What happens during Pair Programming?

Stephan Salinger and Lutz Prechelt

Freie Universität Berlin, Institut für Informatik, Takustr. 9, 14195 Berlin, Germany
salinger,prechelt@inf.fu-berlin.de

Abstract. Successful qualitative analysis of pair programming requires a terminology (such as a set of concepts or a coding scheme) that represents the observed phenomena on an appropriate abstraction level. On the one hand, different analysis goals will require different specialized terminology, on the other hand it would be helpful, if different studies used common terminology so that comparing and combining their results will be easier. We suggest to define terminology in layers: a PP foundation layer that is common to all analyses and more specialized study-specific layers on top. The present article presents this foundation layer which we have derived from audio/video analysis of pair programming sessions in Grounded Theory investigation style. Its concepts describe the individual observable human-to-human utterances and human-to-computer activities that occur during pair programming.

1 Introduction

Pair Programming (PP) is a practice in which two programmers write a program cooperatively, both using the same computer together. In the context of agile development methods, in particular eXtreme Programming (XP, [2]), PP has raised considerable interest and a number of empirical studies on this topic have been performed. So far, most of these studies [7, 10, 12, 13, 15–20, 26–28] use quantitative research methods that treat the PP process as a black box [23]. Such approaches produce some numbers but are inherently unable to really explain how these numbers came to be, because a theory of what is going on during PP is not yet available.

The first step to such a theory would be a qualitative description of the behavioral elements which together compose the PP process. However, the (few) qualitative (or mixed qualitative-quantitative) studies that exist on PP [3, 4, 6, 24, 29] all study individual aspects of the process only, none has yet attempted to present the process elements as such comprehensively.

We believe that a comprehensive, generic, and generally accepted description of the process elements will be needed to make multiple studies comparable and complementary, and hence to obtain deeper insights. We thus aim at producing such a description and will view each process element as a somewhat abstract concept to be characterized and delineated from the others.

In order to make the description generic and generally accepted, we have neither used any of the specialized coding schemes from existing studies of PP [3, 4, 29] or related topics as a starting point, nor have we adopted any other existing descriptive framework (say, a model of cooperation). We feel that too little is known about the PP process to make sure that any such choice would be a sufficiently good fit with PP. Instead we use the most open-minded approach we know of, namely Grounded Theory (GT, [9]), to derive the concepts directly from our raw observations (audio, video and desktop recordings of actual PP sessions) in a bottom-up fashion. For this reason we will devote the space we could have used to discuss related work to a more detailed description of the concepts themselves.

A few global remarks before we start:

1. We believe the investigation of the PP process should proceed in so-called layers. Each layer uses a different perspective on the PP process and is investigated in separate studies. The perspective describes in which respect (say, knowledge transfer in PP, abstraction levels used by participants, PP decision-making, etc.) we are looking for insights, how to cope with subjectivity during the analysis, and what form of result we are aiming at (such as a set of concepts, a coding scheme, a theory) [23].
2. The first layer to be developed, called the *foundation layer*, is the fundament on which all other layers will build. The concepts in the foundation layer, called the *base concept set*, describe the basic activities that occur during a PP process. The base concept set comprises concepts describing directly observable communication events, activities pertaining to the computer, and activities pertaining to the rest of the work environment. The concepts represent neutral description only, no measuring, grading, rating, or other evaluation. Evaluative concepts (*properties* in the language of GT) will be introduced in other layers only.

3. Any other layer will extend (and sometimes also simplify) the base concept set as needed by its perspective. It may add completely new concepts or may differentiate existing ones.

The present article gives an *introduction* to the base concept set only, as many details will have to be left out. We will provide a comprehensive definition of the base concept set in a detailed technical report a few months from now.

It is the purpose of the foundation layer to supply a basic unified terminology with which qualitative studies of PP can talk about their subject so that it becomes easier to relate various studies to one another and to build new studies on top of previous ones.

We will now give a quick introduction to the somewhat customized GT research style we used, shortly explain the nature of the raw data from which we derived our results, and then present the base concept set itself.

## 2 Our Grounded Theory methodology

Grounded Theory (GT) is the research method of choice whenever wanting to start from as few prior assumptions as possible. From the two different styles of GT, we picked the more rigorous variant according to Strauss and Corbin [25], which prescribes to represent the data by means of conceptual (rather than descriptive) codes that are developed directly from phenomena observed in the data (“open coding”), and to investigate relationships between those concepts such as cause/effect or context/element in order to gain a deeper understanding (“axial coding”).

As a minimal requirement for constituting GT work, codes must be explanatory concepts (“theoretical coding”) rather than merely descriptive labels, and observed phenomena must be re-viewed and compared again and again (“constant comparison”) in order to create codes that are highly consistent and fit the data closely [11]. As a result, the codes effectively emerge from (and are thus firmly grounded in) the data.

Our first attempt at applying GT to our PP data failed miserably: We constantly lost focus and drowned in the data. As a remedy, we amended GT by four guiding practices [23]: *perspective* (as described in Section 1), *syntax rule for concept names*, an *analysis metamodel*, and *pair coding*.

The concept name syntax rule is as follows:

```plaintext
conceptname = <actor>.<description>
actor = P1 | P2
description = <verb>_<object>
```

P1 and P2 are the two members of the programmer pair. This part of the name is irrelevant for the concept as such (and will usually be left out in the discussion below), but is an important aid for the analysis process. Verbs and objects will be introduced in Section 4.

The metamodel represents and relates the elements of a GT analysis as a UML class diagram and explains the structure of the analysis results. For developing the base concept set, we used the following metamodel elements: A *Quotation* is a scene in the raw audio/video data, an *Annotation* relates a *Quotation* to a *Concept*. *ConceptClasses* hierarchically group related *Concepts*; some of them may later become GT *categories*. *ConceptRelations* describe certain relationships between *Concepts*. 
**Pair coding** means that all coding work is performed by two people together, much like pair programming, in order to avoid distortions arising from the limited or biased perception of any single individual. We used pair coding to derive the first version of the base concept set; later refinements and polishing were performed by one person alone.

3 Underlying data

The data from which we derived the base concept set consists of recordings of complete PP sessions that contain audio and video of the developers’ conversations and pixel-precise video of the computer desktop [23].

We analyzed data from three rather different such sessions. Session 1 (of length 2:58 hours) features a laboratory session of two German graduate students who had worked together as a pair several times before. They built a small extension to a cleanly designed Java EE web shop system with which they were modestly familiar. The main task difficulty lay in the need to apply certain Java EE technologies (JMS, JNDI, JBoss application server) that the developers had not applied often beforehand.

Session 2 (of length 1:47 hours) features a field session of two professional programmers who worked for the operator of a very large German community portal and also had worked together as a pair several times before. They built an extension to the community portal, which is implemented in PHP. The task difficulty had several aspects including understanding the design (and design rationale) of the pre-existing code, which had been written by offshore programmers.

Session 3 (of length 1:14 hours) features a field session of professional programmers from a mid-sized company specializing in geo-information systems. They worked (in Java) on the visualization of attributes. One important aspect of their task (though this aspect was never spelled out explicitly) was to investigate the amount of refactoring needed and also to perform refactoring. The task had been estimated to require a full day and was not finished during the session. Compared to the other two pairs, these programmers reported to be less well acquainted with each other as a pair.

We annotated these data directly (without transcription) in the ATLAS.ti [1] data analysis software; Session 1 using pair coding, Session 2 and 3 alone. Overall, 2826 instances of codes were annotated to the sessions at 2443 different scenes.

4 The Base Concept Set

The base concept set is an extensible framework (and not a fixed coding scheme) for classifying the basic activities that occur during PP. The base concept set should not be assumed to be fully complete, although we have observed a certain degree of theoretical saturation: once the first PP session had been coded, only two new concept classes arose during the second session and only one new concept class during the third, although those sessions came from widely different PP contexts. Nevertheless, further observations from still very different PP contexts may perhaps require the addition of further concepts.

We aimed at a set of concepts that is large enough to be quite differentiated, yet small enough to be used consistently. The choice of (and strict adherence to) perspective and the use of pair coding were instrumental in achieving this goal and also assured the concept set is highly coherent. However, the granularity of the base concept set is not at all canonical. It could as well be defined in a more detailed manner with more concepts or somewhat coarser with fewer concepts. Besides the perspective used, our main guideline for choice of granularity was obtaining a uniform structure by using similar discriminations in different areas of the concept set where possible. In favor of a uniform structure, we left many possible refinements to further layers to be added later.
Under this analysis regime, a rather regular structure for the concept set emerged from the data. We will use this structure to guide the presentation of the concepts below.

4.1 Coarse partitioning of the base concept set

Early in the analysis process we introduced the following two fundamental concept classes. The first class, HHI (human-human interaction), includes all concepts that describe interactions between the pair members. In the case of the base concept set, these are only concepts for categorizing verbal utterances.

The other class consists of two parts: HCI (human-computer interaction) concepts classify activities related to the computer and requiring the use of keyboard or mouse, while HEI (human-environment interaction) concepts comprise all other activities relating to the work environment. This means that HCI activities can be performed only by the PP driver [27], while HEI activities can be performed by both driver or observer.

Notably missing from the base concept set are any concepts relating to purely internal thought processes. Since such processes are not directly observable, they can, at least in a GT approach, only be introduced indirectly – which means they belong to layers to be created later.

We will now first describe the HHI concepts and then the somewhat less important HCI and HEI concepts.

4.2 Elements of the HHI concepts

As mentioned in the concept name syntax rule above, a concept name consists of a verb and an object. For a first impression of the HHI concepts, this section introduces these verbs (in Table 2) and objects (in Table 1) as the elements composing HHI concepts.

By far not all combinations of a verb and an object from the tables occur in the base concept set, for the following reasons:

1. Not all combinations make sense. For instance, it is impossible to propose an activity, because activity always refers to current events, not to future ones.
2. Some combinations would create concepts that overlap semantically with others. For instance, there is no concept explain_step because knowledge transfer of all kinds (including rationales) is always to be represented by the knowledge object.
3. A few verbs are explicitly constrained to one single object only. For instance, say is meant to be used with off topic only, as it would be too unspecific otherwise.
4. Many combinations, while semantically possible and even plausible, are missing simply because we have never seen them in our data; for example disagree_strategy. It is possible to add such concepts immediately when they occur.

There are two cases where delineations between several verbs or several objects are particularly important as described in the following two subsections.

**explain vs. think aloud** We initially tried to discriminate undirected communication (verb think aloud) from directed communication (verb explain), but overly many ambiguous cases make this discrimination impractical. The definition we finally adopted for the base concept set requires that think aloud can be used only when it happens in the context of a concurrent and ongoing HCI or HEI activity of the speaking person. Thus, think aloud suggests that the speaker is attempting to keep the pair partner informed about the meaning or rationale of the current activity. In contrast, explain is used when communicating a circumscribed issue, for instance a new insight.

This decision has far-reaching consequences for the structure of the base concept set: There is only a single think aloud concept, think aloud_activity\(^1\), whose rather special role will be described in Section 4.3.

\(^1\) This is in contrast to the state of the concept set described in [23].
<table>
<thead>
<tr>
<th>object</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity</td>
<td>An HCI or HEI activity that is currently ongoing.</td>
</tr>
<tr>
<td>completion</td>
<td>Degree of completion of a tactical (basic) work step. Contrast with state.</td>
</tr>
<tr>
<td>design</td>
<td>An aspect or element of the possible structure of the program being written.</td>
</tr>
<tr>
<td>finding</td>
<td>An insight that one pair member has just had and verbalized. Indicator of a knowledge gain by means of thinking. Contrast with knowledge and standard of knowledge.</td>
</tr>
<tr>
<td>gap in knowledge</td>
<td>The fact that certain knowledge is lacking in the pair (as opposed to just one member of the pair). Contrast with standard of knowledge.</td>
</tr>
<tr>
<td>hypothesis</td>
<td>A hypothesis or conjecture, typically regarding a property of the program, its requirements, or the technology or environment.</td>
</tr>
<tr>
<td>knowledge</td>
<td>Explicit declarative knowledge that is neither meta-knowledge (see standard of knowledge and gap in knowledge) nor a new insight (see finding).</td>
</tr>
<tr>
<td>off topic</td>
<td>Anything not directly related to the solution process sought.</td>
</tr>
<tr>
<td>requirement</td>
<td>An actual or assumed requirement of the pair’s task.</td>
</tr>
<tr>
<td>source of information</td>
<td>...such as source codes, documentation, web pages, etc.</td>
</tr>
<tr>
<td>standard of knowledge</td>
<td>An assessment of the level of knowledge regarding a certain topic that is present in one particular member of the pair. Not to be confused with gap in knowledge which concerns the pair as a whole.</td>
</tr>
<tr>
<td>state</td>
<td>Degree to which a strategy has been worked through. Contrast with completion.</td>
</tr>
<tr>
<td>step</td>
<td>A possible next step in the work process; viewed by the actor as an atomic unit of tactical behavior. Contrast with strategy and todo.</td>
</tr>
<tr>
<td>strategy</td>
<td>A possible approach or work plan for solving some non-trivial (sub)problem. Strategies always involve multiple steps.</td>
</tr>
<tr>
<td>todo</td>
<td>A subtask or work item that will have to be completed in the future but cannot or will not be completed right now. “future” may refer to the current or a subsequent pair programming session.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>verb</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amend</td>
<td>Add what the speaker considers to be a relevant extension to an utterance or activity. Implies basic approval of the utterance or activity.</td>
</tr>
<tr>
<td>ask</td>
<td>Ask a (usually open but sometimes closed) question.</td>
</tr>
<tr>
<td>agree</td>
<td>State approval with an utterance or activity. Contrast with decide.</td>
</tr>
<tr>
<td>challenge</td>
<td>State disapproval with an utterance or activity and make a counter-suggestion. Contrast with disagree.</td>
</tr>
<tr>
<td>decide</td>
<td>Select one from a set of multiple explicitly proposed options. Contrast with agree.</td>
</tr>
<tr>
<td>disagree</td>
<td>State disapproval with an utterance or activity without making a counter-suggestion. Contrast with challenge.</td>
</tr>
<tr>
<td>explain</td>
<td>Provide an explanation directed to the other pair member.</td>
</tr>
<tr>
<td>propose</td>
<td>Make an individual suggestion (see agree, challenge, disagree) or suggest a couple of alternative options (see decide, challenge, disagree).</td>
</tr>
<tr>
<td>remember</td>
<td>Observably remember a specific fact.</td>
</tr>
<tr>
<td>say</td>
<td>Say something (only used together with the object off topic).</td>
</tr>
<tr>
<td>stop</td>
<td>Propose stopping or aborting an activity.</td>
</tr>
<tr>
<td>think aloud</td>
<td>Verbalize one’s own current activity and related thoughts.</td>
</tr>
</tbody>
</table>
knowledge vs. finding vs. standard of knowledge Knowledge transfer is one of the most-discussed aspects of PP. The foundation layer’s representation of this aspect can be summarized as follows.

1. Due to observability limitations we consider only explicit (declarative) knowledge [21].
2. We do discriminate between one particular kind of meta-knowledge (standard of knowledge) and other knowledge (knowledge). The former refers to knowledge about the presence or lack of certain knowledge, which often determines degrees of freedom for the subsequent PP process.
3. We do discriminate between settled knowledge (knowledge) and freshly acquired knowledge in the form of sudden insights (finding).

Note that these objects and concepts emerged from the data like all others; they were not introduced a priori.

4.3 HHI concepts

We have found five subclasses of HHI concepts: product-oriented, process-oriented, generic, facades, and other. Figure 1 provides an overview; further details will be explained below.

Product-oriented concepts There are two kinds of product-oriented concepts. The design concepts refer to the possible implementation structures of the program to be written (design decisions, both high-level and low-level). This ranges from naming of variables to implementation of procedures to configuration settings to interfaces of subsystems. In contrast, the requirement concepts refer to the functional and nonfunctional properties the program to be written needs to exhibit and to other external constraints is has to obey.

For the design concepts, HHI events we have observed were propose, agree, decide, disagree, amend, challenge, and ask. Dependencies among these regulate possible event orders; similar kinds of dialog sequence dependencies exist for many other concept classes as well.

Starting point is a propose_design, possibly (but not necessarily) followed by one or more of the other kinds of design event which accept, reject, or complement the proposal or a part of it. These subsequent events can be produced by either pair member, not always in alternating order. Sometimes such a sequence begins by ask_design instead, which is an open question without a proposal.

Example: we observed that one programmer said “We can pull out the not.” (propose_design) and the partner replied “No, I would last_change if that is larger than last_request, return something, else return exit.”. (challenge_design).

For the requirement concepts, HHI events we have observed were propose, agree, challenge, and remember. remember_requirement means a programmer reminds the pair of a given requirement, while the other concepts indicate the pair works on clarifying a vague or ambiguous requirement, which is very common in the XP contexts [2] for which PP is most typical.

Process-oriented concepts There are five classes of process-oriented concepts that concern strategy (strategy and state), tactical behavior (step and completion) and postponed work (todo).

step concepts concern utterances regarding potential immediate next work steps that the speaker apparently considers atomic, such as running a test, reviewing a section of code, or discussing an issue. We found the same verbs with step that we saw with design.

completion is related to step and refers to utterances regarding the degree of completion of the presently ongoing step. Verbs seen in this context were explain, agree, and challenge. completion events can occur even if no corresponding propose_step has ever happened; it is sufficient that the pair performs actions that could have been started by a propose_step.
Fig. 1. HCI concepts. The process, explain, challenge, agree, and disagree concepts classify statements about other statements, while the remainders classify process-oriented concepts.
Utterances regarding longer-lasting, pre-planned, multi-step action are described by strategy concepts. Verbs seen in this context were propose, agree, decide, amend, challenge, and ask. Utterances regarding the degree of completion while working off a strategy are described by state concepts, for which we have seen the verbs explain and agree. Again, an explicit propose_strategy is not strictly needed.

Todo concepts concern utterances that talk about postponing a certain work item until later in the same session or a future session; in our sessions this always concerned steps. So far we have seen only the verbs propose and agree in such contexts.

Obviously, several other verbs could sensibly occur in any of these classes and users of the base concept set should add the respective concepts when needed.

**Generic concepts** Generic concepts describe knowledge-related issues and occur in process-related as well as product-related contexts. There are four classes of generic concepts and three individual cases.

Three of the four classes (knowledge, finding, standard of knowledge) have already been introduced in Section 4.2 above.

For concept class knowledge we found five verbs, namely explain, agree, challenge, disagree, and ask. The core concept is explain_knowledge, which describes that knowledge (that is assumed to be correct) is being transferred from one partner to the other, possibly (but not necessarily) in response to a query (ask_knowledge) and possibly (but not necessarily) followed by utterances expressing evaluation. There is no concept amend_knowledge because we found that discriminating amendments from separate knowledge is often too difficult.

Concept class finding refers to sudden new insights, such as finally having located a long-searched-for defect or, more trivially, arriving at the understanding that something has worked (or not worked) as intended. Verbs found with finding are explain (core concept), agree, challenge, and amend.

Concept class standard of knowledge describes discussions of the level of available knowledge on a certain topic. ask_standard of knowledge queries the partner regarding how much knowledge is available. Example: “And you have never worked on this skript before?” explain_standard of knowledge informs the partner regarding how much knowledge is available. Example: “Can’t remember where our search has found something.” explain_standard of knowledge may look almost exactly like explain_knowledge when somebody rephrases a partner’s explanation of something to make sure s/he has achieved the intended level of understanding.

The final concept class, hypothesis, refers to verbalizations of conjectures and guesses. These can be explanations of program behavior, interpretations of design elements, etc. Example: “Oh, maybe we need to enable the chat?” (propose_hypothesis). Verbs seen in this context were propose, agree, challenge, disagree, and amend.

Now to the individual cases: explain_gap in knowledge refers to utterances concerning a recognized lack of knowledge of the pair as a whole. remember_source of information refers to utterances pointing to (or pointing out the existence of) certain sources of information such as documents. Both of these concepts occurred only rarely and could have been considered special cases of explain_standard of knowledge and explain_knowledge, respectively. We have included them in the base concept set nevertheless because we assume that the moments when they occur are of particular interest for understanding PP processes.

A rather special case is the concept agree_activity, which refers to a consenting utterance with respect to an ongoing HCI or HEI activity. This is the only member of concept class activity that is not a facade concept.

**Facade concepts** In rather loose reference to the Facade design pattern [8], a facade concept is a concept that provides a simplified view of some more detailed internal structure. activity is
the only class of facade concepts in the base concept set. These concepts describe verbal utterances connected to HCI or HEI activities. The most important one is think aloud activity (as mentioned in Section 4.2). It describes that an actor verbalizes aspects of his or her concurrent HCI or HEI activities, such as:

- What am I doing? Why?
- How am I doing it? Why?
- What decisions am I making? Why?
- What insights do I have while doing it?

This means that a phenomenon annotated with think aloud activity will often contain one or several subphenomena of the type propose design or propose hypothesis, etc. The instance of think aloud activity acts as a facade that bundles these subphenomena into a whole.

While think aloud activity relates to one’s own activity, the concepts amend activity, challenge activity and disagree activity relate to HCI and HEI activities (not their verbalizations!) of the partner. Such phenomena may be utterances such as “remember to set lastRequestTime!”, which would be annotated as amend activity plus propose design.

stop activity signifies a proposal to stop or abort an HCI or HEI activity, which we felt may be of sufficient interest to warrant a separate concept.

Other concepts In order to cover all verbal utterances we had to introduce two extra codes: mumble sth classifies an utterance as acoustically incomprehensible or overly fragmentary; say off topic classifies something as not having to do with the PP process proper.

Example For illustrating the use of the concepts, Table 3 shows the encoding of a short episode from session 3.

One can easily see why we do not use transcription and rather work on the videos directly: Auditory and visual information is often so closely knit in such specific ways that a sufficiently information-rich transcription is simply not practical.

4.4 HCI and HEI concepts

HCI and HEI concepts serve two purposes: They provide a basis for investigating nonverbal activity and they provide context for analyzing verbal activity.

These goals are vague, therefore the base concept set provides only a rather coarse framework in this area. Further layers should extend these concepts in a way that fits with those layers’ specific goals.

These are the HCI concepts we found in our data:

- write sth: Typing on the computer, including copy/paste actions.
- search sth: Searching for one or several well-defined target objects. May use an automated search function or a manual process such as scrolling.
- explore sth: Looking around in artifacts or data without a pre-specified search goal in order to explore or obtain an overview.
- do sth: All other activity that uses mouse or keyboard. Example: Modifying the IDE view setup.

These are the HEI concepts we found in our data:

- show sth: Pointing to a specific location in an artifact using the mouse or a finger.
- verify sth: Checking/verifying work products (such as changes to a given file) by way of tests (program execution) or reviews (program reading).
Table 3. Encoding for a short sequence of phenomena; P1 is the driver. The left column represents the audio information, the middle one the video information. The right column shows the codes assigned. Note that only HHI codes are shown. For simplification, the HCI and HEI codes have been neglected as they would require a different, non-compatible granularity of table lines. One HHI code is also not shown: Since P1 verbalizes some of his HCI activity, the full encoding also includes a P1.think aloud_activity code that covers the period corresponding to lines 1 through 3 of the table.

<table>
<thead>
<tr>
<th>#</th>
<th>utterance</th>
<th>further description/context</th>
<th>HHI code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P1: “That one needs All, right?”</td>
<td>P1 is currently editing the argument list of a method call. One argument is currently a call to the method getVisibleAttributes(); P1 is about to change that. By All, P1 presumably refers to the method getAllColumnAttributes().</td>
<td>P1.ask_knowledge</td>
</tr>
<tr>
<td>2</td>
<td>P2: “No idea what this thing does.”</td>
<td>P1 lets the IDE display the names of methods starting with get.</td>
<td>P2.explain_standard_of_knowledge</td>
</tr>
<tr>
<td>3</td>
<td>P1: “Yes, OK”.</td>
<td>P1 finds method getAllColumnAttributes() among the suggested names and inserts a call to it, replacing the previous getVisibleAttributes(). His utterance answers his own previous question.</td>
<td>P1.explain_knowledge</td>
</tr>
<tr>
<td>4</td>
<td>P1: “That is so you have a unique attribute name.”</td>
<td>After a short pause, P1 amends his previous utterance.</td>
<td>P1.explain_knowledge</td>
</tr>
<tr>
<td>5</td>
<td>P2: “OK, I see. OK.”</td>
<td></td>
<td>P2.explain_standard_of_knowledge</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>P1 switches into a different file by using the 'problems' list shown by the IDE. He starts scrolling through that file, but remains silent.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>P2: “Wait a minute.”</td>
<td></td>
<td>P2.stop_activity + P2.propose_step</td>
</tr>
<tr>
<td>8</td>
<td>P1: “That’s again the change from virtualColumn to virtualAttributes.”</td>
<td>Despite P2’s interjection, P1 first continues scrolling until he reaches the point in the file that is marked as erroneous. Only then does he answer.</td>
<td>P1.explain_knowledge</td>
</tr>
</tbody>
</table>

- examine_s.th: Read some previously unknown material (such as program code or configuration files) closely in order to understand it. This concept can often be recognized only indirectly, for instance by a preceding propose_step or a concurrently ongoing think aloud_activity.
- read_s.th: Read text aloud, for instance error messages.
- sketch_s.th: Create sketches, etc., usually on paper.
- read_requirement: Reading requirements specifications quietly.

4.5 Auxiliary codes

We have introduced a small number of auxiliary concepts that help understanding a given encoding for a session. These concepts are not strictly a part of the base concept set and are thus allowed to break the naming syntax rule.

- P(X).become_driver describes role switching events,
- divide work initiates a phase of non-pair work style,
- interrupt indicates an external disruption, and
- wait indicates non-activity while waiting for a search or test run to terminate, etc.

5 Using the base concept set

When using the foundation layer, one should be aware that the base concept set is not necessarily ‘complete’. Therefore, adding a new concept to it may be advisable in some (infrequent) cases.
A lot of advice could be given regarding how to use the base concepts successfully. We will explain only two of the most salient points here.

5.1 Segmentation

When using the base concept set, the unit of analysis, i.e. the granularity at which concepts are assigned to stretches of data, is the individual “phenomenon”, usually one “utterance”. So far we have relied on an intuitive understanding of these terms and not defined what we mean by them. Here are some hints on how to determine the boundaries of phenomena.

In our experience, a syntax-based segmentation of the data [5] does not work well. We recommend the following implicit rule: A phenomenon (as a stretch of time in the data) ends where the concept selected for describing it no longer fits or a different concept appears to become more appropriate. This resembles the criterion used for instance in [22]. Such an approach will often gather multiple sentences into one phenomenon but will sometimes split an individual sentence into multiple phenomena. Examples for the latter are sentences that combine a suggestion with its rationale, which will often be represented as propose_step plus explain_knowledge.

5.2 Multiple annotation

Even when allowing to split a single sentence into pieces and assign a separate concept to each and even though the concepts in the base concept set, except for the facades, are orthogonal (non-overlapping), there are cases where more than one concept appears appropriate. This can happen because natural language utterances can be rather multi-faceted, in particular when they are ambiguous, ungrammatical, or both. At least for GT analyses, it is unproblematic to assign more than one concept in such a case.

6 Conclusion and further work

We have presented the PP foundation layer, the bottom tier of a multi-layer terminology for analyzing pair programming (PP) processes. It takes the form of a set of concepts (“base concept set”) that describe individual interaction steps.

Its most important characteristic is the fact that it has been found (rather than invented), because the set contains only concepts grounded in actual observations, i.e. concepts for which we have seen instances in the PP sessions we have analyzed. This means that the concepts are likely to fit naturally with observations in future analyses, and annotating data with these concepts will be relatively easy.

Although the concepts look innocuous, even obvious, we can assure the reader that they are not. It took a long and rather painstaking process to detect, understand, and untangle them, to sort out the delineations between them, and to describe them in terms that are understandable to somebody who has not seen the original observations. The number of resulting concepts is not small, but the strong internal verb/object structure and resulting concept classes make understanding and successfully navigating the concept set easy.

Some analyses of PP can be done using the foundation layer alone, but most studies will probably add another layer on top that defines additional terminology or refines (splits up) a few concepts from the foundation layer that are of particular interest for that particular study. For instance we are currently using the foundation layer for studying how the level of abstraction changes during the PP process and will add concepts that describe abstraction levels with finer granularity than the foundation layer alone.

These are the next steps we intend to do:

– Validating the PP foundation layer against further PP sessions, in particular with professionals from other companies.
– Comparing the base concept set with thematically related coding schemes from the literature such as those from [22, 14].
– Using the PP foundation layer for various PP investigations.

References


