Complementary analyses of exactness: Writing code and self-repair in meetings

This text is a collection of excerpts and analyses that expand and complement the analyses in the paper "Managing exactness and vagueness in computer science work: A conceptualization of programming and analysis of self-repair in meetings", to be published in *Social Studies of Science*. Section 1 compares the collective writing of computer code with the collective writing of natural language text, while section 2 and 3 complement the analyses of self-repair and hedges in the main paper.

Table of contents

1 Writing code compared to writing in a natural language

While the paper conceptualizes writing code as requiring exactness, it does not include excerpts where the computer scientists studied directly work with code because this is usually done alone and thus does not lend itself well to methods of investigation that focus on natural language in written or spoken form. The typical situation in the projects investigated is that doctoral students or postdocs write code individually and show the results of their work to the other project members. Only rarely do the computer scientists show code directly or work on it collaboratively. The next excerpts in section 2.1.1 provides a glimpse into the practice of collaborative code writing, whereas the excerpts in section 2.1.2 briefly consider collaborative writing in a natural language. The purpose of the analysis is to illustrate the claims made in the paper. The analysis of collaborative code writing should be considered to be exploratory because it cannot be compared to other examples of this practice.

1.1 Writing exact code and talking about it simultaneously

The following excerpt is an example of collective code writing. It shows how the writing of code and the simultaneous talk about the code are interrelated. The excerpt is taken from the "medical evidence project" (ME project). This research project is concerned with the conceptual and computational systematization of evidence from clinical trials. The group wants to develop a program that enables practitioners from the medical domain to search for and compare treatments for diseases. The excerpt below is taken from the beginning of a meeting attended by four project members plus the ethnographer, where one participant is writing the code of a first prototype and the three other project members observe this work, comment on it, and discuss it.

In order to systematize medical evidence conceptually, the group draws on argumentation theory, assuming that clinical trials include arguments for or against treatments that can be summarized, which explains some of the technical terms they use such as "base argument". The group is sitting around a table in an office and a projector projects what RD is doing on his laptop. RD is writing in Java, using the development environment Eclipse. Lines of transcript in italics below describe what RD is doing on the screen, focusing on what he is typing.

#III: the "BaseArgument"

```
01   RD:  .hhh ehm so what I'm thinking is whether we can
          have something like eh: (1.4) (0.5) specific ehm
          base argument.
02        (3.8)
03        Looks in menu for "create Java class" option
04   CG:  Like for drug superiority.
05   RD:  Yeah,
06        Opens "class creator"
07   CG:  =mhm
08   RD:  Exactly. Ehm: So argument, (0.4) base argument,
          (1.0) base arg- or argument from superiority no eh
          base a[rgument,]
09        Types "Argument," then deletes and writes
          "BaseArgument"
10   PV:        [efficacy] °yeah°
11   RD:  Argument from (0.7) superio:rity: (0.7) efficacy.
          Okay? (0.9)
12        Continues typing "FromSuperiorityEfficacy"
```

Line 8 of this excerpt includes three variants of self-repair. The first is that RD *replaces* "argument with "base argument" after a brief pause. If we would just consider what RD says, we could conclude that he is *inserting* "base" before "argument". If we consider the code he types, however, it becomes clear that he is replacing the term "Argument" with "BaseArgument" (l. 9). The second verbal repair is that RD replaces "base arg-" with "argument from superiority", quickly replaced through a third repair "argument from (0.7) superio:rity: (0.7) efficacy" (l. 11). The second repair is not matched by typing as it is immediately followed by a third repair; the third repair is again matched by typing as RD types "FromSuperiorityEfficacy", thereby creating a name for a class called "BaseArgumentFromSuperiorityEfficacy". Both CG and PV are contributing to the name of the class. CG mentions "drug superiority" (l. 4), of which "Superiority" is included by RD (l. 11 & 12), while PV mentions "efficacy" (l. 10), which is also included by RD as "Efficacy".

A comparison of the code that is written and the words that are being said reveals that they are co-produced. Semantically, the written code matches some of the uttered words *exactly*, but the uttered words include additional *conversational noise* that does not find a counterpart in the code: repetitions, pauses, fillers such as "ehm", plus talk by other participants and talk by RD directed at other participants. The code-as-it-is-written retains traces of interactional dynamics as RD writes something, ponders over it, deletes it and replaces it with something else. But once he has named the class, the finished code no longer retains traces of editing, revealing how the practice of writing code "deletes the work" (Star 1995) that went into its production while it is being written. Specifically, the problems encountered and the alternatives pondered over while writing the code cannot be recovered from the code of a working prototype; such problems and alternatives must be documented elsewhere, and we know from methodological considerations of ethnographic writing that such documentations are always selective (Bergmann 2007; Hirschauer 2001).[1]

1 Personal experience suggests that documenting coding work is more difficult than documenting social interaction. During my own efforts in writing code, I found that I was unable to document each step taken and each problem encountered, although I was motivated to do so. As I was pondering over a problem, trying different ways to get code with the proper syntax that did what I wanted, this activity occupied me completely and any-

Although the excerpt is short, the requirement for exactness in writing code is indicated by the care that RD takes in naming the Java class. Participants are collaboratively searching for exactly the right name to express the functionality of the class for the program they are working on. What is given precedence over the verbal interaction among participants in this situation, however, is the formulation of working code that does not produce errors. In the following excerpt, the code that is written is described in italics and where existing code is amended the old code is gray.

#IV: The ArgumentFactory.java Class

```
01   RD:  .hhh SO my idea is that assume that this argument
          factory, (0.6) ehm: (2.4)
02        .h let's assume that it could work as follows, so
          we defin:e (1.2) we define a query, (1.5) SPARQL
          query, (0.4) okay? (1.1)
03        Starts typing "// Define a SPARQL Query"
04   PV:  Mhm (1.3)
05   RD:  So this says let's say ehm string (1.0) SPARQL,
          (2.8) query, (1.5) okay?
06        Types "String sparql,query;", continues to look at
          laptop
07        (1.4)
08   PV:  nods
09   RD:  .hh (1.) eh:m so what does this return? This m-
          this (0.3) class has something like (1.2) ((smacks
          tongue)) ehm:
10   PV:  Ehm:
11   RD:  Public (0.3) list,
12        RD types "public List"
13   PV:  (  ) list,
14   RD:  Of (0.4) base argument (1.5) arguments, (1.3)
15        Continues typing "public List<BaseArgument>
          arguments;", error message appears next to all
          lines of code
16        Okay, ehm get arguments. (0.7) arguments, but
          actually we could even do it like this query.
17        Types "public List<BaseArgument>
          getArguments(query);", error message remains next
          to all lines of code.
18        (10.9)
19        Deletes line "String sparql,query;" and then the
          comment line, error remains; RD continues to look
          at laptop the entire time.
20        Yeah hh. (2.8) hm hm hm hm hm yes, [°yes.°]
21                                           Raises index
          finger (in an "I-got-it" gesture)
22   CG:                                     [ And  ]the
          query is a SPARQL query?
23        Yes. Yeah yeah mhm, (0.3)
24   RD:  so this is a string query, (1.2)
```

thing I documented was a distraction, keeping me from finding a solution. Furthermore, after I found a solution, the solution often seemed obvious after the fact, and I quickly forgot exactly what the problem had been in the first place. The result was a rather eclectic documentation of my work. Although my experience as a lay programmer is not representative for professional programmers, it does suggest that only the most difficult problems may be remembered in retrospect, while small problems, problems that may nonetheless influence the work of coding, remain undocumented and cannot be recovered.

```
25        Adds "String" to "public list<BaseArgument>
          getArguments(String query);", (x) goes away for all
          lines except "public list...".
26        Then, why doesn't it work? (0.3) Because I have to
          impor[t list, okay,]
27        Clicks on the error symbol "x" next to code and a
          context menu opens,
28   PV:       [What   is    ]that string, do we take it from
          (0.3) the ontology? (1.2)
29        PV looks up in a "thinking" way; RD continues to
          look at laptop
30        (   ) can it be: (0.6) £well whatever he, can be
          another kind kind of query£ also,
31        PV looks at CG; RD selects the first option from
          the context menu, "public List<BaseArgument>
          getArguments(String query) {
          return null}; appears
32        (2.1)
33   PV:  for if we take it from (0.4) a database,
34        RD opens the context menu again, selects another
          option and at the beginning of the terminal "import
          java.util.List;" appears as code in the console and
          the error message goes away, RD continues to look
          at laptop
35   CG:  Mhm (yeah)
36   RD:  =Okay? (1.1)
37        RD continues to look at laptop
38   PV:  °(mhm)° (0.3)
39   CG:  °(  )° would this return a list of lists eh: list
          of lists?
40                                                  RD looks
          up from laptop again for the first time
41   RD:  No[. list, list]
          ((...))
```

In this long excerpt, RD describes the main purpose of the Java class "ArgumentFactory". Given an input SPARQL query, the main method of this class outputs a list of arguments which are described by the Java class type BaseArgument.[2] The exact specification of the method is not defined beforehand but rather elaborated while coding, a process not yet concluded at the end of the excerpt. In the process of the writing a first code draft of the class, RD produces several syntax errors, which are progressively eliminated, on which I will focus in the following:

At the beginning of the sequence, RD declares two variables, "sparql" and "query" of datatype String (ll. 5). This is how the console looks like at this point:

```
public class ArgumentFactory {

// Define a SPARQL Query

      String sparql,query;

}
```

2 Brown (2006) observes that the code that is being written is retrospectively and prospectively linked to other code. This can also be observed here: BaseArgument is another Java class of the same project; SPARQL is a query language that can be used to query semantic data bases; the type List that is later used is part of a standard Java library.

RD proceeds by creating a new variable of type List named “arguments” (ll. 12-15). Both the list of arguments and the query are initially not logically linked. For the computer scientists it is obvious that the list is to be populated by the query, but the computer must be told so explicitly. This is done in a conceptual correction of the code. RD rewrites the argument list as a method with a single parameter “query” (l. 17). The new written method “getArguments” with the query parameter makes both previously declared variables obsolete which is why they are deleted by RD together with the comment (l. 19). The conceptual correction results in the following code:

```
public class ArgumentFactory {

	public List<BaseArgument> getArguments(query);

}
```

This code, however, is erroneous, which the development environment Eclipse indicates by showing a red “x” next to the respective line of code. With “hm hm hm hm hm” (l. 20) RD is hearably concerned with the correction of this error but does not articulate to the other participants what his understanding of the problem is. He then finds a partial solution by specifying the datatype “String” for the “getArguments” method parameter “query” (l. 25) and then proceeds to solve the remaining errors (ll. 26).

Because of how RD defined the getArguments method, the compiler expects a return value and does not compile without one. Since the method is not fully specified at this point it does not yet return anything, which produces a syntax error. The addition of “return null” (l. 31) avoids the syntax error by stating explicitly that the method returns nothing. This demonstrates that the compiler requires completeness. At the same time, “return null” marks an intermediate stage of the programming work because a method that is supposed to return something and returns nothing clearly does not achieve the purposes for which it is being developed yet. The compiler, however, cares nothing about these purposes and cannot know them. The addition of “return null” allows the computer scientists to proceed nonetheless; they can work on other classes and test the code in order to discover whether the other code does what it is supposed to do.

In addition to the solutions of the errors already mentioned, the compiler requests a missing import to identify the class List, what RD proceeds to do (l. 34). The code is updated to:

```
import java.util.List;

public class ArgumentFactory {

	public List<BaseArgument> getArguments(String.query) {
		return null;
	}

}
```

While RD ponders over the syntax errors Eclipse indicates, he looks at his laptop and not the other participants. In fact, during the entire sequence, beginning in line 1 and ending in line 40, RD looks only at his laptop. While RD verbalizes what he is writing, he only verbalizes his search for a solution once (l. 26). He also checks the others’ understanding twice with “okay?” (ll. 2 and 36) but does not look up as they respond. Also, as PV asks a question without looking at anybody specifically at first (l. 28), RD does not respond, so that PV then directs her talk at CG (l. 30). This limited responsiveness on RD’s part indicates that the activity of collective code writing is here secondary to the solution of errors that the activity of code writing produces. Nothing strictly forces RD to correct the errors immediately, he could also interact with the other participants. That he nonetheless focuses on the errors’ solution points to the fact that erroneous code is nonworking code, which means that any error must be fixed eventually, so it may as well be done immediately.

Now I turn to excerpts from meetings where participants work on and write natural language text, both German and English. These excerpts reveal differences between writing code and writing text and indicate that writing does not need to be as exact as writing code.

1.2 A limited need for exactness in writing in natural language

#V: "but you know what I mean"

```
01   PV:  [mhm] Yeah, well (0.7) it's an initial idea °let me
          see if I find the document here° (1.0) (      )
          (2.9)(     ) (8.2) (some questions) (5.9) (for
          example) (2.0) Some questions could be which
          treatment was the most (efficacious) for people of
          these characteristics (1.0) who suffer this
          disease, (1.1)
02        PV opens a text document, then reads from screen
          from the section "Questions for recommendation" a
          bullet point: "Which treatment is the most
          efficacious for people of [characteristics age,
          country, ethnicity, etc.] who suffer [disease or
          problem] to [output_indicator e.g. decrease of
          IOP]?"
03   CG:  Mhm
04   PV:  Eh: problem to (0.7) this output indicator or (0.3)
          endpoint.
05   CG:  Mhm
06   PV:  Sorry. I have to: (0.6) change the- (2.0) well but
          you know what I mean °ha ha ha°
07   ND:  the the the the segmentation of the (0.6) patients
          it's it's important because I saw also in the other
          document that the the focus group I don't know they
          did patients are divide in using several different
          segment but variable segment.
08        While ND talks, PV deletes "output_indicator" and
          writes "endpoint" instead and safes the document
```

This excerpt is taken from another earlier meeting of the ME project, PV, CG, ND, and the ethnographer are participating. The purpose of this meeting is to represent information about clinical trials in the form of RDF triples, which are a machine-readable knowledge representation (RDF stands for resource description framework, it was conceptualized as a standard by the world wide web consortium, the previously mentioned SPARQL is a query language for RDF). Participants are currently discussing what knowledge they should be able to represent computationally, which they argue depends on the purpose of their project, a purpose they can elucidate by considering what questions their knowledge representation should be able to answer. PV is just now reading one such question from a document she drafted, which is also projected on a screen so that the others can read it as well. As PV is reading "problem to (0.7) this output indicator" (l. 4), she deviates from the text uses self-repair and includes "or endpoint" as an alternative to "output indicator". "Endpoint" is a term from the medical domain participants have been talking about previously as describing an outcome that can be measured. PV begins to state that "Sorry. I have to: (0.6) change the-", likely aiming for an expression such as [wording] as in "change the [wording]", stops and says "well but you know what I mean" instead.

The natural language document is here malleable to their previous discussion. PV herself points us to a potential reason why writing in a natural language does not require as much exactness as writing code. In writing something in a natural language, participants do not need to use exactly the grammatically and semantically correct expression because those who are to read the document, including PV herself, can anticipate what she most likely was trying to say at the time of writing; a human project participant can grasp that “ouput_indicator” in this case can be understood to mean “endpoint”, while a computer would have been unable to treat “output_indicator” and “endpoint” as having the same reference (without having been explicitly instructed to do so). While the meanings of natural language expressions vary over the course of a conversation (Liberman 2012), the meaning of expressions in code are fixed. That a text in natural language can include various errors and still be a useful and meaningful document to (human) participants is shown in the next excerpt, also taken from the ME project.

#VI: *“you will fix it”*

```
01  RD:  ((clicking)) okay impact ((clicking)) hhh (2.8) ich
         (0.9) kann mir vorstellen, (5.6)
02       Types "ich kann mir vorstellen, "
03  DZ:  ein solches Sys:tem in meinen täglich Arbeit als-
04  RD:  Yeah (0.8) solches Sytem (1.8) °(      ) solches
         System°
05       Types "ein solches System "
06  DZ:  Ein solches System, in meinen täglichen Arbeit-
         (0.5)
07  RD:  In meiner (0.7) täglichen (0.6) Arbeit (0.5) ein
         (1.0) zu:: °set° (0.4) zen um (0.7) Therapie (0.3)
         Entschei- (1.0) zu unter (0.7) stützen, (2.0) sorry
08       Types "in meiner taglichen Arbeit einzudetzen um
         Therapieentscheidungen yu unterstuetzxen?"
09  DZ:  Yeah
10  RD:  You-  you[ w]ill fix it.
11  PV:           [hm]
12       mhm ha ha
```

In excerpt #VI, participants are working on a survey for medical practitioners. At this point of the project, the group has a prototype that they want to test in a controlled fashion. They are planning to ask medical practitioners to test the prototype and the survey is to capture their impressions of it. The survey is formulated in German and the question they are currently working on translates as “I can imagine using such a system in my daily work to support decisions about medical therapies”. In line 7, we observe that RD says this sentence slowly in German, omitting the second part of “Entschei[dung]” (decision), but without any hearable grammatical mistakes. What he writes down at the same time, “in meiner t[!]aglichen Arbeit einzu[!]detzen um Therapieentscheidungen [!]yu unterstuetz[!]xen?” (l. 8) however, includes four grammatical mistakes. In contrast with excerpt #IV, it is noteworthy that RD does not correct these errors, although he is aware of at least some of them. Instead, he suggests that DZ and PV correct these mistakes at a later point in time (l. 10). The document RD is working on has the status of a draft and for the time being it is acceptable if it includes grammatical mistakes because the others are able to understand what RD meant to write and can therefore continue to work with this draft. During this meeting, participants also change the order of single questions or sets of questions several times, a practice that would be much more problematic if they were working on code.

In the paper “Managing exactness and vagueness in computer science work”, I specify four rules that programmers needed to adhere to when writing code. I argued that appropriate instructions need to (1) use the proper formatting and punctuation of (2) the semantically correct expressions (3) in the required order, (4) forming a complete program. The two excerpts #V and #IV show that these rules can be easily violated while writing in a natural language. Excerpt #V shows a violation of rule 2: PV wrote down a semantically incorrect expression with "output" but counts on the other participants to understand that she actually means "endpoint" in this case. Excerpt #VI shows a violation of rule 1 as RD makes several grammatical errors. In this meeting, participants also violate the remaining two rules: They violate rule 3 by changing the order of the questions several times, and participants conclude the meeting without having a final set of questions without errors, thereby violating rule 4. Of course, programmers will frequently also violate one of these rules as they work, but this turns their code into something of no utility until the rule-violations are corrected. While a natural language draft has utility because it expresses ideas such as an important concept or the potential structure of a finished document, a code stub that violates one of the four rules cannot be compiled or interpreted and therefore not tested as to whether it does what it is supposed to do. Participants working on the survey, on the other hand, have more certainty as to their achievements. They can agree at the end of the meeting that they have now a useful set of questions in a sensible order and the fact that their draft includes grammatical mistakes does not jeopardize their agreement or the work undertaken to reach it.

2 Variations of exactness

The next excerpt in section 2.1 complements the analysis of hedges in the paper "Managing exactness and vagueness in computer science work". The excerpt in section 2.2 shows how participants distinguish between design choices and implementations necessitated by a model.

2.1 Reaching an approximate understanding through hedges

The next excerpt is from the first regular meeting of the "interaction simulation project" (IS project) I attended as an ethnographer. Excerpt #VII is part of the long excerpt #I that can be found in Appendix II of the paper "Managing exactness and vagueness in computer science work". In the main text of the paper, I consider line 38 and lines 68 to 74 from this meeting where participants discuss potential problems of how they implemented a psychological model, the PAD model. Here, I also consider the beginning of the meeting, where participants explain to me for the first time what this PAD model is. This provides another example of how understandings are negotiated among participants and how approximate understandings of the prototype's functionality are indicated by hedges.

#VII: "like a point and click adventure"

```
01   RD:  PAD space is (0.5) just for ((ethnographer, ET)).

02   ST:  Yeah, is that three dimensional spac:e,
          ((clicking)) °e:hm (           )° ((clicking))
          (1.1) ((clicking)) (1.1)

03   RD:  .h so the point where we are is that, if you are eh
          creating a tutoring system like this, (1.8) every
          message that (1.3) is (0.8) so basically the way
          this works is that the person (0.3) the user, (0.7)
          the learner, (0.7) can choose from a number of
          (0.5) predefined options. (1.0)

04        So, (0.3) use a number of options to give feedback,
          (0.7) .hh and what we're after is that (0.9)

05   ET:  this is like a point click adventure?

06   RD:  in some sense,
07  ET:  °yeah°
08  RD:  you could say so,
09  ET:  °hm°
10  RD:  I mean (   -) there is going to be speech (0.5)
         processing but in order to control a bit (0.4) we
         want to give some options that people can (0.5)
         follow.
11  ET:  Mhm (0.3)
12  RD:  can choose to follow, (0.9) .h a:nd the idea is
         that depending on what they choose (0.5) this has:
         an impact, (1.1) on the person that ge- that gets
         this (0.3) message. (1.3)
13       So this can increase (0.5) the mood positively,
         ((...))
```

Before the beginning of this excerpt, ST has provided a summary of the changes he implemented. CG then asked whether they are still using the PAD model, at which point RD proposes that they should explain to the ethnographer what "PAD space is" (l. 1) before continuing. ST replies "yeah, is that three dimensional spac:e", and then begins to search for a visual representation of the PAD space on his computer. RD takes this opportunity to describe the current stage of the project himself. He begins to say something about "every message that (1.3) is" (l. 3), stops and then begins with a different line of reasoning, a form of self-repair Schegloff (2013) calls "aborting" (p. 52). While RD may have been aiming to say something about the emotional impact of messages, this is an example of self-repair where we cannot infer with confidence what RD might have been aiming to say, so it cannot be used to juxtapose expressions, but the repair is nonetheless important because RD uses a hedge as he continues, "so basically the way this works" (l. 3), and explains that the player has a number of conversational options to choose from. The hedge "basically" literally tells me that RD is giving me the basics of how the interaction simulation works. This provides me with a first approximate understanding of the prototype, so that I can (hopefully) gain a better understanding of the more technical aspects of their work that they are going to discuss next. During one of the several pauses RD makes, I interject and provide a candidate understanding, asking whether this is "like a point and click adventure" (l. 5), referring to old adventure games that also provided the player with different conversational options. RD responds with a hedge, "in some sense" (l. 6). Again, the hedge informs me that I have reached an approximate understanding of the prototype. While the availability of different options is similar, the interaction simulation is also different in other ways. One technological difference is that there is going to be "speech (0.5) processing" (l. 10) in future versions of the prototype. A similarity is that the computer scientists also require some control over the dialogue as is necessary in games, which is why the player is restricted to a set of conversational options that have been predefined. Another difference is the scientific foundation of the interaction simulation. The virtual character's emotional response is affected by the emotional stimulus linked to the option and controlled by a model that simulates mood and emotion called the PAD space.[3]

3 In adventure games, choosing different conversational options often does not have a significant impact on the game. By contrast, RD highlights the importance of choices in their interaction simulation by using self-repair: "we want to give some options that people can (0.5) follow" (l. 10), then inserts "can choose to follow" (l. 12), stressing the importance of choice for their simulation, which follows the assumption that different ways of saying something are also meaningful in real conversations.

2.2 Distinguishing between choice and necessity in implementation

This section considers two cases where speakers clarify through self-repair (self-repairs that do not correct mistakes but alter meanings) that what they implemented was based on a personal choice and not on assumptions derived from a model. Such repairs indicate to the other participants that this particular aspect of a program prototype is not firmly based on other models but also that it can be more easily changed.

#VIII: “with respect to one dimension” (expanded excerpt #0 from the paper)

```
01  RD:  Ahm (1.6) okay. (0.6) That'[s it.]
02  PV:                             [And t]his is
         considering the the weights: already. (0.3)
03  RD:  This is considering the weights, yes. (1.1)
04  PV:  That they sa::[y (w]e          efficacies)
         [ehm-]
05  RD:                 [but-]
         [Yes,] yes, but ehm: (0.7) but this relies, on
06  DZ:  ((clears throat))
07       (1.2)
08  PV:  Hm
09  RD:  For now, (0.4) relies on a very simple aggregation
         scheme, (0.5) okay? (0.5) .h so the aggregation
         scheme ((click)) says- so the arguments down here,
         (0.8) yeah?
10  PV:  Mh[m]
11  RD:    [T]hey are- they they return either zero or one.
         (3.0)
12  PV:  [Mhm],
13  DZ:  [Mhm]
14  RD:  Because in a particular study, (0.5) it has been
         shown that it's superior or not. (0.5)
15  PV:  yeah, (0.2)
16  RD:  So one means has been shown superior, and zero
         would mean the negation, so has not been shown
         superior. (0.6)
17  PV:  Mhm, yeah, (0.3)
18  RD:  With respect to some dime[- on]e dimension, it's
         always with respect to one dimension.
19  PV:                           [what]
20  DZ:  Mhm.
21  PV:  Mhm,
22  RD:  (and) this is- this is- my intuition is that this
         is binary, (1.6) so either the study has shown
         (0.3) that Timolol is better at reducing IOP than
         the other one, or not.
23  PV:  Mhm
24  RD:  Okay? (0.3) So at the lowest level this binary.
```

```
(1.8)
```

This is an extended version of excerpt #0 from the paper "Managing exactness and vagueness in computer science work". Members of the ME project talk about a prototype that RD has written in Java and now presents to PV and DZ. The self-repair of "some dime[nsion]" to "one dimension" (l. 18) has already been considered in the paper. Another self-repair follows in line 22. Here, RD starts to say "this is", likely aiming to say "this is [binary]", but then self-repairs to say "my intuition is that this is binary" instead. The program prototype that RD wrote to aggregate medical evidence assumes that a single medical study either provides evidence that a drug or treatment is superior or that it is not; the prototype does not consider the size of effect of results, their statistical significance or any other aspect that may qualify the superiority of a treatment. By adding "intuition", RD makes it clear to the other participants that what he implemented does not follow a scientific model or established technical procedure, it was a personal choice that somehow feels right. As the meeting continues, this choice is questioned for its adequacy, to which RD responds by claiming that they have to make assumptions of this kind in order to be able to implement anything.

The next excerpt introduces another distinction between design choice and necessity in the computational implementation of a scientific model such as the PAD model.

#IX: "it wasn't really supposed to be like that"

```
01   RD:   takes quite a long to- to show (an reaction)
          right, (1.1)
02   ST:  eh it it was supposed to be like that, the
          movements e:h on the emotion axis, (0.5) ehm (0.4)
          well it wasn't really supposed to be like that, I
          (0.5) defined what will be the extent of the
          effects on emotion, currently cause the scale is
          from minus one to one, currently I'm increasing it
          maximum by zero point two. This is up to us to
          (0.2) increase or decrease it to a higher extent.
```

RD observes here that the virtual character does not respond quickly to the emotional stimuli linked to conversational options selected by ST as he tests the simulation. ST replies that "it was supposed to be like that, the movements e:h on the emotion axis" and then self-repairs and states "well it wasn't really supposed to be like that, I (0.5) defined what will be the extent of the effects on emotion", before explaining the exact numerical effect of each conversational option. ST thereby makes a distinction between behavior of the prototype that is a direct consequence of the model's original design and behavior that is a consequence of design choices for aspects not dictated by the model; the model dictates that there is a positive or negative stimulus, but not the size of its effect. Exploiting this distinction allows participants to remain faithful to the scientific model while at the same time manipulating the surface behavior caused by it. In this case, RD later proposes to increase the size of effect of emotional stimuli, so that the virtual character changes its emotional state more quickly.

3 Contrasting cases: Other forms of self-repair that alters talk

The next two excerpts are taken from a meeting of the ME project. In both cases, participants are using self-repair that alters the talk (the repair does not respond to grammatical or other mistakes). They use self-repair in a sequential context where they are *not* talking about the existing functionality of a prototype and the excerpts show that the repair operations do *not* introduce a more exact expression in this context, supporting Proposition A of the paper “Managing exactness and vagueness in computer science work”.

#X: “there’s no substance”

```
01   RD:  So we need to formalize this. (1.1)

02   PV:  Yeah, this is what- eh[::]

03   RD:                        [Th]en- then- then people can
          [still say: it's crap, it's crappy, or we-]

04   PV:  [its called preferences and benefits.     ]

05   RD:  It's crappy, we don't like it, or whatever, .h

06   CG:  yeah

07        but I mean now they are criticizing that (0.2) ((PV
          clear throat)) there's no substance, (1.3) or is
          not- is [not] described, the substance is not
          described.

08   PV:          [mhm]

09   CG:  Mhm

10   PV:  [Yeah (     )]

11   RD:  [Which is- wh]ich is fine, it's not described,
          so: .hhh so we need to described it properly
          ((...))
```

#XI: “he’s also very...”

```
01   RD:  Yeah. And (0.3) he is interested in continuing this
          to some extent, and ehm (0.9) I think he's also
          very:: (0.8) he- he has a very good eh: training in
          theoretical computer science (0.6)

02   PV:  Mhm

03   RD:  formal grammars, automata, whatever, ahm so: (0.5)
          I think he could help a lot with the:: (0.3)

04   PV:  The framework.

05   RD:  Yeah. (1.1)
```

In excerpt #X, participants are talking about the reviews they received for one of the project-related papers they submitted and how they should continue their project. RD argues here that they need to “formalize” (l. 1) their approach. In line 7, he begins to say “they are criticizing that ((...)) there's no substance”, then replaces this with “the substance is not described”. This changes the meaning from a statement of facts – that their ME project has no substance – to the observation that the substance of the project is not described in their paper, allowing for the possibility that their project has in fact substance.

In excerpt #XI, taken from a later point in the same meeting, RD is talking about a potential new team member who could support the ME project by working on one of their topics of interest. RD begins to say “I think he's also very: [good]” and replaces this statement with “he has a very good a:h training in theoretical computer science”. Instead of making the general claim that the addition to their team is “very good”, RD limits his praise to the training this person received, perhaps because he can give this more indirect praise with more confidence.

Both cases of self-repair that alter meanings can be said to be more adequate than the expressions they replace, but they cannot be said to be more exact. In both cases, RD withdraws a factual statement and thereby introduces possibilities of interpretation. These possibilities become visible through comparison of the repairable with the repaired statement. In excerpt #X, the repair introduces the possibility that their project does have substance and in excerpt #XI, the repair introduces the possibility that the new team member may not be as good as his training. Together with excerpts #5 and #6 in the paper, excerpt #X and #XI support Proposition C: Self-repair that alters expressions in contexts where participants do not talk about the existing functionality of a prototype does not *limit* the space of possible meanings as a comparatively more exact expression does, it instead *opens* the space of possible meanings in situations where participants feel they have limited it prematurely or inadequately.

References

Bergmann JR (2007) Flüchtigkeit und methodische Fixierung sozialer Wirklichkeit: Aufzeichnungen als Daten der interpretativen Soziologie. In: Hausendorf H (Ed.) *Gespräch als Prozess*, Tübingen: Gunter Narr, pp. 33-66.

Brown B (2006) *'The next line': understanding programmers' work*, TeamEthno 2: 25-33.

Hirschauer S (2001) *Ethnografisches Schreiben und die Schweigsamkeit des Sozialen: Zu einer Methodologie der Beschreibung*, Zeitschrift für Soziologie 30 (6): 429-451.

Liberman K (2012) *Semantic Drift in Conversations*, Human Studies 35 (2): 263-277.

Star, SL (1995) The Politics of Formal Representations: Wizards, Gurus, and Organizational Complexity. In: Star SL (Ed.), *Ecologies of Knowledge. Work and Politics in Science and Technology*, New York: State University Press, pp. 88-118.