

Participatory Programming and the Scope of Mutual Responsibility: Balancing Scientific, Design and Software Commitment

Catherine Letondal
Institut Pasteur
28 rue du Docteur Roux
75015 Paris, FRANCE
letondal@pasteur.fr

Wendy E. Mackay
INRIA Bâtiment 490
Universit de Paris-Sud
91405 Orsay, FRANCE
mackay@lri.fr

ABSTRACT

Over the past seven years, we have been conducting a variety of participatory design activities with research biologists, programmers, and bioinformaticians at the Institut Pasteur in Paris. We first describe the history of these activities and how they have created the beginnings of a participatory design culture. We introduce participatory programming, which integrates participatory design and end-user programming, and examine how it acts as a medium for forging scientific ideas. Finally, we reflect on three poles of activity: the computational medium, scientific hypotheses and participatory design.

Categories and Subject Descriptors

H.1.2 [User/Machine Systems]: Human factors

H.5.2 [User Interfaces]: Evaluation/methodology, Theory & methods, Prototyping, User-centred design

General Terms

Design, Economics, Human Factors, Software Psychology.

Keywords

Participatory design, participatory programming, tailoring, end-user programming, responsibility, software evolution, software commitment, co-adaptation

1. INTRODUCTION

In many participatory design contexts, users are not expected to program: the lines of responsibility between programmer and user are clearly drawn and users are expected to use the tools that the programmers produce. In other contexts, software is explicitly designed to facilitate tailoring by users to meet individual needs [27, 34]. These users stop short of actually programming, due to lack of technical expertise or the

complexity and scope of the programming tasks. Fischer's [11]

MetaDesign approach attempts to empower users by enabling them to act as designers at use-time, using domain-oriented rather than full programming languages.

However, some users must modify their software tools, even if they are not trained as programmers. For example, Eisenberg's programmable applications [9] allow users to add features to a drawing tool by programming in Scheme. Morch [28] identifies three levels of end-user tailoring: customization, integration and extension, which includes both programming and design rationale.

We work primarily with biologists at the Institut Pasteur, most of whom operate in a highly-competitive, computer-intensive environment. They are active users of a variety of scientific software tools, ranging from algorithms for experimental data analysis to on-line search engines for enormous gene databases. Their needs are highly specific and they must either adapt or create software that allows them to address individual biological research questions. In this article, we refer to *biologists* as primarily *users* of software tools, not programmers, and contrast them with *bioinformaticians* who have formal training in both biology and computer science. The Institut Pasteur also has computer scientists, not trained in biology, who design and distribute general-purpose tools to support biologists, as well as performing their own research in bioinformatics and even human-computer interaction.

This paper describes our introduction of participatory design at the Institut Pasteur over the past seven years, involving all three groups. We focus on the collaborative development of tools such as *biok*, which explicitly support end-user programming. This has led us to define *participatory programming* as a logical extension of participatory design, in which users participate in the creation of software tools they can ultimately tailor and program themselves [14]. Here, the division of labour between programmer and user becomes blurred and issues of tailoring, end-user programming and participatory design interact in complex ways.

We begin with a brief description of our teaching and participatory design activities at the Institut Pasteur and describe how participatory programming has evolved. Next, we describe the reflections of participants on their participatory design and programming work. Finally, we discuss the tensions among

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Proceedings Participatory Design Conference 2004, Toronto, Canada.

Copyright 2004 ACM 1-58113-851-2/04/07...\$5.00.

users from different disciplines, and how the scope of mutual responsibility varies so as to balance scientific, design and software commitments.

2. EMPIRICAL STUDY

Our analysis of the scope of responsibility occurs in the context of several research and development projects each involving participatory design activities. We begin each project by visiting the biologists in their laboratories and interviewing them or videotaping them at work. We then hold discussions, brainstorming sessions and design workshops and invite biologists, bioinformaticians and computer scientists. Some meetings focus on a particular issue, others are general discussions. Participants usually bring examples of their work and we create paper mock-ups to illustrate and work through design ideas. We also conduct video brainstorming sessions [23], in which participants interact with their prototypes while being videotaped.

2.1 Software Development Research Projects

The most extensive project has been the on-going development of *biok* [18], a prototype environment with domain-oriented specialized features for dealing with bioinformatic data. *Biok* enables biologists to manipulate, for example, DNA strings and protein sequences, and visualize their features. This tool is designed to be programmable by biologists, providing meta-application mechanisms for radical tailoring [21, 28].

Another multi-year project has been the development of augmented laboratory notebooks, which link on-line functionality to paper notebooks. The project has involved biologists, archivists and managers in the design, development and testing of four prototypes, each based on different underlying technologies [26]. Although not explicitly programmable in the same way as *biok*, a key goal has been to enable biologists to control the relationship between their annotations on paper and the functionality offered by the system, so that they can easily appropriate and modify it over time, the principle of co-adaptation [24].

We have also developed several smaller, more focused projects, involving collaborations among ourselves, bioinformaticians from other laboratories, bioinformatics students and biologists. One project organized three participatory design workshops over two years to inform the development of a graphical widget enabling the 3D visualization of molecules [37]. A more recent project, called *Mobyle*, involved 20 interviews of biologists in their labs, as well as a large video brainstorming workshop (with 24 participants) and two smaller prototyping workshops [20].

Finally, we run an annual intensive four-month course for research biologists to teach them aspects of computing. During this course, computer scientists and bioinformaticians from the Information Technology (IT) department, as well as visiting professors, cover programming techniques, theory, e.g., algorithm development, logic, problem modelling and design methods, and technical applications, e.g., databases and web technologies, relevant for biologists. What is perhaps more unusual is that we also explicitly teach participatory design techniques as part of the course [22], following Mackay's video tutorial [25]. For the past two years, students have used these

techniques to design and develop a usable system as a class project. An interesting side-effect has been the exposure of the computer science teachers to participatory design techniques. Most are now aware of and accept the approach and some have even asked for help to run design workshops for their student's projects.

2.2 Research Setting

The Institut Pasteur, in Paris, France, is a non-profit private foundation "dedicated to the prevention and treatment of diseases through biological research, education and public health activities". Founded by Louis Pasteur in the 1880s, the Institut Pasteur is a world-renowned research laboratory, attracting researchers from approximately 60 countries. It is also a teaching institution, with 200+ doctoral students and visiting researchers who participate in 15 diploma courses.

The IT department installs, develops and maintains scientific software for biologists, including support for scientific databases and network infrastructure. It also runs continuing-education courses on scientific software and programming for the biologists. In addition, some members of the IT department pursue their own research in bioinformatics and even human-computer interaction. From the biologists' perspective, the IT department is primarily a service for dealing with local technical problems and email accounts. However, some biologists also collaborate on specific projects. Because the centre does not have sufficient resources to provide "programmers-for-hire", the IT department seeks to create general-purpose tools that support multiple biologists, valuing reuse and accessibility over individual solutions to particular problems.

2.3 Participants: Differing Perspectives

In this paper, we identify three categories of participants in our participatory design activities. The largest group are research *biologists*, "end-users" with few or no programming skills. Although most take introductory computer science courses and are, in principle, capable of programming, the majority avoid it. Unlike computer scientists or bioinformaticians, their primary goal is to solve a biology problem; they view software as a means to an end, not an end in itself. We initially assumed that biologists would value creating flexible tools, if only because they might need to reuse them themselves, but this was rarely the case. In one interview, for example, a biologist expressed both surprise and amusement that his colleague, a bioinformatician, was strongly motivated to make his software usable by others. In another group discussion, several biologists expressed frustration with the IT centre's reluctance to write individual program solutions for them, finding odd the focus on creating general tools to be used by multiple biologists.

At the other extreme are computer scientists, who have little or no training in biology, but are charged with the problem of creating software tools for them. Unlike biologists, the object of their work is the software they create. Their training leads them to value software flexibility and the creation of generic tools that perform a variety of functions for diverse people with different needs.

Bioinformaticians bridge the gap, with training in both computer science and biology. We see two types of bioinformaticians:

those who maintain a strong interest in biology and seek to create tools that both they and other biologists in the same research area can use, and those who are 'seduced' by the computer and turn their attention almost exclusively to creating biological software tools. Bioinformaticians have an advantage over computer scientists in this domain, because they deeply understand the software problem from the user's perspective. On the other hand, they sometimes define the problem very narrowly, with a personal perspective, making their tools less generally useful.

Bioinformaticians are distinguished from biologists by their interests and behaviour: Biologists focus on biology and publish biology research articles whereas bioinformaticians are interested in both biology questions and the design of tools to address them. They publish biology articles but also create more general-purpose software tools, which they distribute to a wider audience. (Note that the definition of a bioinformatician is somewhat arbitrary: a small number of biologists develop an interest in programming and achieve a high level of proficiency, effectively becoming bioinformaticians.) Both computer scientists and bioinformaticians view software as a general resource and are motivated by the fact that it might be used by others, whereas biologists are primarily concerned with answering their individual biology research questions.

2.4 Participatory Design Activities

Our participatory design activities include a variety of techniques for discovering biologists' needs, ranging from a large survey to individual observation and interviews. We also regularly conduct design workshops with varying specific activities according to their needs at the time.

2.4.1 Campus-wide Survey

In 1996, we conducted a campus-wide survey with 40 questions grouped into categories: computing education, software and network use, access to technical documentation and types of technical problems encountered [16]. Figure 1 shows the main groups that we identified through the analysis of the survey data (about 600 answers) plotted along two dimensions: level of programming autonomy and level of use of scientific computing:

- *Occasional users* were the largest group (36%) and did not use scientific computer tools directly.
- *Non-Unix users* (15%) were mostly independent from the IT department and primarily used their own microcomputers and statistical software.
- *Young scientists* (15%) were interested in bioinformatics, and were able to program or at least build Web sites.
- *Learners* (15%) were more-established scientists who had taken a computing course recently, often from the IT department.
- *Gurus* (6%) were already heavily involved in computing and programming.

Both the computing skills and the available computer tools have changed greatly in the intervening years since this survey was taken. The Institut Pasteur has hired a significant number of bioinformaticians to act as local developers [12]. In various laboratories, young scientists (Ph.D.s and post-doctoral fellows) are now more likely to have had formal training in computing and the number of convenient software packages for biologists has increased, particularly via the Internet.



Figure 1. Campus-wide survey analysis of computing use

2.4.2 Observation and Interviews

Our own research projects have led us to observe, interview and videotape many biologists in their laboratories. We have conducted over 65 interviews over the past seven years. Approximately 30 were dedicated to the research on end-user programming [18], 20 have been recently organized for the design of Mobylye [20] and 15 for the augmented laboratory notebook research project [25]. We asked for volunteers who were willing to be interviewed in their laboratories. We sometimes asked them to arrive at a time when we could videotape them as they performed specific activities relevant to the project, such as performing certain on-line analyses or preparing their laboratory notebooks. In some cases, other members of the lab joined in to show us their work, either similar or in contrast to the original person being interviewed. We consider both in situ observation and interviews to be an on-going activity and observations from one project are often relevant to other projects.

2.4.3 Participatory Design Workshops

The above projects all use participatory design workshops to create and explore design ideas. We ask participants to bring specific examples of work artefacts, such as DNA sequences or notebook pages. We also provide participants with a variety of prototyping materials for creating mockups of proposed software interfaces. The participants then generate ideas and act out how they would like to, for example, visualize a DNA sequence or search for information in a laboratory notebook. Videotaping very short clips [23] forces participants to think concretely in terms of interaction with the software tool and provides a record that other participants can easily understand.

We sometimes replay videos from earlier workshops as a source of inspiration, and on some occasions, the videos have been used by programmers as a design specification for software that they then prototype and develop.

Many of the people who volunteered to be interviewed or observed in their labs have also participated in participatory design workshops. The largest project, *biok*, has involved a series of video brainstorming and prototyping workshops over several years. We drew prototyping themes from brainstorming sessions and from use scenarios, which based on interviews and observation. Each workshop involved from 5-30 people, with participants from the Institut Pasteur or other biological research laboratories, as well as biology researchers who were students in our programming course. The laboratory notebook project has also involved a series of workshops, spanning four years. Participants included not only biologists and programmers, but also archivists and managers, who offer a contrasting perspective on the requirements for an augmented laboratory notebook.

Our annual *Informatics in Biology* course is also a source of participatory design workshops. Last year, 9 of 15 students actively used participatory design techniques, including interviewing other biologists, organizing brainstorming sessions and running prototyping workshops with paper, postits and transparencies to simulate proposed software before implementing it. (The other students developed algorithms with little or no user interface component.)

Figure 2 shows the chronology of most of the workshops organized from 1996-2004, including the campus-wide survey and other user studies, tied to *biok's* development.

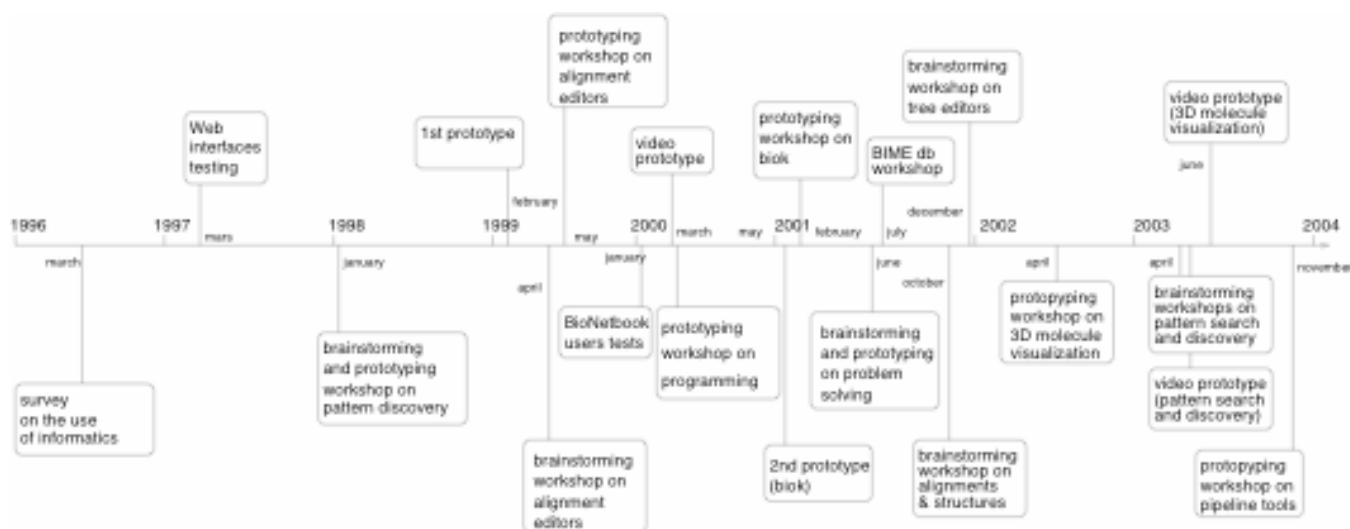


Figure 2. History of Participatory Design Workshops at the Institut Pasteur

2.5 Participatory Programming

In exploratory research domains such as biology, a key challenge for computer scientists is to develop highly-flexible tools that address variants of problems [29]. Some software tools, such as

Web interfaces that wrap local programs [17], simple data editors or large scientific databases are relatively easy to make flexible. However, building generic and flexible tools that support common tasks is extremely difficult especially considering the number of such tasks and the problem of anticipating them. This

influenced the design and implementation of *biok*, a programmable data analysis environment. The objective is to create both an open tool for tailoring, programming and design of new components as well as a directly-usable bioinformatics data analysis application.

However, simply opening up the tailoring space is not sufficient, if the tool is to be used alone. We have been experimenting with *participatory programming*, a logical extension of participatory design, in which users participate in the creation of software tools that they can ultimately tailor and program themselves. We want to reduce the need for tailoring through user-centred methods at design time. At the same time, we want to increase the scope of the tailoring space at use time through a reflexive software architecture. We believe that the design of a tool, not simply its structure but also the process by which it has been designed, plays an important role in how well it can be adapted to user needs. The choice of central features and the ability to tailor are dual concerns with respect to design: the better adapted to user needs, the less tailoring will be required. At the same time, the easier it is to tailor, the more likely it can be adapted as users' needs change. (This view contrasts with a strategy sometimes found in commercial software, in which tailoring is viewed as a means of overcoming flaws in the tool.) Our design process involved videotaping biologists as they used existing tools and we also note which functionality they requested during interviews and design workshops. These influenced which functions were included, i.e. not left to tailoring. We also organized specific workshops devoted to brainstorming ideas for new features, which provided additional requirements and also a deeper understanding of the tailoring space.

2.5.1 Design for Flexibility

Designing tailorable systems requires identifying potential dimensions for evolution and creating an interface for modifying tools, i.e. a sound meta-technique design [33].

2.5.1.1 Finding Potential Dimensions for Evolution

From the very beginning of the design process, it is important to consider the potential dimensions along which features may evolve. Interviews with users help inform concrete use scenarios, whereas brainstorming and future workshops create a design space within which design options can be explored. As Trigg [35] and Kjaer [15] suggest, participatory design helps identify which areas in a system are stable and which are suitable for variation. Stable parts require functionality to be available directly, without any programming, whereas variable parts must be subject to tailoring. For example, the visual alignment tool in *biok* vertically displays corresponding letters in multiple related sequences. Initial observations of biologists using this type of tool [19] revealed that they were rarely flexible enough: biologists preferred spreadsheets or text editors so they could manually adjust alignments, add styles and highlight specific sections. It became clear that this functionality required explicit tailoring support.

2.5.1.2 Design of Meta-techniques

Scenarios and workshops are important for designing meta-level features. Scenarios sometimes reveal programming areas as side issues. The goal is not to describe the programming activity per

se, but rather to create an analogy between the task, how to perform it, and the relevant programming techniques. We identified several types of end-user programming scenarios:

- *Programming with examples*: One participant suggested that the system learn new tags from examples. Another proposed inferring regular expressions from a set of DNA sequences. These led to a design similar to SWYN [3].
- *Scripting*: One participant said that text annotations, associated with data, can act as a “to do” list, which can be managed with small scripts associated with the data.
- *Command history*: During a brainstorming session focusing on data versioning, a participant suggested the complementary idea of command history.

Design strategies for programming features

In designing the *biok* tag editor, we had to consider 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?

We found prototyping workshops invaluable for addressing such issues: They help explore which interaction techniques best trigger programming actions and determine the level of complexity of a programming tool. Figure 3a shows one group from an alignment sequence workshop building a pattern-search mockup, including syntax for constraints and errors. Figure 3b shows a later workshop on an early version of *biok*, in which biologists were unable to understand how to edit code to define new tags. After a brainstorming session, one person built a paper-and-transparencies prototype. We created an A3-size storyboard mockup and walked through the tag creation scenario with the biologists. *Biok's* current tag editor resulted directly from these workshops.



Figure 3a. Prototyping a pattern-search



Figure 3b. An annotation scenario

Participatory approaches are also helpful when designing language syntax [30, 5] or deciding on the granularity of code modification. In object-oriented programming terms, we found that biologists are more likely to need new methods than new classes. Since defining new classes is a skilled modelling activity, we designed *biok* so that user modifications at the user level do not require subclassing. User-edited methods are performed within the current method's class and are saved in directories that are loaded after the system. However, visualizing tags required the user to create new classes, which lead us to provide this as a mechanism in the user interface.

2.5.1.3 Setting the context for tailoring situations

Despite the end-user programming aspect of *biok*, few prototyping workshops focused on programming, per se. We expected that biologists would invent unanticipated programming constructs or new expressions for existing constructs. Instead, we found that users are not primarily interested in programming features. We agree with [4] that allowing programming during workshops is a poor idea: modifications take too long and exclude participants who are not programmers from the design process. In the (rare) cases in which we tried it, one bioinformatician built a GUI for her application, while another built a dataflow diagram instead of a paper-based version of the user interface. Neither activity allowed others to participate.

Our observations of biologists showed that most programming situations correspond with breakdowns: particular events cause users to reflect on their activities and trigger a switch to programming [24]. Programming is not the focus of interest, but rather a means of fixing a problem. It is a distant, reflexive and detached "mode", as described in [38] or [13]. While end-user programming tools may seek to blur the border between use and programming [8], it is important to take this disruptive aspect into account, by enabling programming in context. Developing and playing through scenarios are particularly useful for identifying breakdowns and visualizing how users would like to work around them.

3. REFLECTING ON PARTICIPATORY PROGRAMMING

Participatory design is always greatly influenced by the context in which it occurs. We wanted to take the opportunity, after seven years, to reflect both on how participatory design with this user population has evolved and how our concept of it has changed. We decided to ask former and current participants to help us reflect on participatory design at the Institut Pasteur. We asked 11 biologists, bioinformaticians and computer scientists to reflect on their experiences. Here we focus on the effect of working across disciplines, with the corresponding tensions and shifts of responsibilities that it entails.

O'Day et al. [29] studied collaborations between biologists dealing with microarray data and computer scientists, statisticians and mathematicians. They describe how software tools play the role of boundary objects: although the actual object is the same, each discipline interprets it differently. They explain how the collaborative work, aiming at understanding the biological data, relies on the use of these boundary objects to build histories, and to make these histories converge, in order to produce a consensus hallucination. They show how important it is for both computer scientists and biologists to manage the tension that comes from the fact that both have to learn, and both are unsure of how to bring their disciplinary strengths together.

Biologists who do not program must still constantly reassess technological issues as their particular branch of biology evolves. Because programming resources are limited and communicating effectively with programmers is often difficult, they are under increasing pressure to learn more computing and take greater responsibility for the software they use. For programmers without training in biology, the situation is reversed. Although capable of implementing code, they do not necessarily understand the underlying scientific foundations. They are under increasing pressure to learn more biology and take greater responsibility for the accuracy and appropriateness of their algorithms.

Yet crossing these disciplinary boundaries is difficult. Individual biologists and programmers must position themselves along a continuum with respect to their technological skills, both in their own and in the other domain. This creates a tension between programmers and biologists: cooperation is possible, but not required, on both sides. It is tempting to require that everyone become a bioinformatician, with extensive training in both domains. However, this is very expensive and does not take optimum advantage of individuals' capabilities and interests. Also, even bioinformaticians constantly struggle to stay current with the technical advances in both disciplines.

Each participant thus has a certain level of technical skill in a particular domain and a corresponding sense of responsibility with respect to the quality of the code created. Lowered perceived responsibility leads to lowered mutual involvement. For example, if a biologist cannot decipher how a particular algorithm is implemented, she abdicates her responsibility for it, trusting that it is implemented properly. If she thinks she can understand it, she is more likely to work with a programmer to ensure that it is correct. The willingness of biologists and computer scientists to interact changes, however, when we present opportunities for *weak coupling*, such as offered by

participatory design workshops: these may result in unexpected and interesting collaborations.

The investment of participants in a participatory design context or in using participatory programming tools can be analyzed in terms of attention [1], or the time and effort spent [24]. However we can also explore anticipated social, professional and scientific outcomes with respect to their associated responsibilities. The next section describes the development of participatory design activities and how participatory programming brings an additional medium and context where convergences of stories can happen and mutual involvement and responsibility can evolve.

3.1 Three Complementary Poles

This section describes how collaborations among biologists, bioinformaticians and programmers occur, with flows of information along three poles¹: computational medium, scientific theory and participatory design activities. Figure 4 shows how roles, artefacts and ideas can be organized with respect to these three poles.

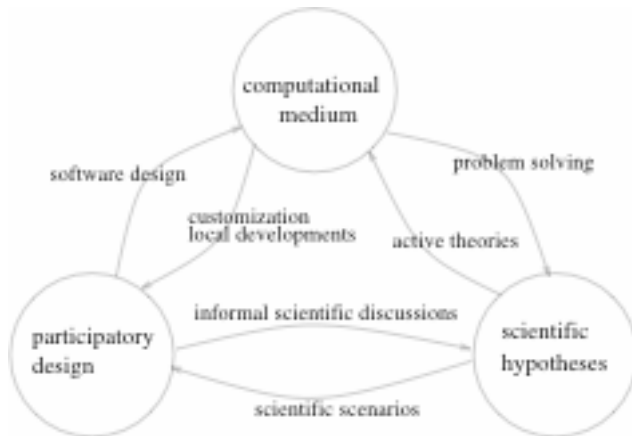


Figure 4. Participatory programming involves interaction along three poles: computational medium, scientific hypotheses and participatory design.

- *Computational medium*: encompasses any software artefact, ranging from public packages to local use-once scripts and customization files. This pole also acts a medium to enable semi-formal reasoning, programming, e.g. naming, specifying conditionals, as well as practices by biologists and bioinformaticians, including copy-pasting small chunks of code.
- *Scientific hypotheses*: includes the models, experiments and theories that form the focus of interest for the biologist. Operationalising hypotheses allows biologists to perform bench experiments. Models, which are built from software results, previous experiments, meeting discussions and bibliographic searches, may be depicted as abstractions that

play an active role in the scientific process and in the use of computing.

- *Participatory design*: the collection of activities that allow biologists and programmers to collaborate on developing biological research tools, including in situ observations, interviews and participatory design workshops.

More and more scientific models are operationalised as computer programs, effectively turning a model into what Saurin² terms *active theory*. The fields of bioinformatics and genomics are composed mostly of tasks that are defined by computer artefacts. The expression medium [7,6] is encoded as strings and problems are expressed as string operations, including comparisons, counting, word search, trees and graphs. Computer programs can actively support on-going scientific reasoning, from formally stating hypotheses to verifying hypotheses' results. Finally, programs are a persistent medium and are increasingly used as a tool for sharing ideas among biology researchers. The outcomes and activities that flow among the three poles include:

- *participatory design* → *computational medium*: The most tangible outcome of a participatory workshop is a software package, designed from requirements and prototypes gathered in interviews and design workshops.
- *computational medium* → *participatory design*: Observing actual software, programming, tailoring [36] and ad-hoc design practices [8] provides ideas and information that facilitate the on-going design activity.
- *computational medium* → *scientific hypotheses*: In a growing number of biological sub-domains, scientific hypotheses may be built from ad-hoc or opportunistic software use, combining several analysis tools.
- *scientific hypotheses* → *computational medium*: When implemented in programs, scientific models turn into active theories that can be run and experimented with.
- *participatory design* → *scientific hypotheses*: One reason cited for attending participatory design workshops is that they become a forum for discussing and sharing scientific ideas and hypotheses.
- *scientific hypotheses* → *participatory design*: Through design workshops, scientific ideas play a leading role in the artefacts that are ultimately produced; biologists often push particular hypotheses and some are able to teach them to the computer scientists participating in the workshop.

The above framework can also be applied, on a smaller scale, to our *Informatics for Biology* course. Biologists learn how to represent a biological research problem in a computational medium enabling them to explore a solution. The course on algorithms teaches them how to formalize problems and specify hypotheses, translating their research problems from a biological formulation into computer science terms and concepts. The course content itself is partly drawn from several collaborative activities, and actively encourages participatory design workshops and user studies.

¹ We refer to them as *poles* because each dimension is polarized with respect to the participants' perceived level of responsibility and the level of coupling between participation and outcomes.

² Local Seminar at Institut Pasteur, 1995

3.2 Scope of Responsibilities

This section describes the scope of weak and strong responsibilities toward each of the three poles. For each pole, we analyze what type of input is needed and how this input may collaboratively come from the two other poles: scientific theories together with participatory prototyping, to build software tools, participatory programming together with biological histories to provide input in workshops and software tools together with informal discussions to construct scientific hypotheses.

3.2.1 Collaboratively Building Computational Media

Active theories result when scientific hypotheses are placed in a computational medium and generate potentially worrisome questions of responsibility. Whenever a programmer implements an evolutionary model or the structure of a gene, he or she takes on responsibility for those who use it. Widely-distributed algorithms such as *BLAST*, *genscan* and *fastDNAmI*, spare biologists from choosing among hypotheses, delegating the responsibility to well-known scientific experts. The biologists need not question their assumptions or even know what those assumptions are. Increasingly, biology students are taught such models as part of their university curriculum and practitioners learn about them in continuing-education courses.

Software design results when participatory design prototypes are implemented in a computational medium and involves a range of responsibilities. In some short-term development projects, such as [37], participants expect working software as the outcome, requiring a high level of responsibility for all participants, but especially programmers. In long-term research projects, such as [25] and [18], outcomes are less immediate and may include both concrete research results, such as eventual working prototypes, and more ephemeral reflections on participatory design itself.

In our interviews, most workshop participants said they did not expect software as a direct outcome; they enjoyed such workshops as a place to meet other biologists and discuss ideas in an informal setting. They expressed confidence that their contributions would ultimately affect software, probably due to their past experience with one of the organizers, who had already developed software for them [17]. For most, then, the level of perceived responsibility for software design in participatory design workshops is low, for both biologists and programmers. However, there were some exceptions. One biologist complained that, for him, workshops would be useful only if carried out in the context of a specific project in which a computer scientist was hired to implement the software. This view favours only focused and specific developments rather than input to more general tools.

Algorithmics and interactive system design: requires active collaboration between algorithmic experts and computer system designers to build software tools. Simply providing a medium for testing and building hypotheses is not enough: biologists encounter too many problems while using scientific software. Participatory prototyping was important for designing usable and appropriate features, such as visualizing patterns extracted from a set of DNA sequences.

3.2.2 Providing Input to Participatory Workshops

Input to participatory workshops was mainly provided by tailoring activities or local software developments and the observation of scientific data analyses as scenarios for design. We also stress here the important and complex involvement of local developers in these participatory activities.

Tailoring: decisions result in persistent consequences. Henderson & Kyng [14] define tailoring as changing stable aspects not of the matter of the tool, but of the tool itself. Some factors act as barriers to customization [24], such as an anticipated loss of time or lack of interest in the tool. We have observed situations in which users were worried about taking responsibility for the changes to a software artefact resulting from tailoring. Similarly, [2] observes how a user, having entered the same set of programming statements several times neglects to use the “Save” function. This may be due to a fear of the potential consequences and corresponding responsibility for the resulting changes to the tool.

Local developer: often take on different types of responsibilities. We interviewed three local developers who quickly program tools that their user-colleagues use a few times. We found two apparently contradictory elements: the rhythm of software development activity and their participation in participatory design activities. Their activities are similar to that of extreme programming [31] which emphasizes:

- *short development cycles*: in this case, they were linked to weekly lab meetings.
- *staying close to the user*: here, the user is a colleague and the developer is him/herself a user.
- *courage*: (or taking responsibility for the fact that code will change). In bioinformatics, developers are expected to provide quick, working code examples and be available to fix problems. They need not create highly flexible code, but are responsible for the scientific outcome.

Local developers also participate in collaborative design activities, either implicitly, by seeding or reseeding of locally developed programs, customization artefacts or spreadsheets [10] or through direct participation in workshops and interviews. Customized artefacts and local developments may serve as material for building a collaborative design process [14, 8]. Local developers in bioinformatics are unlikely to seek systematization [36], since the code is specific to the current issues in the lab and is not meant to be reusable. This means that they do not always perceive their involvement in collaborative building of software tools as important. However, we found that their software artefacts served to highlight the importance of particular features which should be present in our software.

These local developers appreciated the participatory design workshops, both as users and as designers. They explained that their software problems are small in scope, highly varied and simple. In some cases, the developer runs the program and sends the results to a colleague, effectively acting as the program's user interface. But the local developers find the design workshops particularly helpful when it is up to them to create a separate GUI. They say this is where most problems occur: One gave an example of a Web program that displayed database search results in overly small chunks and said this sort of mistake could have

easily been avoided if they had run a short prototyping workshop. Interestingly, since they are already trained in biology, they do not need additional interviews or workshops to gain information about the domain; in fact, they are often hired because of their explicit biological knowledge.

Such local developers also play an important role as participants in workshops. They bring their knowledge of actual problems and a deep understanding of the communication problems that can arise between biologists and programmers. They often serve as cheerleaders for workshops, encouraging other biologists to participate.

3.2.3 Mediating Scientific Hypotheses

Both brainstorming and prototyping workshops provided an unexpected medium for producing scientific hypotheses: biologists used them as a forum for learning and exchanging knowledge and the topic of several workshops was directed towards the design of scientific algorithms.

A local bioinformatics culture has begun to develop at the Institut Pasteur. Workshops are generally popular: participants report that they enjoy meeting other biologists, to share problems and solutions and to update their current knowledge about bioinformatics tools. Biologists find it natural to discuss problems using their own data, walking through a scenario, often with paper mockups, to illustrate the issues. According to [32], what constitutes a *problem* for a biologist is actually an *instance of a problem* for a computer scientist. By creating a common ground for discussion across disciplines, both biologists and computer scientists can address problems from a perspective that makes sense to them. Other attempts to organize informal discussion groups have met with less success, perhaps because the latter involve only discussion and not active prototyping.

Most participants attend multiple workshops and it is not difficult to find new volunteers. All but a few spoke very highly of their experiences. Those who did not expressed frustration when the workshop did not immediately produce a software outcome that they could use. However, the rest felt that participating did not imply actual involvement in the official objectives of the workshop, nor did they ask about the ultimate outcomes. Also, a number of workshop participants have agreed to manage student projects from the Bioinformatics course, which brings with it a corresponding commitment to participate in additional workshops, even though they are billed as "only" for educational purposes.

Scientific Modelling Workshops: focused on how to specify the user's interactions with an algorithm, letting the user provide input as the variant of the algorithm's heuristics [21]. Participants first specified the actual problem and then developed a paper prototype. This led us to discover the most appropriate context and specify the interaction, i.e. how the user could trigger a specific algorithmic variant. For example, the user could manually highlight and thus specify an alternative within a previously-displayed result.

4. SUMMARY AND CONCLUSION

Our goal in this paper is to reflect on seven years of participatory design at the Institut Pasteur. Participatory design workshops proved popular: despite their busy schedules, many people

voluntarily attended multiple workshops. They viewed these as a valuable forum for scientific exchange, ranging from simple tips and techniques for using particular software tools to deep exchanges of scientific ideas. We have also seen how design workshops allow people to shift roles, particularly bioinformaticians who participate both as potential users as well as programmers.

The *Informatics in Biology* course provided us with additional insights, confirming what we saw in the participatory design workshops. The course projects also changed how the computer science professors viewed participatory design: they now take for granted that it will improve the quality of the student projects. Teaching biologists participatory design empowers them, giving them ideas for potentially useful programs and increasing their ability to communicate with programmers. (Note that enhancing communication between biologists and programmers is an explicit goal of the course.)

In summary, we see a weak coupling among the three poles (computational medium, scientific theory and participatory design activities) and the tensions that arise among them. One of the reasons participatory design works in this context is because it is both light enough and useful enough. In other words, neither biologists nor programmers have a strong commitment or responsibility with respect to participatory design workshops. Yet, with a relatively small amount of effort, they obtain something useful.

Situations requiring greater commitment would mean that fewer people would participate, although each effort would more likely result in a finished tool. Conversely, if less effort required were required, as in an open-ended discussion, participants would have fewer results. (Interestingly, it is not true that more people would participate. Such a discussion forum exists, but it is less well attended than the workshops.)

The trick with open participatory design workshops is to find an appropriate balance between low-responsibility and useful results. In general, these participants were satisfied with their participation in the workshops; whether or not specific tools resulted from a particular session.

In a similar way, end-user programming and local developments must be both light enough and useful enough: End-user programming involves partial, limited and non-obligatory participation, and is potentially collaborative, providing results that are useful to a wider population. Local developments are specific and short-term and need not lead to generic environments, but are clearly directly useful.

Finally, we see the same balance in the context of the student projects in the Bioinformatics course. The project leaders have a mixed but limited responsibility, as both clients of the software developed by the students and at the same time, as their teaching assistants. Their responsibility is light enough, because they have the full support of the computer science faculty and yet useful enough, because the individual projects meet their specific needs and correspond to actual research projects.

5. ACKNOWLEDGMENTS

Our thanks to the many biologists, programmers and bioinformaticians who let us interview them, videotape them at

work, and who participated in the design workshops. Special thanks to Katja Schuerer, Olivier Amanatian, Pascal Costa-Cuhna, Florence Hantraye, Bertrand Neron, François Huetz, Lionel Frangeul, Philippe Bouige, Michel Beaudouin-Lafon, Victoria Dominguez, Michaela Muller-Trutwin, Pierre Dehoux and Thierry Rose.

6. REFERENCES

- [1] M. B. Alan F. Blackwell. Applying attention investment to end-user programming. In *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, 2000, pp. 28–30.
- [2] O. W. Bertelsen, T. Eskildsen, and W. Sperschneider. Programming in the kitchen. In *Proceedings of INTERACT 2003*, Sept. 1-5, 2003, Zürich, Switzerland.
- [3] A. 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.
- [4] S. Bødker and K. Gronbaek. Design in action: From prototyping by demonstration to cooperative prototyping. In *Design at Work: Cooperative Design of Computer Systems*, Chapter 11., pp. 197–218. Hillsdale, New Jersey Lawrence Erlbaum Associates, 1991.
- [5] C. K. V. da Cunha and C. S. 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*, Apr. 2003.
- [6] A. DiSessa. *Changing Minds: Computers, Learning, and Literacy*. MIT Press, 1999.
- [7] A. DiSessa and H. Abelson. Boxer: a reconstructable computational medium. *Studying the Novice Programmer*, pp. 467–481. Lawrence Elbaum Assoc. 1989.
- [8] Y. Dittrich, L. Lundberg, and O. Lindeberg. End user development by tailoring: Blurring the border between use and development. In *Proceedings of the CHI'03 Workshop on End-User Development*, Apr. 2003.
- [9] M. Eisenberg. Programmable applications: Interpreter meets interface. In *ACM SIGCHI Bulletin*, 27(2):68–93, April, 1995.
- [10] G. Fischer and J. Ostwald. Seeding, evolutionary growth, and reseeding: Enriching participatory design with informed participation. In *Proceedings of Participatory Design Conference (PDC'02)*, T. Binder, J. Gregory, I. Wagner (Eds.), Malmö, Sweden, June 2002, CPSR, P.O. Box 717, Palo Alto, CA 94302, pp. 135–143, 2002.
- [11] G. Fischer and E. Scharff. Meta-design—design for designers. In *Proceedings of Designing Interactive Systems (DIS 2000)*; eds: D. Boyarski and W. Kellogg, New York City, ACM., pp. 396–405, Aug. 2000.
- [12] M. Gantt and B. A. Nardi. Gardeners and gurus: patterns of cooperation among cad users. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI '92)*, pp. 107–117, ACM, 1992.
- [13] J. Greenbaum. PD, a personal statement. In *Communications of the ACM*, 36(6):47, June 1993.
- [14] [A. Henderson and M. Kyng. There's no place like home: Continuing Design in Use In *Design at Work: Cooperative Design of Computer Systems*. J. Greenbaum and M. Kyng eds., chapter, pp. 219–240. Hillsdale, New Jersey Lawrence Erlbaum Assoc., 1991.
- [15] A. Kjaer and K. Madsen. Participatory analysis of flexibility. In *Communications of the ACM (CACM)*, 38(5):53–60, May 1995.
- [16] C. Letondal. Résultats de l'enquête sur l'utilisation de l'informatique à l'Institut Pasteur. Technical report, Institut Pasteur, Paris., Apr. 1999. <http://www.pasteur.fr/recherche/unites/sis/B6/9/9.html>.
- [17] C. Letondal. A web interface generator for molecular biology programs in Unix. *Bioinformatics*, 17(1):73–82, 2000.
- [18] C. Letondal. Programmation et interaction. Doctoral Dissertation. Université de Paris Sud (XI), Orsay, 2001.
- [19] C. Letondal. Software review: alignment edition, visualization and presentation. Technical report, Institut Pasteur, Paris, France, May 2001. <http://bioweb.pasteur.fr/cgi-bin/seqanal/reviewedit.pl>.
- [20] C. Letondal and O. Amanatian. Participatory design of pipeline tools and web services in bioinformatics. In *Proceedings of Requirements Capture for Collaboration in eScience Workshop*, NESCS, Edinburgh, Jan. 2004.
- [21] C. Letondal and U. Zdun. Anticipating scientific software evolution as a combined technological and design approach. In *Second International Workshop on Unanticipated Software Evolution (USE2003)*, 2003.
- [22] W. Mackay. Educating multi-disciplinary design teams. In *Proceedings of Tales of the Disappearing Computer*, Santorini, Greece. ACM Press, June, 2003.
- [23] W. Mackay, A. Ratzler, and P. Janecek. Video artifacts for design: Bridging the gap between abstraction and detail. In *Proceedings of ACM Designing Interactive Systems (DIS 2000)*, Brooklyn, New York. ACM, 2000.
- [24] W. E. Mackay. Triggers and barriers to customizing software. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'91)* pp. 153–160. ACM, 1991.
- [25] W. E. Mackay. Using video to support interaction design. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'02)*, Minneapolis, MN, DVD Tutorial. ACM, Apr. 2002.
- [26] W. E. Mackay, G. Pothier, C. Letondal, K. Bøegh, and H. E. Sørensen. The missing link: augmenting biology laboratory notebooks. In *Proceedings of 15th annual ACM symposium on User interface software and technology (UIST'02)*, Paris, pp. 41–50. ACM Press, Nov. 2002.
- [27] A. MacLean, K. Carter, L. Löfvstrand, and T. Moran. User-tailorable systems: Pressing the issues with buttons. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'90)*, pp. 175–182. ACM, 1990.
- [28] A. Morch. Three levels of end-user tailoring: Customization, integration, and extension. In M. Kyng and L. Mathiassen, editors, *Computers and Design in Context.*, pp. 51–76. MIT Press, Cambridge, MA, 1997.

- [29] V. L. O'Day, A. Adler, A. Kuchinsky, and A. Bouch. When worlds collide: Molecular biology as interdisciplinary collaboration. In *Proceedings of European Conference on Computer-Supported Cooperative Work (ECSCW'01)*, pp. 399–418, 2001.
- [30] J. Pane, C. Ratanamahatana, and B. Myers. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies*, 54(2):237–264, Feb.. 2001.
- [31] M. Rittenbruch, G. McEwan, N. Ward, T. Mansfield, and D. Bartenstein. Extreme participation - moving extreme programming towards participatory design. In T. Binder, J. Gregory, and I. Wagner, editors, *Proceedings of the Participatory Design Conference 2002 (PDC'02)*, Malmo, Sweden, pp. 23–25, June 2002.
- [32] K. Schuerer. Course in informatics for biology: Introduction to algorithmics. Technical report, Institut Pasteur, Paris, France, 2003.
www.pasteur.fr/formation/infobio/algo/Introduction.
- [33] O. Stiemerling, H. Kahler, and V. Wulf. How to make software softer: Designing tailorable applications. In *Designing Interactive Systems (DIS'97)* Amsterdam, pp. 365–376, 1997.
- [34] R. Trigg, T. Moran, and F. Halasz. Adaptability and tailorability in Notecards. *INTERACT'87*, North Holland, 1987.
- [35] R. 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)*, Aug. 1992, Larkollen, Norway, 1992.
- [36] R. H. Trigg and S. Bødker. From implementation to design: Tailoring and the emergence of systematization in CSCW. In *Proceedings of Computer-Supported Cooperative Work (CSCW'94)*, pp. 45–54, 1994.
- [37] P. Tuffery, B. Neron, M. Quang, and C. Letondal. i3DMol: Molecular visualization. Technical report, Institut Pasteur, Paris, France, 2003.
<http://www.pasteur.fr/letondal/biok/i3DMol.html>.
- [38] T. Winograd. From programming environments to environments for designing. In *Communications of the ACM (CACM)*, 38(6):65-74, June 1995.