

Should we Lower the Need for Tailoring or Increase the Tailoring Space? A Design Approach to EUD.

Catherine Letondal
letondal@pasteur.fr

Institut Pasteur
Pôle Informatique
28, rue du Dr. Roux
75724 - Paris Cedex 15, France

September 20, 2003

1 Introduction

In biology research, software evolves rapidly and users' needs are highly diverse. Facilitating this type of software evolution in context requires two primary considerations: the technical issues involved in the layered opening of the tailoring space to the user at run-time, and the methodological issues based on user-centered design studies. This paper describes the latter: our design approach and how it helped to build a flexible environment for biological data analyses.

2 Context

The aim of my work was to build an environment for biological data analyses, *biok* [LZ03] that explicitly accounted for problems I observed through interviews with biologists. As of today, biologists already have plenty of tools, including several hundreds of Web tools, to analyze their data. However, users still have difficulties conducting their analyses. Typical problems, such as those described in [CFL⁺03] involve a combination of heterogeneous tools, requiring the development of simple scripts that perform some parsing and glueing. In addition users often need to apply a simple operation that is not provided by the tool. For example, they may search a DNA sequence for characters other than A, C, T and G. They may also be interested in slight variations in how a computation is specified, such as searching for a common pattern in a set of sequences but excluding those that are repeated several times within a given sequence.

3 Approach

As [Eis95] notes, developing new tools with an extended set of functionalities is just a short-term solution, since new needs show up all the time, and generally lead to even more complex tools. In order to address this type of problem, I chose to investigate general software flexibility principles, adapted for end-user programming within the user interface. These principles, described in [LZ03], rely on a full reflexive architecture based on XOTcl dynamic introspection features. The main goal of our technical choices are *to open the tailoring space, or meta-level space, as much as possible*, while helping the user to understand and manipulate the code.

4 Tailoring as repair

I believe that opening the tailoring space, even though a *general technical solution*, does not help if used alone. It is commonly acknowledged that the design of a tool, i.e. not only how it is structured but how it is designed, and how the design process has been conducted, plays an important role in its *actual and potential adaptation* to the user needs. Central features and tailoring are dual concerns with respect to design: the more a tool is adapted to the user needs, the less tailoring it intrinsically needs. Likewise, there is a concern that tailoring techniques could be offered to users in quickly developed commercial software to overcome flaws in the tool, i.e. "this tool is not perfect, but feel free to tailor it".

I videotaped biologists using existing tools and I observed which main functions they sought when visualising their data. Such observations brought information on what *not* to leave to tailoring. I also organized several workshops where biologists brainstormed features needed

in an alignment editor, and built prototypes for some of these features. From these observations and workshops, I got direct requirements for the tool, and, as described in the next section, a better understanding of the tailoring space.

5 Design for flexibility

Designing a tailorable system requires finding potential dimensions for evolution and a sound meta-technique design [SKW97].

5.1 Finding potential dimensions for evolution

The potential dimensions along which base-level features can evolve must be investigated from the beginning of the development process on. Interviews help the designer to acquire knowledge about various concrete situations of use. Likewise, brainstorming and future workshops create a design space along which design choices can be made and potential variations can be detected. As Trigg [Tri92] and Kjaer [KM95] suggest, participatory design helps to both identify areas in the system that can be considered as stable, and areas that show the potential for variation. Stable parts require functions that must be directly available, without any programming, whereas variable parts must be available to tailoring.

I created an alignment visualization tool in *biok* that vertically displays corresponding letters in multiple related sequences. During interviews, I observed how biologists actually use this type of tool. (I conducted a software review that describes more than 40 of them [Let01].) Biologists often rejected such tools because they lacked some critical feature, often a basic one such as the choice of a 'gap' character. The tools were also rejected because they were not flexible enough. Biologists preferred to use a spreadsheet or a text processor to manually fix an alignment, to add styles to highlight some parts of the alignment, or to put free text within columns. Observing users working on this type of flexible support provided me with information on the potential dimensions where flexibility might be needed.

5.2 Design of meta-techniques

A sound design, based on scenarios and workshops is also required for meta-level features.

5.2.1 End-user Programming Scenarios

Programming scenarios emerged as side-problems in several occasions. In these situations, the goal was not for the participants to describe a programming activity, but

it was possible, as a designer, to make an analogy between the task, or the way to perform it, and programming techniques.

- *Programming with examples*: during the workshop addressing the definition of tags, a participant suggested a system that is able to learn a new tag from examples; a system able to infer regular expressions from a set of sequences has also been proposed, leading to something similar to SWYN [Bla00].
- *Scripting*: annotations, e.g text associated to data, are sometimes "to do" lists that could be managed as small scripts associated with the data.
- *Command history*: a brainstorming focusing on data versioning addressed the complementary idea of command history.

5.2.2 Design strategies for programming features

A prototyping workshop can help to design programming tools, either by finding the interaction techniques that are appropriate to trigger a programming action, or to determine the complexity level for a programming tool. In a prototyping workshop centered on sequence alignment editors, one group built a mockup for pattern searching, with a syntax allowing for constraints and a given number of errors. In a subsequent workshop around a first version of *biok*, biologists focused on the definition of new tags. The tool provided, which required code editing, proved too difficult to understand. After a long brainstorming session, one of the participant built a prototype by using transparencies on the overhead projector. Shortly after I used a storyboard-mockup, composed of several A3 paper sheets with screenshots, to play the tag creation scenario in front of a few biologists. They interrupted from time to time to sketch various solutions. More generally, the tag editor in *biok* would not have been developed as it is, without addressing the following issues: Must programming be available in a special editor? Must it require a simplified programming interface? Should the user interface be interactive? Should it be accessible via graphical user interface menus? As [PRM01] or [dCdS03] suggest, a participatory approach is also helpful when designing the syntax of the language.

An important aspect that came from interviews and workshops was the *granularity* of code modification. According to our observations, biologists more often need new methods than new classes. New types of objects are not that common in bioinformatics, and defining a new class is a modelling activity that requires some experience. Moreover, user modifications at the method level in *biok* do not require sub-classing. Overloading occurs at system load time: user-edited methods are performed within the current method's class and are saved in directories that

are loaded after the system. The visualization functions through tags, however, really required for the user to be able to create new classes, which is why the mechanism is provided in the user interface.

Another workshop focused on how to specify the user’s interactions with an algorithm, letting the user provide input as the variant of the algorithm’s heuristic [[LZ03]]. Participants first specified the actual problem and then developed a paper prototype. This led us to find the most appropriate context and the interaction, i.e. how the user could trigger the algorithmic variant. For example, the user could manually highlight and thus specify an alternative within a previously-displayed result.

6 Setting a design context for tailoring situations

Interestingly, few prototyping workshops focused on programming, per se. In an “ideal” workshop, we envisioned that biologists would invent unanticipated programming constructs or new kinds of expressions for existing constructs. Instead, we found that users are not primarily interested in programming features. (See our rationale in section 4.) In fact, the use of programming during workshops is questioned by some researchers [BG91], because a modification takes too much time and causes inappropriate interruptions, or, when performed by a skilled participant, excludes non-programmers from the design process. Such situations happened when bioinformaticians, who were skilled in computing, wanted to apply their knowledge. One of them, instead of building a mockup focused on her scenario, built the whole GUI for the application. Another one built a dataflow diagram for a command history instead of the paper-based prototype to simulate what happens in the user interface.

How then can participatory design workshops be useful for in the context of EUD?

6.1 Participatory design

Participatory design is known to be useful to design strategies addressing typical breakdown situations: in a workshop, the users can design workarounds when faced with a problem. My interview observations showed that most programming situations correspond to breakdowns. In such cases, programming, rather than being the direct object of interest, emerged as the only remaining means of fixing a problem. This corresponds to a distant, reflexive and detached “mode”, similar to what is described in [Win95] or [Gre93].

Although end-user programming tools seek to blur the border between use and programming [DLL03], programming is in fact *not* in the continuity of using, but instead

in a distance. This is why a particular event or unusual situations may trigger a similar switch to programming or tailoring. I refer here to [Mac91] showing in her study about triggers and barriers to customize software, that people often wait for unusual events to take the time for this activity. The design of end-user programming environments should take into account the disruptive aspect of tailoring situations by enabling programming *in context*, as we did in *biok*.

6.2 Reflexive distance and scientific users

It should be noted however that reflexive breakdown corresponds to a usual situation in the context of scientific users. Reflection or rational distance indeed *belong to normal scientific activity*. For example, biologists often experiment in order to find the appropriate tool to get interesting results. They usually describe such explorations, at least partially, in their laboratory notebooks. A tool supporting exploratory search and the *reflection* process that occurs afterwards would be useful. Biologists could reuse their most successful steps, provide comments on the appropriate context of using particular tools, or evaluate the quality of specific results. Biologist could record not only *what* particular actions produced which results but also *why* these actions were chosen.

7 Conclusion

This paper describes a design approach, illustrated in figure 1, that both seeks to reduce the need for tailoring through user-centered methods at design time and to increase the scope of the tailoring space at use time through a reflexive software architecture.

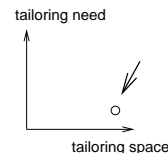


Figure 1: Tailoring need and space.

Three main reasons for rooting EUD tools and EUD research in a design process were highlighted:

1. A sound and user-centred design should lower the need for tailoring: no tailoring should be necessary for central and critical features of a tool.
2. Meta-techniques development also needs a user-centred design process, not only usability tests.

3. By highlighting tailoring as a breakdown situation and not as a feature, participatory design practice and theory provide a conceptual frame for EUD.

References

- [BG91] S. Bodker and K. Gronbaek. Design in action: From prototyping by demonstration to cooperative prototyping. In *Design at Work: Cooperative Design of Computer Systems, Chapter 11.*, pages 197–218. Hillsdale, New Jersey Lawrence Erlbaum Associates, 1991.
- [Bla00] Alan Blackwell. Swyn: A visual representation for regular expressions. In *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2000.
- [CFL+03] M.F. Costabile, D. Fogli, C. Letondal, P. Musio, and A. Piccinno. Domain-expert users and their needs of software development. In *In Proceedings of the HCI 2003 End User Development Session*, 2003.
- [dCdS03] Ceclia Kremer Vieira da Cunha and Clarisse Sieckenius de Souza. Toward a culture of end-user programming understanding communication about extending applications. In *Proceedings of the CHI'03 Workshop on End-User Development*, April 2003.
- [DLL03] Yvonne Dittrich, Lars Lundberg, and Olle Lindeberg. End user development by tailoring. blurring the border between use and development. In *Proceedings of the CHI'03 Workshop on End-User Development*, April 2003.
- [Eis95] Michael Eisenberg. Programmable applications: Interpreter meets interface. *ACM SIGCHI Bulletin*, 27(2):68–93, April 1995.
- [Gre93] J. Greenbaum. PD, a personal statement. *CACM*, 36(6):47, June 1993.
- [KM95] A. Kjaer and K.H Madsen. Participatory analysis of flexibility. *CACM*, 38(5):53–60, May 1995.
- [Let01] Catherine Letondal. Software review: alignment edition, visualization and presentation. Technical report, Pasteur Institute, Paris, France, may 2001. <http://bioweb.pasteur.fr/cgi-bin/seqanal/review-edital.pl>.
- [LZ03] C. Letondal and Uwe Zdun. Anticipating scientific software evolution as a combined technological and design approach. In *USE2003: Second International Workshop on Unanticipated Software Evolution*, 2003.
- [Mac91] Wendy E. Mackay. Triggers and barriers to customizing software. In *Proceedings of ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 153–160. ACM Press, 1991.
- [PRM01] J.F. Pane, C.A. Ratanamahatana, and Brad Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, February 2001.
- [SKW97] Oliver Stiemerling, Helge Kahler, and Volker Wulf. How to make software softer - designing tailorable applications. In *In Proc. DIS'97 (Amsterdam)*, pages 365–376, 1997.
- [Tri92] Randall H. Trigg. Participatory design meets the mop: Informing the design of tailorable computer systems. In *Proceedings of the 15th IRIS (Information systems Research seminar In Scandinavia) Gro Bjercknes, Tone Bratteteig, Karlheinz Kautz (eds.), August 1992, Larkollen, Norway.*, 1992.
- [Win95] Terry Winograd. From programming environments to environments for designing. *CACM*, 38(6):65 – 74, June 1995.