Programs for People:

What We Can Learn from Lab Protocols

Keeley Abbott
School of EECS
Oregon State University
Corvallis, OR, USA

Christopher Bogart
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA

Eric Walkingshaw
School of EECS
Oregon State University
Corvallis, OR, USA

***Abstract*—Humans play an active role in the execution of certain kinds of programs, such as spreadsheets, workflows and interactive notebooks. Interacting closely with execution is especially useful when end-users are learning from examples while doing their work. In order to better understand the language features needed to support this kind of use, we investigated a particularly rigid and formalized category of "program" people write for each other: lab protocols. These protocols present a linear, idealized process despite the complex contingencies of the lab work they describe. However, they employ a variety of techniques for limiting or expanding the semantic interpretation of individual steps and for integrating outside protocols. We use these observations to derive implications for the design of interactive and mixed-initiative programming languages.**

I. Introduction

A growing number of end-user programming scenarios such as workflows and interactive notebooks serve the dual purpose of being how-to documents for humans to learn from, as well as mixed-initiative execution and programming environments for the accomplishment of real tasks. Minimalist learning theory [1] labels these situations "active learning": learning skills in the context of one's work. Although research on the cognitive aspects of active learning continues, less is known about what *programming language features* are needed to support this interplay between: (1) the designer-specified sequence of activities and explanatory examples, and (2) the user-specified sequences that emerge as the user transitions from learning to doing.

This interplay is a form of *mixed-initiative* execution [2] in which human and computer activities are interleaved. But unlike the carefully scripted interaction of, say, a wizard, in these active learning tools it is not always clear which party, computer or human, is leading the way. How can the same program not only communicate a fixed example but also continue to help the learner understand the hows and whys of generalizing that example to some real application? How can a mixed-initiative programming environment be designed to allow the user to take the reigns when necessary, without altogether abandoning the computer's guidance?

Like workflows and interactive notebooks, biological laboratory protocols also have this same dual purpose: they teach a process by example, while also serving as a template for applying the process to a real-world goal. However, there are many differences in how protocols are *executed* compared to computer programs. Correspondingly, there are also many differences in how protocols are *written* compared to computer programs. Much of the existing work with protocols has been sociological work focusing on their role in human activity. In this paper, however, we focus on the distinctive language features necessary to support humans executing protocols, as contrasted with computers executing programs.

Understanding protocols can lead us to discover new models for creating or improving current designs for mixed-initiative programs. If the language features found in protocols can be translated into design insights for mixed-initiative execution (Section I-A), they could potentially be integrated into existing programs for active learning, where human participants already play an active role in execution.

Furthermore, the design insights extracted from the language features found in protocols can inform the design of a mixed-initiative domain-specific language. This could be used to build a new class of flexible mixed-initiative execution programs, in which control of process and workflow would not be "dominated" by either the human or machine, but rather shifted along a continuum of control. This could allow both human and computer to work in concert to achieve a goal, using the best of each of their abilities.

A. Mixed-initiative execution

There are many programs that involve a mix of work between human and computer actors that can be construed as part of the broader spectrum of mixed-initiativity: distributed human computation [3] in which a computer organizes and integrates feedback from many people; wizards and interactive forms in which a human supplies information in between processing steps; computational notebooks like IPython [4] in which computers fill in intermediate results as humans edit programs; and the familiar interactive computing scenario where subprograms are offered to users as tools to use freely in the course of the their work.

In all these cases, however, either the computer or the human maintains primary control of the process. In *computer-led* mixed-initiative programs the wizard or form mostly forces the user along a particular path, and the distributed computing paradigm parcels out small tasks to humans at its own convenience. Conversely, in *human-led* mixed-initiative

programs, the interactive tool paradigm puts the computer's services completely at the user's disposal.

However there are tasks for which a more flexible exchange of initiative would be useful; none of these paradigms suit such situations. Computer-led mixed-initiative programs, for example, do not easily allow the human actor to *interleave* or *integrate* their own plans, goals, and constraints with the actions the computer needs them to perform. They also do not take advantage of resources or capabilities the human has beyond the program designer's expected audience. This might make the decision to submit themselves to the computer's guidance an unnecessarily significant time investment, both in carrying out the human tasks the computer might assign, and in tailoring the results the combined task ends up producing.

Conversely, human-led mixed-initiative programs may not provide enough guidance for a user without the experience to carry out the task at the macro level. For example, de Souza and Leitão [5] describe the plight of a Microsoft Word user trying to make page numbers appear consistently in a document where some pages are oriented vertically and some horizontally. In their example, the computer provided the pieces of functionality needed to complete the task, but did not succeed in communicating to the human which pieces and in what sequence the pieces should be applied.

B. Research questions and contributions

Although there has been research on the flexible relationship between human action and human plans, less is known at a granular level about the features and properties of written protocols that accommodate such flexibility. Thus the high-level goal of this research is to understand how people write detailed, reusable processes (that is, programs) to be carried out (executed) by other people, in a way that manages to convey both constraints and degrees of freedom for the protocol user. We look specifically at *biological protocols*, which are used to describe step-by-step how to accomplish tasks in a lab.

Protocols are written at different levels of formality for different purposes. At the least formal extreme, a scientist might create a protocol in their own lab notebook for personal use as a way to recall the process between iterations or when writing up results. Slightly more formal are protocols developed for specific internal or institutional use, which may address safety or quality standards specific to the organization, or may be employed to ensure consistency between actors. At the most formal extreme, there exist fully elaborated and highly refined protocols used to communicate novel techniques to other scientists around the world.

We focus on the formal extreme in this study. Specifically, we analyze peer-reviewed protocols published in *Cold Spring Harbor Protocols*. Unlike personal or internal protocols, these represent programs written for people unknown to the author(s) of the program. This aligns with our long-term goal of developing a language for mixed-initiative programming. Additionally, the protocols are more consistently formatted, and similar protocols contain similar visual and organization features, which makes them easier to analyze.

We have the following research questions:

RQ1. How are individual instructions expressed, and what is the granularity of an instruction? What balance of constraint and freedom do they allow, and how do they accomplish this?

RQ2. In what ways do protocol writers control and manage the factors that contribute to imprecise execution of protocols? Are there any distinguishing features that highlight the importance of steps *within* the protocol, and when there can be variance, how do protocol authors clue users in to this?

RQ3. How are relationships *among* protocols expressed in a ways that constrain or allow flexibility in composing them? Is there any inter-protocol structure, and if so, what does it look like? Are large protocols separated into multiple smaller protocols? Do protocols refer to external protocols?

In answering these questions, we make these contributions:

C1. A protocol coding scheme, summarized in Tables I and II.

C2. Descriptions and examples of how protocols specify instructions in terms of simple actions, goals, tasks, and open-ended tasks (RQ1; Subsections IV-B,IV-D).

C3. How protocols describe constraints and allowable variability within instructions (RQ2; Subsection IV-A).

C4. How control flow, error handling, and references to external protocols are specified in a lightweight fashion that relies on human adaptability and the flexibility of the instructions (RQ2 & 3; Subsections IV-C)

C5. Design implications for workflow tools (Section VI).

II. Related Work

As early as 1960 [6], researchers have noticed the differences between human instructions and computer programs, and called for more flexible flows of control between people and computers. Mechanisms for trading off initiative have been explored, for example discourse-based techniques (e.g. [7], [8]), allowing humans to shape machine learning searches by interactively adding constraints (e.g. [9], [10]), or simply explicitly hardwiring the interleaving of human and computer tasks (e.g. [11]). That prior work has been more about how to implement or improve mixed-initiative programs; in this paper we are investigating what role the written language behind mixed-initiative programs needs to play.

While programs and protocols seem superficially similar, the *pragmatics*[1] around their use are quite different. Researchers have noted that people use plans [13] and protocols [14] as resources, not as rigid instructions to be followed mechanically. Timmermans and Berg [15] argue that:

> *"subordination and rearticulation of the protocol to meet the primary goals of the actors involved is a sine qua non for the functioning of the protocol in*

[1]Horn and Ward [12] define pragmatics, a subfield of linguistics, as "the study of those context-dependent aspects of meaning a which are systematically abstracted away from the construction of content or logical form". Pragmatics explains how context can turn "it's getting late" from a literal observation about time to a hint for guests to leave after a party.

the first place. [...] Tinkering is a prerequisite for the protocols functioning."

Their claim is that people simply would refuse to adopt an overly inflexible protocol, because it would be so unlikely to precisely fit their goals, expertise, and resources.

Suchman [13] argues that human plans are more like descriptions or predictions than prescriptions of what actions a person will take given the most likely sequence of future actions; they are resources for anticipating likely future events. In programming language terms, perhaps, they are more declarative than imperative in that people draw on knowledge of the whole plan rather than blindly following each step.

Different actors adapt protocols to their own purposes. Timmermans and Berg emphasize that actors following a protocol have their own "trajectories", that is their own goals and priorities; they stray from the protocol to various extents, in various ways, for a variety of reasons. For example, a cancer patient may be careful to take medications prescribed in a research protocol, but might skip some measurement steps, since their priority is getting better, not furthering the research. Timmermans and Berg also cite the example of emergency personnel choosing to follow the CPR protocol even when the patient is not likely to be saved, using the time and visible activity as a way to soften the blow to a family coping with a sudden death. Lynch [14] describes protocols as being "situated in other procedures": they are not executed in a vacuum, but are meant to apply in situations where appropriate actors, goals, materials, and values are already present.

Protocols simplify coordination because, they "contain explicit criteria on whether, when and how next steps are to be taken. Personnel delegate some of their coordinating tasks to it, and the protocol appoints specific tasks to them." [15]. Sidner [7] et al. have shown one way this simplification can have practical benefit in human-computer interaction: knowledge about task state can help a computer guess the correct interpretation of an otherwise ambiguous human action.

Some existing software tools play a similar role to protocols as tailorable process descriptions. For example the Bioconductor project [16] uses literate [17] R programs, called "workflows" and "vignettes", to describe how various packages can be marshalled to perform tasks. In these programs the data file and task are merely illustrative examples; the program does not differ *semantically* from any other R program, but it is meant *pragmatically* to be used primarily as a resource for copy/paste creation of a new program to do a similar task.

Process description tools like CoScripter [18], and Automator [19] provide platforms for designing and annotating *scripts* for repetitive tasks. In the case of CoScripter, these scripts are limited to parameterized web-based tasks, where the parameters are instantiated from a database of local values. The "You" construct in CoScripter presents an opportunity for mixed-initiative execution by pausing the script and providing the user an opportunity to respond to a prompt. Like CoScripter, Automator provides similar functionality for turning control over to the user, but these are more structured than CoScripter's open-ended "You" construct.

III. Methodology

We performed a qualitative analysis of lab protocols using open-ended coding. In this section, we describe the coding scheme we used and discuss how it was developed and applied. We present the coding scheme in two parts. In the first, we describe the coding scheme applied to each *step* of a protocol. These codes provided the data needed to address RQ1 and RQ2. In the second, we describe the coding scheme applied to each *external reference* in a protocol. These codes provided the data needed to address RQ3.

A. Coding scheme for protocol steps

To start, each of the authors reviewed a set of 2–3 protocols and independently developed a coding scheme to categorize each protocol step and to note features in each step relevant to our research questions. By comparing and discussing our individual efforts, and by working through each of the annotated protocols together, we developed a single coding scheme by consensus to use throughout the protocol analysis.

Within this scheme, each step of the protocol is assigned (1) a *kind*, which describes the primary action of the step (e.g. physical, measurement, monitoring); (2) a *precision*, which describes how the step is stated (e.g. as an instruction, as a goal); and (3) zero or more *additional features* relevant to our research questions (e.g. gives additional advice, describes contingencies). The complete coding scheme is described in Table I. For each kind, precision, and feature, we provide an example from the protocols assigned that code. We reference protocols using the notation [ID; Step], where *ID* is the protocol ID number, and *Step* is the step or section number.[2]

Next, two of the authors coded the same protocol using the established coding scheme to establish agreement in the use of the scheme. We used Cohen's Kappa [20] to measure the agreement on each of the three components (kind, precision, features) of the coding scheme. We chose Cohen's Kappa as a conservative measure that corrects for coincidental agreements since we checked agreement on a small sample. The results of the analysis are listed below.

- *Kind*: 52 coding assignments, of which 50 were identically coded by both researchers (Kappa=0.868).
- *Precision*: 52 coding assignments, of which 47 were identically coded by both researchers. (Kappa=0.732).
- *Features*: 55 coding assignments, of which 51 were identically coded by both researchers. (Kappa=0.815).

This demonstrates a "substantial" to "almost perfect" agreement [20] for each of the components. Therefore, one author coded an additional 19 protocols, bringing the total number of coded protocols to 20. The coded protocols include randomly chosen recent free protocols from CSHP's website where ten were from the Bioinformatics category, five from the Neuroscience category, and five from the Laboratory Organisms category.

[2]To retrieve a protocol, click its ID number in the PDF version of this paper, or append the ID number to the url "http://dx.doi.org/10.1101/pdb.prot".

Kind	Example
`physical` *Manipulate physical objects*	"Slowly pour 14 mL of lysis buffer …" [5384; 3]
`virtual` *Manipulate virtual objects*	"Using NimbleScan software, open the scanned image …" [5385; 56]
`cognitive` *Evaluate or analyze*	"… check the quality of the amplification." [4974; 12]
`measurement` *Take a measurement*	"Quantify the DNA yield …" [4974; 14]
`non-human` *No required human action*	"This method was adapted from an original protocol …" [4918; pg. 3]
`selection` *Make a selection decision*	"Select a gene/genomic locus …" [5491; 1]
`monitoring` *Watch or monitor*	"… monitor wet slides …" [4706; 12]
`branching` *Perform parallel actions*	"This step can be performed during … (Step 12)." [5237; 17]
`looping` *Repeat a process or step*	"Repeat Step 10." [5385; 12]

(a) *Kind* of action described in the step.

Precision	Example
`instruction` *Straightforward instruction*	"Anesthetize the animal by injecting intraperitoneally with 16 µL ketamine/xylazine per gram of body weight." [5410; 1]
`goal-directed` *Describes goal, not action*	"Reduce the volume to a 15-µL concentrate using …" [4974; 20]
`task-directed` *Describes task, not goal/action*	"Examine the ruffled morphology characteristic of mats by …" [085076; 3.ii]
`open-ended` *Requires creativity or ingenuity*	"Determine the amount of MNase to use for each cell type experimentally." [5237; 23]

(b) *Precision* of the description of the step.

Features	Example
`advice` *Suggested method or process*	"For best results …" [4918; 8]
`constraints` *Avoid this method or process*	"Do not use syringe needles to load the pipettes …" [5201; 1]
`expected outcomes` *Expected result or outcome*	"This provides a final effective dilution of 10–6 viable cells per milliliter …" [5492; 7]
`optionality` *Optional process*	"(Optional) Remove cells for additional analysis." [085076; 4.iii]
`wiggle room` *Alternative method or process*	"This mixture can be stored for up to 6 mo at -20°C." [4974; 8]
`contingencies` *In-line troubleshooting*	"If the density of cells and chromosome spreads is insufficient, pellet the cells, resuspend in a smaller volume of fixative, and repeat with fresh drops." [4706; 8]
`information` *Background information*	"The PCR product is denatured and the nonbiotinylated strand is removed in this step." [5491; 37]
`reference` *Outside source*	"… see Preparation of Fixed Xenopus Embryos for Confocal Imaging (Wallingford 2010b)." [5427; 2.i]
`mapping` *Map instruction over a set*	"Treat all reserved aliquots with 20 µg of proteinase K for 30 min at 65°C." [084848; 33]
`visual pattern match` *Human visual processing*	"View whole-cell K+ currents and record for further analysis (Fig. 3)." [5014; 35]

(c) Additional *features* that appear in the step.

TABLE I: Coding scheme for protocols steps. Each step is assigned a kind, a precision, and zero or more features.

B. Coding scheme for references

To address RQ3, we developed a separate coding scheme for references to other instructions, protocols, papers, or policies. The coding scheme was developed through a similar consensus-based approach as described in Section III-A.

References in the analyzed protocols can be separated into two broad categories. The first category includes references to other instructions, protocols, or policies that are intended to be *integrated* into the execution of the current protocol. The second category includes references to other documents that provide supplementary information, but are *not integrated* into the execution of the protocol.

Within our coding scheme, each reference is assigned (1) a *reference type*, which indicates the type of document referred to (e.g. manufacturer instructions, institutional protocol); (2) either an *integration strategy* for integrated references (e.g. use concurrently), or alternatively the *role* of a non-integrated reference (e.g. background information); (3) zero or more additional features. The complete coding scheme is described in Table II.

To establish agreement in the use of the coding scheme, two researchers coded the same two protocols. Again, we apply Cohen's Kappa as a conservative measure of agreement for each of the coding components. The results of this analysis are listed below.

- *Reference Type*: 57 coding assignments; 50 were identically coded by both researchers (Kappa=0.748).
- *Integration Strategy*: 55 coding assignments; 50 were identically coded by both researchers (Kappa=0.914).
- *Non-integration Role*: 57 coding assignments; 55 were identically coded by both researchers (Kappa=0.737).
- *Features*: 58 coding assignments; 51 were identically coded by both researchers (Kappa=0.611).

This shows "substantial" to "almost perfect" agreement [20] for three of the four components. We observed only "moderate" agreement on the *features* component. Since this category has a small number of possibilities, and a high frequency for one of them (link), it is more difficult to distinguish the raw agreement rate we observed (88%) from chance. Since the accuracy of this component is not central to our results, we did not perform another iteration. Therefore, one author coded the remaining 18 protocols included in our data set.

IV. Results

In this section we summarize and interpret the results of the study in response to the research questions listed in Section I-B. In the spirit of open science, our raw results (coding data and calculations) are published in-full online.[3]

A. Precision of protocol steps

In contrast to Licklider's claim that "instructions directed to computers specify courses; instructions directed to human beings specify goals" [6], we found a surprising number of steps in the form of straightforward `instructions` (Table Ib).

[3]https://github.com/lambda-land/ProtocolStudy-Data

Reference Type	Example
`manufacturer` *Manufacturer instructions*	"... according to the manufacturer's recommendation ..." [4974; 4.i]
`academic` *Academic protocol or paper*	"This can be accomplished in yeast by overexpressing herpes simplex virus thymidine kinase to phosphorylate nucleosides (Lengronne et al. 2001)" [5385; 2.ix]
`laboratory` *Institutional protocol or policy*	"... using standard restriction digestion protocols." [5168; 7.iii]
`equipment` *Equipment instructions*	"For other array platforms, perform washes as directed by the microarray supplier." [5385; 8.ii]

(a) *Type* of document referred to.

Integration Strategy	Example
`before` *Use reference before starting*	"The double-stranded RNA needed for RNAi is prepared as in Preparation of Double-Stranded RNA for Drosophila RNA Interference (RNAi)." [4918; Related Information]
`concurrent` *Use reference in parallel*	"Imaging can also be performed simultaneously with ..." [5427; Discussion]
`splice` *Use reference, then continue*	"Combine PCR products of replicate samples and purify them using the QIAquick PCR Purification kit ..." [4974; 4.i]
`merge` *Combine reference with protocol in an unspecified way*	"... the power of such a screening method is further enhanced when it is combined with the simple pyro- screening enrichment protocol, another pyrosequencing-based innovation we have developed (Liu et al. 2009)." [5491; 9.i]

(b) *Strategy* for integrating a reference into the protocol.

Non-integration Role	Example
`derivation` *Derived from reference*	"The 6C assay combines three different methodologies: chromosome conformation capture (3C) (Dekker et al. 2002), chromatin immunoprecipitation (ChIP), and cloning (Fig. 1)." [5168; 1.i]
`comparison` *Similarity to reference*	"... it may be useful to employ other labeling methods, such as those described in ..." [5460; 1.i]
`background` *Additional information*	"... (for details, see CSH Protocols articles Embedding Mouse Embryos and Tissues in Wax and Sectioning Mouse Embryos)." [4820; 4.ii]

(c) *Role* of a non-integrated reference.

Features	Example
`modifications` *Use reference with modification*	"If another species is used, surgical conditions and drug dosages should be optimized for that species." [5410; 2.iii]
`link` *Location/link to external source*	"For imaging cleared embryos, see Preparation of Fixed Xenopus Embryos for Confocal Imaging (Wallingford 2010b)." [5427; 5.i]

(d) Additional *features* that appear in the reference.

TABLE II: Coding scheme for references. Each reference is applied a type, either an integration or non-integration code, and zero or more features.

These are expressed as directives to the user that contain little to no ambiguity about the task the user is to complete:

- *"Wash the cells with 5 mL of ice-cold PBS containing protease inhibitor cocktail."* *[5168; 4]*
- *"The moment that the slide lightly taps the tip and the tip breaks, a small amount of the dye will leak from the needle tip. Release the 'clean' button."* *[4918; 7.ii]*
- *"Remove the Schneider's insect medium from the samples."* *[5460; 6]*

Overall, we observed 777 examples of direct instructions out of 833 total coded steps (93.3%).

The next most precise kind of step was also the second most frequent. A `goal-directed` step describes an expected goal, but leaves a degree of ambiguity about *how* the user is to achieve it. Most examples of `goal-directed` steps we observed involve bringing materials to a particular physical state such as temperature or concentration:

> *"Add BrdU to a final concentration of 400 μg/mL and incubate for the desired period."* *[5385; 2]*

Others instruct the actor to produce a measurement or calculation without spelling out a procedure:

> *"Calculate the percentage of the total responses."* *[5453; 12]*

Overall, we observed 33 examples of `goal-directed` steps, across 11 of 20 protocols.

Moving further from straightforward instructions, a `task-directed` step describes a task where the goal itself is ambiguous, or depends on the higher-level goals of the person carrying out the protocol. Many of the examples we observed instruct the actor to perform an evaluation without providing an explicit indication of what the actor should be evaluating or looking for:

- *"Analyze 10 μL of the PCR products on a 1.5% agarose gel to check the quality of the amplification."* *[4974; 12]*
- *"Using a microscope (100X and 400X total magnification), evaluate the degree of digestion."* *[5014; 13.ii]*

We observed 21 examples of `task-directed` steps, across 12 of 20 protocols.

Finally, we observed `open-ended` steps in just two of the protocols. These steps are the least precise. They provide only a general description of the intent of a step, but leave the goal and task unspecified.

> *"DNase concentrations to achieve optimal smearing sizes may differ for each cell line and therefore must be determined empirically for each cell type."* *[5384; 20]*

This type of step seems to require more creativity or ingenuity by the user. As a collection of concrete and highly-polished processes, it is not surprising that our data set provides relatively few examples of such steps.

B. Linearity of protocols

Overall, the protocols we analyzed were surprisingly linear. That is, they consist mostly of long sequences of instructions

with very few `branch` or `loop` steps.[4] The branches that do exist are expressed by directing the actor to skip ahead or back to a particular step (similar to a GOTO) depending on the result of a measurement or the task the actor is trying to accomplish. Similarly, loops are expressed by directing the actor to repeat a certain range of steps.

- *"Proceed with step 7 during centrifugation . . . "* *[5384; 6]*
- *"Repeat this step with the upper phase . . . "* *[5168; 41]*
- *"Repeat steps 25.iii–25.v."* *[5168; 25.vi]*

We observed that only 10 of 20 protocols include any form of branching or looping at all, and among these non-linear protocols, they averaged just 2.3 `branch`/`loop` steps per protocol. One protocol [5385] contained 8 `loop` steps but no other protocol had more than 3 branches and/or loops. When compared to all steps, `branch` and `loop` steps were just 2.76% of the total coding instances.

In a few instances, instructions resemble higher-order functions [21], such as the following `map`-like step that applies an instruction to every member of a set.

> *"Slowly pipette 120 µL of nuclei suspension into tubes #1–#7 . . . "* *[5384; 9]*

Higher-order functions may be used to "flatten" a non-linear portion of a protocol to make it appear more linear. A single step that maps an instruction over multiple physical elements (tubes, phases, beads) is structurally simpler than `branch`/`loop` steps that reference other steps by name.

C. Accommodating imprecise execution

Previous research has shown that protocols are only loosely followed in practice (see Section II). A protocol describes an idealized course of action, but the person executing it will frequently deviate from this course by modifying, reordering, adding, or even skipping steps. By way of analogy, consider a program or protocol as describing a path. In a computer program, the path specified is very narrow, like a tightrope, which the computer follows precisely. In a human protocol, the path is much wider, and the person following it may wander freely from side to side, stop to smell the flowers, and so on.

We observed six different features within the analyzed protocols that explicitly accommodate this imprecise execution model by broadening or narrowing the path, or by helping the user find their way back when lost. These are the first six features listed in Table Ic. The frequency of each of these six features is summarized in Table III.

Broadening. The features `optionality` and `wiggle room` represent attempts by the protocol writer to *broaden* the path. Quite different from anything found in computer programs, steps that exhibit these features explicitly encourage actors to vary from the narrowest interpretation of the step semantics and rely on their own judgment.

Steps with the `optionality` feature indicate explicitly that a step is optional. While people are free to use their judgment to skip steps at any point in a protocol, the optionality feature highlights to the actor that a particular step is not essential to the ultimate goal of the protocol.

> *"At this point, the cell pellet can be snap-frozen in liquid nitrogen and stored at -80°C until use."* *[5168; pg 5]*

Similarly, steps with the `wiggle room` feature provide an explicit range of execution, suggesting, for example, that a particular measurement can be estimated.

> *"Incubate the mixture for 15-30 min at 70°C . . . "* *[5491; 12.ii]*

Code	No. of protocols	No. of instances
`advice`	19 (95%)	108 (12.9%)
`constraints`	20 (100%)	108 (12.9%)
`expected outcomes`	9 (45%)	18 (2.04%)
`optionality`	10 (50%)	23 (2.76%)
`wiggle room`	14 (70%)	35 (4.20%)
`contingencies`	6 (30%)	6 (0.72%)

TABLE III: Frequencies of features supporting the imprecise execution of protocols.

Narrowing. The features `advice`, `constraints`, and `expected outcomes` represent attempts by the protocol writer to *narrow* the path. These features were much more common. The presence of narrowing features makes it clear that protocol writers assumes that actors will wander from the path they are describing. These features focus the actor's attention on important aspects and expectations of the protocol.

Steps with the `advice` feature provide additional tips to help the actor navigate the most important sections of the protocol.

> *"For best results, inject the embryos with the needle at an angle slightly greater than 45° relative to the embryo surface in the posterior quarter of the embryo . . . "* *[4918; 8]*

Steps with `constraints` explicitly constrain the execution path to ensure that the actor doesn't deviate at an important point. For example, the following instruction about a chamber is followed by a constraint about its positioning.

> *"Close the hybridization chamber tightly and incubate it in the dark for 16 h in a 60°C water bath. The hybridization chamber should be submerged in water, but direct contact with the bottom of the water bath should be avoided."* *[4974; 29]*

Constraints provide a way to alert the actor that an extra level of care and precision should be used on a particular step.

Finally, steps with the `expected outcomes` feature provide a sort of checkpoint after an instruction to ensure that the actor can recognize that they are still on the right path.

> *"This step typically takes 12 min at 14,000g."* *[4974; 20.iii]*

Expected outcomes are similar to assertions in computer programming languages.

[4]However, we did observe many references to external protocols, which correspond roughly to subroutine calls and complicate the question of control flow somewhat. See Section IV-D.

Code	No. of protocols	No. of instances
`before`	20 (100%)	99 (45.0%)
`concurrent`	16 (80%)	61 (27.7%)
`splice`	13 (65%)	39 (17.7%)
`merge`	8 (40%)	21 (9.5%)

TABLE IV: Frequency of integration strategies for references to other instructions and protocols.

Finding the way back. Steps with the `contingencies` feature provide an explicit way to recover from errors, which might have been caused by deviating from the protocol.

> *"Use the markings on the 1.5-mL tube to provide an estimate of the pellet volume. If the pellet is very small, resuspend it in 300 µL of cell lysis buffer."* *[084848; 19]*

This feature was quite rare: only 6 instances in our data set. It seems that actors are typically expected to recover from errors on their own.

D. Integrating other instructions and protocols

One of our research questions (RQ3) concerned relationships between protocols and other instructions, whether protocols refer to each other and how they integrate. We observed that protocols refer to other resources very often. We identified a total of 261 references across all 20 protocols. The vast majority of these (246, 94.3%) were to explicitly named instructions or protocols (e.g. cited academic protocols or specific manufacturer's instructions), while the remainder were generic references to documents in the actor's lab (e.g. lab policies or ethics documents). We coded all 261 references using the coding scheme summarized in Table II.

Recall from Section III-B that we can classify references according to whether they affect the semantics of the current protocol, that is, whether they *integrate* with the protocol or merely provide background or supplementary information. We observed that 220 (84.3%) of the references integrated with their protocol. The observed instances of each integration strategy are summarized in Table IV.

The `splice` integration strategy most often occurred with manufacturer's instructions for lab "kits", packages that include solutions and equipment needed to do a common lab procedure. A spliced-in reference is similar to a subroutine call in a computer programming language.

A large source of `before` and `concurrent` integrations was a list of "recipes" and "warnings" given in the preamble of every protocol. Recipes are protocols or instructions that must be followed before the current protocol can be executed. Warnings are additional instructions that must be kept in mind during the execution of the protocol, typically for safety reasons (coded as `concurrent`). For example:

> *"<!>DTT (Dithiothreitol; 0.1 M)"* *[5559; materials]*

This warning refers to a brief appendix entry:

> *"DTT (Dithiothreitol) is a strong reducing agent that emits a foul odor. It may be harmful by inhalation, ingestion, or skin absorption. When working with the solid form or highly concentrated stocks, wear appropriate gloves and safety glasses and use in a chemical fume hood."* *[5559; cautions]*

This DTT warning is typical of a `concurrent` integration; it alters the semantics of the rest of the protocol by specifying how to handle DTT safely. In fact, DTT appears only once in the protocol among a list of ingredients in step 5 [5559; 5]. While this step just says to "prepare the mixture", the warning is much more specific and demanding: the worker must wear gloves and glasses and perform the action in a specialized workspace. Also, unlike a spliced integration, a concurrent integration is applied implicitly throughout a protocol, to be triggered when relevant. In this example, the reference appears at the top of the protocol; the user must notice it and interleave it appropriately wherever DTT appears later on. Sometimes the integration point(s) are left even more implicit:

> *"It is essential that you consult the appropriate Material Safety Data Sheets and your institution's Environmental Health and Safety Office for proper handling of equipment and hazardous materials used in this protocol."* *[085076; materials]*

In some ways, the `concurrent` strategy is similar to aspect-oriented programming [22]. It separates concerns related to specific materials or actions and the actor is responsible for weaving those concerns into the execution of the protocol.

The `modification` feature captured 11 instances where a reference was integrated with some explicit changes.

> *"Purify the eluate using a MinElute PCR Purification Kit according to the manufacturer's protocol with the following modifications: . . . "* *[5385; 33-33.i]*

Finally, we observed 21 instances of the `merge` integration strategy across 8 of the protocols. In some cases, these references seemed intended to help the user know when to choose *this* protocol by describing how the protocol might fit into a larger workflow and combine with other processes. The 17 instances across 8 protocols of the `comparison` role for non-integrated references also seemed tailored for this purpose, as illustrated by the following example.

> *"Pei et al. (1997) describe an alternative method to isolate Arabidopsis guard cell protoplasts suitable for patch clamping."* *[5014; pg 6]*

V. Discussion

Although the results in Section IV-A show that protocols are *written* precisely, Section II discusses the substantial evidence that, in practice, protocols are not *executed* precisely. Rather, a protocol describes an idealized path to follow in order to accomplish a task. Protocol authors seem to be well-aware of how protocols are used and tailor the presentation of protocols to support this execution strategy by alerting the protocol user about which shortcuts and side-trips are dangerous and which might be useful. By calling extra attention to especially important instructions, or providing checks to help the executor determine whether they are still on track to accomplish their task, the protocol author builds in checkpoints, redundancy,

and safety measures to ensure the task will be successfully accomplished.

These features demonstrate how differences between humans and computers motivate differences between protocols and programs. On the one hand, the computer does exactly what it is told, which is useful for predictability and repeatability. On the other hand, the human can use common sense, which supports flexible error handling and recovery. Human common sense stands in sharp contrast to the commonly observed situation in imperative programs where the computer blithely chugs away after an error, unless the programmer has built in an explicit, complex system of fail-safes.

Similarly, the ambiguity and allowance for variation written into protocols are advantageous by allowing the user to *interleave* or *integrate* other plans: their own plans, goals, and constraints, and the auxiliary recipes, warnings, and procedures suggested by the protocol's author. Since protocol writers provide explicit allowances for variation, and explicit constraints and checks to reign in the variation they expect their users to make on their own, this suggests they know these features are important and must be supported if they want users to adopt their protocols.

The linearity that we observed in protocols may also play a role in supporting the integration of user goals and outside plans. Since protocol authors describe the path as a straight line rather than a garden of forking paths, they make it easier for users of the protocol to understand the document as a whole, and reason about the consequences of skipping, adding, or varying steps. That is, linearity allows the user a greater "field of view" when it comes to execution of the protocol. Rather than seeing a small slice of the process, they have access to many steps in advance, allowing them to plan and incorporate outside information in a more interactive way.

VI. Implications for design

The design of literate "workflow" tools may benefit from our findings. Workflows, such as those provided by Bioconductor [16] or Galaxy [23], and interactive computation tools, such as IPython [4], are like protocols in that they describe a complex sequence of activities that a user may need to adapt to their own purposes, but the user is manipulating code and program entities instead of beakers and solutions. These tools could borrow the following features from protocols:

Path width. Authors of workflow *tools* could include explicit ways to add and execute invariants and tests that constrain a user's modifications to a workflow step. Or even without such changes, authors of *workflows* could add assertions and tests in locations where users are most likely to tinker with the script. Workflow users presumably use trial-and-error to adapt scripts, and could benefit from such hints about the expected scope of the exploration process.

Connection to original. Workflow tools could preserve and make visible the relationship between an original demonstration protocol and any script the user creates by adapting it. When users edit example code to meet their own needs, the original becomes harder to recover and relate to the tailored script. This forfeits some of the benefits that protocols provide since they no longer communicate adherence to a normative process, and their hints about the path and its width may be lost or damaged due to the user's experimentation.

Composition. Some workflow tools do not really allow for composition the way protocols do: IPython for example can only call plain Python code, and Bioconductor workflows can only call ordinary R packages. Perhaps auxiliary workflows could be referenced in some of the ways protocols are (Table IIb), but adding facilities to help the user interleave the workflows' actions, and using the assertions and tests mentioned above to check that their actions interleave correctly.

Our previous work on variation languages is relevant for the design and implementation of these features. We have developed techniques for adding variation points to programs and other data structures, which can be used to explicitly broaden execution paths [24]–[26]; and we have developed editing techniques for maintaining a history of explicit variations which can be used to maintain a connection to the original example as well as previously explored alternatives [27].

VII. Conclusion

At the start of this study, we wanted to know how to improve the state of the art in mixed-initiative execution, to help people and computers collaborate more flexibly and effectively on tasks. This is important because in many cases, such as active learning scenarios, neither the computer nor the user alone have complete information about the sequence of tasks that need to be completed. To that end we investigated biological protocols, to mine them for ideas about what the human side of a human-computer program should look like. What we found embedded in these documents was a toolbox of ways to specify tasks, that take advantage of human capabilities:

- Ways of communicating both limits and liberties from the central activity sequence
- A simplified, linear structure, perhaps aimed at lowering the perceived risk of adoption, and cost of tailoring
- Simple statements of expected intermediate results in lieu of elaborate error-handling mechanisms

These tools seem immediately applicable to the design of workflow tools that help end-user programmers learn a complex task while adapting it to a real application. Beyond that, we hope to use these results to inform the design of more effective mixed-initiative features in programming languages, in which human-computer interaction is a continuum of control, rather than as a function of "domination" by one or the other.

Acknowledgments

We would like to thank Margaret Burnett and Stephen Ramsey for helpful comments and advice in the early stages of this research, Stephen for directing us to CSHP as a repository of high-quality protocols to study, and the generous support of the Alfred P. Sloan foundation.

References

[1] J. M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT press Cambridge, MA, 1990.

[2] E. Horvitz, "Principles of mixed-initiative user interfaces," *ACM SIGCHI 1999*, pp. 159–166, May 1999.

[3] A. J. Quinn and B. B. Bederson, "Human computation: A survey and taxonomy of a growing field," *SIGCHI Conference on Human Factors in Computing Systems*, pp. 1403–1412, 2011.

[4] F. Perez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

[5] C. s. de Souza and C. F. Leitão, "Semiotic engineering methods for scientific research in HCI," *Synthesis Lectures on Human-Centered Informatics*, vol. 2, no. 9781598299441, pp. 1–122, 2009.

[6] J. C. R. Licklider, "Man-computer symbiosis," *IRE Transactions on Human Factors in Electronics*, vol. HFE-1, no. 1, 1960.

[7] C. L. Sidner, C. Rich, and N. Lesh, "Collagen: Applying collaborative discourse theory to human-computer interaction," *AI Magazine*, vol. 22, no. 4, pp. 15–26, 2001.

[8] C. Miller and R. Larson, "An explanatory and "argumentative" interface for a model-based diagnostic system," in *ACM Symposium on User interface Software and Technology (UIST)*. Monterey, California, United States: ACM, 1992, pp. 43–52.

[9] N. Lesh, J. Marks, C. Rich, and C. L. Sidner, ""man-computer symbiosis" revisited: Achieving natural communication and collaboration with computers," *IEICE Transactions on Information and Systems*, vol. E87-D, no. 6, pp. 1290–1298, 2004.

[10] T. Kulesza, S. Stumpf, W.-K. Wong, M. M. Burnett, S. Perona, A. Ko, and I. Oberst, "Why-oriented end-user debugging of naive bayes text classification," *ACM Transactions on Interactive Intelligent Systems*, vol. 1, no. 1, pp. 1–31, 2011.

[11] T. Lau, E. M. Haber, T. Matthews, and G. Leshed, "Coscripter: Automating & sharing how-to knowledge in the enterprise," in *SIGCHI Conference on Human Factors in Computing Systems*. Florence, Italy: ACM, 2008, pp. 1719–1728.

[12] G. L. Ward and L. R. Horn, *The Handbook of Pragmatics*. Blackwell Publishing, 2004.

[13] L. A. Suchman, *Plans and Situated Actions: The Problem of Human-machine Communication*. Cambridge University Press, 1987.

[14] M. Lynch, "Protocols, practices, and the reproduction of technique in molecular biology," *The British Journal of Sociology*, vol. 53, no. 53, pp. 203–220, 2002.

[15] S. Timmermans and M. Berg, "Standardization in action: Achieving local universality through medical protocols," *Social Studies of Science*, vol. 27, no. 2, pp. 273–305, 1997.

[16] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry *et al.*, "Bioconductor: Open software development for computational biology and bioinformatics," *Genome Biology*, vol. 5, no. 10, p. R80, 2004.

[17] D. E. Knuth, "Literate programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984.

[18] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. a. Lau, "End-user programming of mashups with vegemite," *IUI 2009*, pp. 97–106, 2009.

[19] Apple. Mac basics:automator. [Online]. Available: https://support.apple.com/en-us/HT2488

[20] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[21] P. Sestoft, "Higher-order functions," in *Programming Language Concepts*. Springer, 2012, pp. 77–91.

[22] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP 1997-Object-oriented Programming*. Springer, 1997, pp. 220–242.

[23] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor *et al.*, "Galaxy: a platform for interactive large-scale genome analysis," *Genome Research*, vol. 15, no. 10, pp. 1451–1455, 2005.

[24] M. Erwig and E. Walkingshaw, "The Choice Calculus: A Representation for Software Variation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 1, pp. 6:1–6:27, 2011.

[25] E. Walkingshaw, "The Choice Calculus: A Formal Language of Variation," Ph.D. dissertation, Oregon State University, 2013.

[26] E. Walkingshaw, C. Kästner, M. Erwig, S. Apel, and E. Bodden, "Variational Data Structures: Exploring Trade-Offs in Computing with Variability," in *ACM SIGPLAN Symposium on New Ideas in Programming and Reflections on Software (Onward!)*, 2014, pp. 213–226.

[27] E. Walkingshaw and K. Ostermann, "Projectional Editing of Variational Software," in *ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE)*, 2014, pp. 29–38.