Thinking about Notational Systems

This questionnaire collects your views about how easy it is to use some kind of notational system. Our definition of “notational systems” includes many different ways of storing and using information – books, different ways of using pencil and paper, libraries or filing systems, software programs, computers, and smaller electronic devices. The questionnaire includes a series of questions that encourage you to think about the ways you need to use one particular notational system, and whether it helps you to do the things you need.

Section 1 - Background information:

What is the name of the system?	
How long have you been using it?	
Do you consider yourself proficient in its use?	
Have you used other similar systems? (If so, please name them)	

Section 2 - Definitions:

You might need to think carefully to answer the questions in the next sections, so we have provided some definitions and an example to get you started:

Product

The product is the ultimate reason why you are using the notational system – what things happen as an end result, or what things will be produced as a result of using the notational system. This event or object is called the product. Any product that needs a notation to describe it usually has some complex structure.

Notation

The notation is how you communicate with the system – you provide information in some special format to describe the end result that you want, and the notation provides information that you can read. Notations have a structure that corresponds in some way to the structure of the product they describe. They also have parts (components, aspects etc.) that correspond in some way to parts of the product.

Notations can include text, pictures, diagrams, tables, special symbols or various combinations of these. Some systems include multiple notations. These might be quite similar to each other – for example when using a typewriter, the text that it produces is just letters and characters, while the notation on the keys that you press tells you exactly how to get the result you want. In other cases, a system might include some notations that are hard for humans to produce or to read. For example when you use a telephone the notation on the buttons is a simple arrangement of digits, but the noises you hear aren’t so easy to interpret (different dialling tones for each number, clicks, and ringing tones). A

telephone with a display therefore provides a further notation that is easier for the human user to understand.

Sub-devices

Complex systems can include several specialised notations to help with a specific part of the job. Some of these might not normally be considered to be part of the system, for example when you stick a Post-It note on your computer screen to remind you what to write in a word processor document.

There are two kinds of these sub-devices.

- The Post-It note is an example of a helper device. Another example is when you make notes of telephone numbers on the back of an envelope: the complete system is the telephone plus the paper notes – if you didn't have some kind of helper device like the envelope, the telephone would be much less useful.
- A redefinition device changes the main notation in some way – such as defining a keyboard shortcut, a quick-dial code on a telephone, or a macro function. The redefinition device allows you to define these shortcuts, redefine them, delete them and so on.

Note that both helper devices and redefinition devices need their own notations that are separate from the main notation of the system. We therefore ask you to consider them separately in the rest of this questionnaire.

To review how we intend to use these terms, consider the example of typing business letters on a word processor. The product of using the word processor is the printed letter on paper. The notation is the way that the letter looks on the screen – on modern word processors it looks pretty similar to what gets printed out, but this wasn't always the case. If you want to find and replace a particular word throughout a document, you can call up a helper device, the search and replace function, usually with its own window. This window has its own special notation – the way that you have to write the text to be found and replaced, as well as buttons that you can click on to find whole words, or to find the word in upper and lower case etc.

Section 3 – Parts of your system:

What task or activity do you use the system for?	
What is the product of using the system?	
What is the main notation of the system?	

.When using the system, what proportion of your time (as a rough percentage) do you spend:	
Searching for information within the notation	%
Translating substantial amounts of information from some other source into the system	%
Adding small bits of information to a description that you have previously created	%
Reorganising and restructuring descriptions that you have previously created	%
Playing around with new ideas in the notation, without being sure what will result	%

Are there any helper devices? Please list them here, and fill out a separate copy of section 5 for each one.	
Are there any redefinition devices? Please list them here, and fill out a separate copy of section 5 for each one.	

Section 4 – Questions about the main notation:

~~Visibility and Juxtaposability: ability to view components easily~~
~~Systems that bury information in encapsulations reduce visibility. Because examples are important for problem solving, such systems are to be deprecated for exploratory activities; likewise, if consistency of transcription is to be maintained, high visibility may be needed.~~

- ~~How easy is it to see or find the various parts of the notation while it is being created or changed? Why?~~
- ~~What kind of things are more difficult to see or find?~~
- ~~If you need to compare or combine different parts, can you see them at the same time? If not why not?~~

Viscosity: resistance to change
API Viscosity
A viscous system needs many user actions to accomplish one goal. Changing all headings to uppercase may need one action per heading. (Environments containing suitable abstractions can reduce viscosity.) We distinguish repetition viscosity, many actions of the same type, from knock-on (or "domino") viscosity, where further actions are required to restore consistency.
What are the barriers to change inherent in the API, and how much effort does a targeted developer need to expend to make a change?

- When you need to make changes to previous work, how easy is it to make the change? Why?
- Are there particular changes that are more difficult or especially difficult to make? Which ones?

~~Diffuseness: verbosity of language~~ *(somehow replaced by work-step-unit)*
~~Some notations can be annoyingly long-winded, or they can occupy too much valuable "real-estate" within a display area. Big icons and long words reduce the available working area.~~

- ~~Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?~~
- ~~What sorts of things take more space to describe?~~

~~Hard Mental Operations: high demand on cognitive resources~~
~~A notation can make things complex or difficult to work out in your head, by making inordinate demands on working memory or by requiring deeply nested goal structures.~~ *(somehow replaced by working framework)*

- ~~What kind of things require the most mental effort with this notation?~~
- ~~Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?~~

~~Error-Proneness:the notation invites mistakes and the system gives little protection~~
~~Enough is known about the cognitive psychology of slips and errors to predict that certain notations will invite them. Preventative mechanisms (e.g., check digits, enforced declaration of identifiers, etc.) can redeem the problem.~~

- ~~Do some kinds of mistake seem particularly common or easy to make? Which ones?~~
- ~~Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?~~

Closeness of Mapping: closeness of representation to domain
Domain Correspondence
How closely related is the notation to the result it is describing?
How clearly do the API components map to the domain?
Are there any special tricks?

- How closely related is the notation to the result that you are describing? Why? (Note that in a sub-device, the result may be part of another notation, rather than the end product).
- Which parts seem to be a particularly strange way of doing or describing something?

Role-Expressiveness: the purpose of an entity is readily inferred
Role Expressiveness
Role-expressive notations make it easy to discover why the author has built the structure in a particular way; in other notations each entity looks much the same, and discovering their relationships is difficult. Assessing role-expressiveness requires a reasonable conjecture about cognitive representations.
How apparent is the relationship between each component and the program as a whole?

- When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?
- Are there some parts that are particularly difficult to interpret? Which ones?
- Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?
- **When reading code that uses the API, is it easy to tell what each section of code does? Why?**
- **Are there some parts that are particularly difficult to interpret? Which ones?**
- **When using the API, is it easy to know what classes and methods of the API to use when writing code?**

~~Hidden Dependencies: important links between entities are not visible~~
~~If one entity cites another entity, which in turn cites a third, changing the value of the third entity may have unexpected repercussions. Examples: cells of spreadsheets; style definitions in Word; complex class hierarchies; HTML links. Sometimes actions cause dependencies to get frozen for example, soft-figure numbering can be frozen when changing platforms. These interactions with changes over time are still problematic in the framework.~~

- ~~If the structure of the product means some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?~~
- ~~In what ways can it get worse when you are creating a particularly large description?~~
- ~~Do these dependencies stay the same, or are there some actions that cause them to get frozen? If so, what are they?~~

Progressive Evaluation: work-to-date can be checked at any time

Progressive Evaluation

Evaluation is an important part of a design process, and notational systems can facilitate evaluation by allowing users to stop in the middle to check work so far, find out how much progress has been made, or check what stage in the work they are up to. A major advantage of interpreted programming environments such as Basic is that users can try out partially completed versions of the product pro- gram, perhaps leaving type information or declarations incomplete.

To what extent can partially completed code be executed to obtain feedback on code behavior?

- How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?
- Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?
- Can you try out partially-completed versions of the product? If not, why not?
- **How easy is it to stop in the middle of the scenario and check the progress of work so far?**
- **Is it possible to find out how much progress has been made? If not, why not?**

~~Provisionality: degree of commitment to actions or marks~~

~~Even if there are hard constraints on the order of doing things (premature commitment), it can be useful to make provisional actions such as recording potential design options, sketching, or playing "what-if" games. Not all notational systems allow users to fool around or make sketchy markings.~~

- ~~Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? What features of the notation help you to do this?~~
- ~~What sort of things can you do when you don't want to be too precise about the exact result you are trying to get?~~

Premature Commitment: constraints on the order of doing things

Premature Commitment

Self-explanatory. Examples: being forced to declare identifiers too soon; choosing a search path down a decision tree; having to select your cutlery before you choose your food.

To what extent does a developer have to make decisions before all the needed information is available?

- When you are working with the notation, can you go about the job in any order you like, or does the system force you to think ahead and make certain decisions first?
- If so, what decisions do you need to make in advance? What sort of problems can this cause in your work?

Consistency:similar semantics are expressed in similar syntactic forms

Consistency

Users often infer the structure of information artifacts from patterns in nota- tion. If similar information is obscured by presenting it in different ways, usabil- ity is compromised.

Once part of an API is learned, how much of the rest of it can be inferred?

- Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear? Please give examples.
- Are there places where some things ought to be similar, but the notation makes them different? What are they?

~~Secondary Notation: extra information in means other than formal syntax~~
~~Users often need to record things that have not been anticipated by the notation designer. Rather than anticipating every possible user requirement, many systems support secondary notations that can be used however the user likes. One example is comments in a programming language; another is the use of colors or format choices to indicate information additional to the content of text.~~

- ~~Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?~~
- ~~If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?~~
- ~~Do you ever add extra marks (or colours or format choices) to clarify, emphasise or repeat what is there already? [If yes: does this constitute a helper device? If so, please fill in one of the section 5 sheets describing it]~~

Abstraction (Management): types and availability of abstraction mechanisms
Abstraction Level
Abstractions (redefinitions) change the underlying notation. Macros, data structures, global find-and-replace commands, quick-dial telephone codes, and word-processor styles are all abstractions. Some are persistent, some are transient. Abstractions, if the user is allowed to modify them, always require an abstraction management redefinition subdevice. It will sometimes have its own notation and environment (e.g., the Word style-sheet manager) but not always (e.g., a class hierarchy can be built in a conventional text editor). Systems that allow many abstractions are potentially difficult to learn.
What are the minimum and maximum levels of abstraction exposed by the API, and what are the minimum and maximum levels usable by a targeted developer?

- Does the system give you any way of defining new facilities or terms within the notation, so that you can extend it to describe new things or to express your ideas more clearly or succinctly? What are they?
- Does the system insist that you start by defining new terms before you can do anything else? What sort of things?
- If you wrote here, you have a redefinition device: please fill in one of the section 5 sheets describing it.

NEW / Learning Style
What are the learning requirements posed by the API, and what are the learning styles available to a targeted developer?

- ...

Hard Mental Operations: high demand on cognitive resources / **Working Framework**
What is the size of the conceptual chunk needed to work effectively?

- What kind of things require the most mental effort with this notation?
- Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?

Diffuseness: verbosity of language / **Work-Step Unit**
How much of a programming task must/can be completed in a single step?

- Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?
- What sorts of things take more space to describe?

- **Does the amount of code required for this scenario seem just about right, too much, or too little? Why?**
- **Does the amount of code required for each subtask in this scenario seem just about right, too much, or too little? Why?**

NEW / Penetrability
How does the API facilitate exploration, analysis, and understanding of its components, and how does a targeted developer go about retrieving what is needed?

- ...

NEW / API Elaboration
To what extent must the API be adapted to meet the needs of a targeted developer?

- ...

[NOVL]

- Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

[IMPR]

- After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they?
- Could it be improved specifically for your own requirements?

Section 5 – Questions about sub-devices:

Please fill out a copy of this page for each sub-device in the system.
This page is describing (tick one box):

a helper device □,

or a redefinition device □

- What is its name?
- What kind of notation is used in this sub-device?

When using this sub-device, what proportion of the time using it (as a rough percentage) do you spend:	
Searching for information	%
Translating substantial amounts of information from some other source into the system	%
Adding small bits of information to a description that you have previously created	%
Reorganising and restructuring descriptions that you have previously created	%
Playing around with new ideas in the notation, without being sure what will result	%

In what ways is the notation in this sub-device different from the main notation?
Please tick boxes where there are differences, and write a few words explaining the difference.

Visibility and Juxtaposability Is it easy to see different parts?		
Viscosity Is it easy to make changes?		
Diffuseness Is the notation succinct or long-winded?		
Hard Mental Operations Do some things require hard mental effort?		
Error Proneness Is it easy to make errors or slips?		
Closeness of Mapping Is the notation closely related to the result?		
Role-Expressiveness Is it easy to tell what each part is for?		
Hidden Dependencies Are dependencies visible?		
Progressive Evaluation Is it easy to stop and check your work so far?		
Provisionality Is it possible to sketch things out?		

Premature Commitment Can you work in any order you like?		
Consistency Are any similarities between different parts clear?		
Secondary Notation Can you make informal notes to yourself?		
Abstraction Management Can you define new terms or features?		
[NOVL] Do you use this notation in unusual ways?		
[IMPR] How could the design of the system be improved?		