	1.26972	0.73009	11.00000	0.02790
						0.03420
						0.02879
						0.02676
						0.03223
				/		0.02803
			(			0.04408
						0.02191
		ΤΤ	/			0.06115
						0.29967
						-0.00173
	$\mathbf{O}$					0.02206
				70		0.02366
		<b>- </b>				0.02692
						0.01828
						0.02179
						0.01809
<b>G10</b>	4	0 01202	1 24707	0 00070	11 00000	0.01939
C10	1	-0.01383	1.34727	0.80072	11.00000	0.01783
C11	1	-0.05729	1.03322	0.87461	11.00000	0.02227
C12	1	-0.20696	1.15586	0.85976	11.00000	0.02403
C13	1	-0.45217	1.62315	0.78923	11.00000	0.02996

#### **User Model and Program Model**

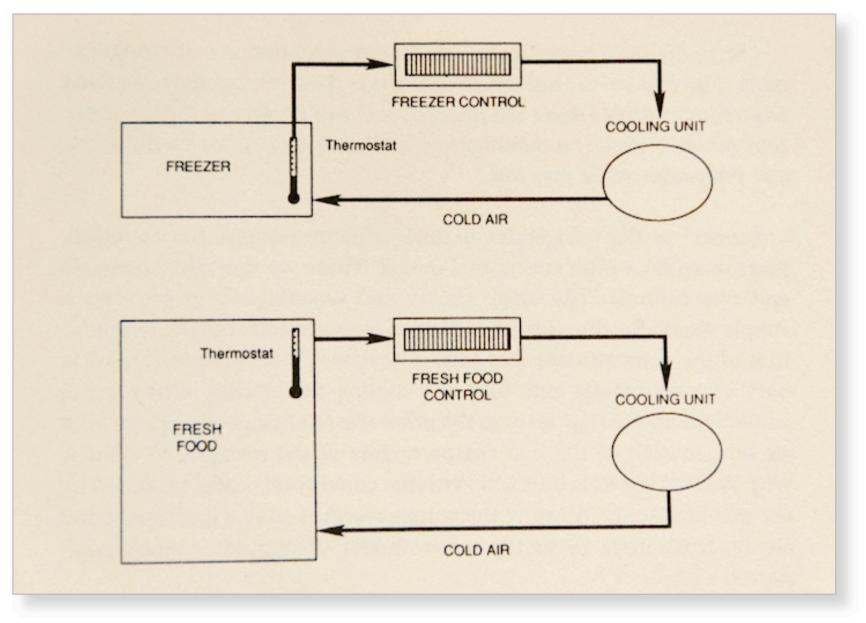
# What did they expect?

- The **User Model**: The mental understanding of what the program will do for them.
- <sup>o</sup> The **Program Model**: Set in stone. The Law.

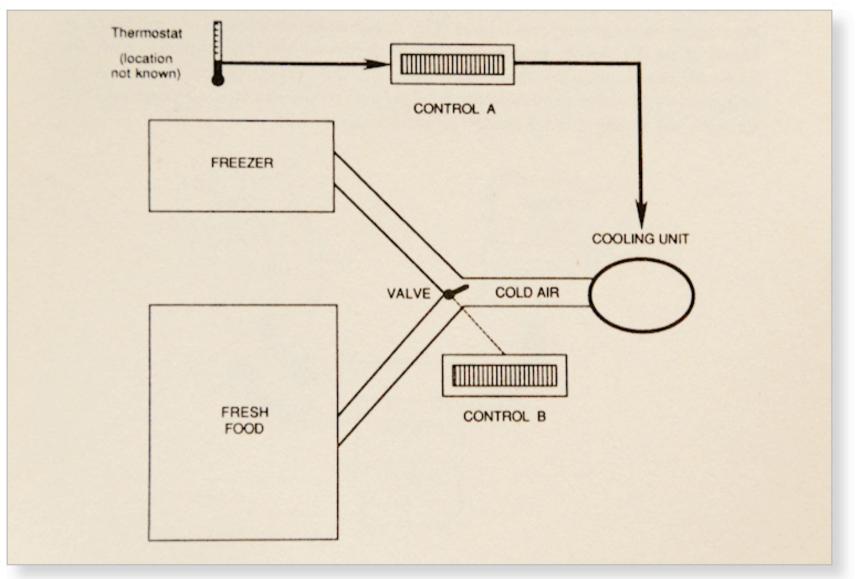
#### The GUI



#### **The User Model**



#### **The System Model**



# So, what did they expect?

- How do you find out what they DO expect?
- You ask.
- You watch.

#### A good interface

"A user interface is well designed when the program model conforms to the user model."

or, in other words:

"A user interface is well designed when the program behaves exactly how the user thought it would."

# **The Design Process**

# Small annoyances do ruin your day

- Sticky key space bar?
- Make the same mistake over and over?
- Flickering monitor?

#### **A Rubbish Example**

For example, designing an outside rubbish bin requires a number of choices between conflicting requirements.

- It needs to be heavy, so it isn't blown away. It need so to be light, so it can be emptied.
- It needs to be large to hold a lot of rubbish. It needs to be small so it doesn't get in the way.
- It needs to be open, so that people can put things in. It needs to be shut so that things don't blow away.

#### **Options, Choices & Preferences**

"Every time you provide and option, you are asking the user to make a decision"

- <sup>o</sup> Design is the art of making choices.
- If you are designing and you try to put the burden of making choices onto the user, you are not doing your job.

# **Process of designing a product**

- Activity-based design: Come up with a list of things that users might want to do.
- <sup>o</sup> **Imaginary users**: Come up with an imaginary user for each potential group of users.

## Who are we designing for?

- At a superficial level we think we're designing for users, but no matter how hard we try, we are designing for who we think the user is, and that means that we are designing for ourselves.
- Until we make the shift and let the users tell us how our software works, it simply can't be usable.

# Who are we designing for?

- We think we're designing for users.
- But really we are designing for who we think the user is.
- And that means that we are designing for *ourselves*.
- Until we make the shift and let the users tell us how our software works, it simply can't be usable.

#### **People make mistakes**

- The same processes that make us creative and insightful by allowing us to see relationships between seemingly unrelated things also lead to...
  - » **Slips**: Form an appropriate goal, but mess up in the performance. Slips are almost always small things and are usually easy to discover.
  - » Mistakes: Form the wrong goal, and you've made a mistake. Mistakes are difficult or impossible to detect

     after all, all actions performed are appropriate for the goal.

#### How many 'goes' do you get?

It usually takes five or six attempts to get a product right. This may be acceptable in an established product, but consider what it means in a new one. The problem is that if the product is truly revolutionary, it is unlikely that anyone will guite know how to design it right the first time. But if a product is introduced into the marketplace and fails, well that's is it. Perhaps it could be introduced a second time, or maybe even a third time, but after that it is dead: everyone believes it to be a failure.

### How many 'goes' do you get?

- <sup>o</sup> It takes five or six attempts to get a product right.
- <sup>o</sup> If it's truly new, nobody quite knows how to design it.
- <sup>o</sup> If it's introduced to 'The World' and fails, that's it.
- Maybe you get a second or third chance, but after that, everyone will believe it's a failure.

#### **GUI design Considerations**

## Consistency

- Consistency helps people learn your program.
- Before GUIs, everyone invented the very fundamentals of the user interface
  - » :q!
  - » С-х С-с
  - » F7
- Ctrl+C, Ctrl+V etc....
- make the program model match the user model!

#### **Just Because Microsoft Does It...**

... doesn't mean it's right!

But at least consider:

- Whether it's right or not, if they are doing it in popular programs then millions of people will think it's right – and at least know it.
- Don't be too sure that it's not right. They spend more money on usability testing than you do. They did it that way because more people can figure out how to do use it that way.

#### Creativity

- Don't be creative: to make a user interface easy to use, you are going to have to channel your creativity into some other area.
- Before you design anything from scratch, you absolutely must look at what other popular programs are doing and emulate this as closely as possible.

## **People Can't Read**

- Users don't have a manual, and if they did, they wouldn't read it.
- In fact, users can't read anything, and if they could, they wouldn't want to.
- So, knowledge that is required for the process, needs to be 'in the world' - i.e. the GUI to augment what is in the head of the user already.

# **People Can't Use Input Devices**

- Keyboard
- Mice
- Colour blindness
- Touch screen
- Trackballs
- Pens

## **People Can't Remember**

 Precise behaviour can emerge from imprecise knowledge for these four reasons:

- 1. Information is in the world
- 2. Great precision is not required
- 3. Natural constraints are present
- 4. Cultural constraints are present

### **Metaphors & Icons**

Icons and descriptions have to be meaningful!

#### **Automation & Heuristics**

- Do something automatically that the user *probably* wants to get done.
- Example: if the user types 'teh', there is a very high chance that they actually meant 'the'.
- Use automation with care, but use it where it is beneficial. Make sure it's easy to undo!

## **Days are Seconds**

- It takes days to design a small fragment or aspect of complex software.
- <sup>o</sup> The user experiences this in a matter of **seconds**.
  - » Watch out for things that the user is supposed to work out within seconds that took the designer/programmer days and days to think about and create!

#### **Months are Minutes**

- Software packages can take **months** or years from initial conception to shipping the final bits.
- During this time you will have learned a lot about your own program, how it works and the underlying principles.
- <sup>o</sup> For **you** the learning curve of various aspects of your program may not appear to be very steep.
- The **user** has to figure all of this out within the first few minutes of using the software.

### **Features and Complications**

Try to notice if you are adding complications or removing complications.

### **Seconds are Hours**

- Your program needs to be responsive. Boredom sets in immediately when nothing happens.
- <sup>o</sup> Always respond immediately to a user request.
- Break up long operations, do them in the background etc.
- Collect long operations into one really long one.

# Six steps to good software design

- 1. Invent some users.
- 2. Figure out the important activities.
- 3. Figure out the user model-how will each of your. imaginary users expect to accomplish those activities?
- 4. Sketch out the first draft of the design.
- 5. Iterate over your design again and again, making it easier and easier until it's well within the capabilities of your imaginary users.
- 6. Watch real humans trying to use your software. Note the areas where people have trouble, which are probably areas where the program model isn't matching the user model.

### **Explorable Systems**

# **Explorable System: Visibility**

 In each state of the system, the user must readily see and be able to do the allowable actions. The visibility acts as a suggestion, reminding the user of possibilities and inviting the exploration of new ideas and methods.

### **Explorable Systems: Feedback**

 The effect of each action must be both visible and easy to interpret. This allows users to learn the effects of each action and to develop a good mental model of the system.

## **Explorable Systems: No Risk**

 Actions should be without cost. When an action has an undesirable result, it must be readily reversible.

## **Usability Tests**

# **Usability tests**

- You don't need to test with a lot of users.
- You don't really care about statistics the purpose of usability testing is to find flaws in your design.
- Five to six users is all that's required.
- We are digging for truffles here! Three or four pigs in a forest will most likely find the same number of truffles as will 1,000 of them.
- Usability tests are a measure of learnability, not usability.

# **Programmers play, while others...**

- Programmers tend to download and play with a lot of software and are not afraid of it.
- Ordinary people tend to not do something until they think they understand it fully.
- Programmers are born without a lot of sympathy for how much trouble ordinary people have using software.
- Programmers can keep 19 things in their short-term memory; normal people can keep five. Programmers are exceedingly rational and logical, to the point of exasperation; normal people are emotional and say things like "My computer hates me".

### The Best Reason for Usability Tests:

- **YOU**
- They are a great way to educate programmers about the real world. During these tests, the programmer will experience some reality about the real-world humans and how they use their product.

### The End

# **Putting the User in Charge**

- No interface
- Command line
- Questions and Answer
- Menus

## **Seven stages of action**

Forming the goal	Goal
Forming the intention	Execution
Specifying the action	
Executing the action	
Perceiving the state of the world	Evaluation
Interpreting the state of the world	
Evaluating the outcome	