

Metaphorical Representation in Collaborative Software Engineering

James D. Herbsleb

Bell Laboratories, Lucent Technologies

263 Shuman Boulevard

Naperville, IL 60566

+1 630 713 1869

herbsleb@research.bell-labs.com

ABSTRACT

Finding a useful abstract representation is fundamental to solving many difficult problems in software engineering. In order to better understand how representations are actually used in key collaborative software engineering tasks, this empirical study examined all of the spoken representations of software behavior in 9 domain analysis sessions. It found that about 70% of them were metaphorical, representing system behavior as physical movement of objects, as perceptual processes, or in anthropomorphic terms ascribing beliefs and desires to the system. The pattern of use of these representations indicates 1) that they were not merely temporary placeholders, but rather their use persisted even when a specialized and more formal vocabulary had been developed, and 2) the metaphoric descriptions appear to reflect actual use of metaphor, rather than just a choice of vocabulary. The use of metaphor is explained in terms of how well they serve human cognitive abilities and collaborative needs. The predominance of metaphorical representations in synchronous collaborative sessions raises important issues about the possible misleading effects of metaphorical thinking. It also raises questions about the compatibility of the spoken representations with other representations (e.g., diagrams, specification languages) that trigger, and capture the results of, the verbal collaborative work.

INTRODUCTION

It has long been observed informally, as well as in case studies [24] and in interviews [36] that software engineers frequently talk about software behavior in anthropomorphic terms, e.g., in terms of what a component “knows” or is “trying to do.” This strikes many as sloppy and imprecise, hence undesirable. Dijkstra [12] has gone so far as to suggest that computer science faculty implement a system of fines to stamp it out among their students, although he acknowledges this would be very difficult to do.

The purpose of the empirical research reported here is to begin systematically investigating the use of these sorts of metaphorical representations in software engineering. If, in fact, their use is as pervasive as many believe, one might wonder why this is so, since it is certainly not taught explicitly as part of any standard software engineering curriculum. As the remainder of this introduction points out, prior research gives us good reason to expect that metaphorical representations of behavior,¹ and anthropomorphic representations in particular, have enormous advantages both for individual cognition and for the distributed cognition that underlies collaborative work. But pervasive use of metaphorical representations would also raise serious questions about the possible misleading effects of metaphorical thinking, and about the possibility of errors being introduced by virtue of the potential incompatibility of metaphorical representations with other, more standard, representations used in computer science.

The empirical study reported here is a content analysis of all descriptions of software or system behavior that occurred in nine domain analysis sessions. The objective is to determine the frequency with which metaphorical descriptions occurred, and to investigate several questions about how and when they were used so as to begin to understand the significance of the role they play. The specific research questions are presented at the conclusion of the introduction. The empirical methods are described in Section 2, and the results in Section 3. The paper concludes with a discussion in Section 4 of the potential opportunities and dangers revealed by these findings, and of new research questions raised.

¹ The terms “system behavior” and “software behavior” will be used interchangeably, since all the software in this study is embedded, and the behavior of the software and the system are not easily separable.

Representations in Software Engineering

There are probably few who would dispute that representations play a key role in problem solving. Herbert Simon [31] may have given us the clearest statement of the pivotal role of representations in intellectual work when he claimed that “solving a problem simply means representing it so as to make the solution transparent (p. 153).” The influence of different types of representations on an individual’s ability to solve a problem has been documented many times (e.g., [9, 21]).

Cognitive processes of individuals

There have been a number of studies that have investigated the representations and cognitive processes of programmers. Several of these have emphasized the need for abstract, integrative structures, such as plans [28] or schemata [20] for achieving and maintaining a high-level view of the developing program. Other studies have documented the effects of using various types of lower level representations such as notations [15] and visual programming languages [17].

Some types of representations seem to have particular advantages because they make use of specific, powerful, cognitive abilities. An excellent example is visualization. It is clearly not the case that all visual representations are useful for all purposes (e.g., [16]), but when a visualization matches a task well, the visual processing capabilities of the brain allow a tremendous amount of information to be conveyed in compact form which is readily grasped. For example, software visualization technologies (e.g., [1]) appear to be a significant aid to understanding very large, complex software systems.

Representations and collaborative work

For collaborative software engineering, representations must not only match and support the cognitive tasks of individuals, they must support the “distributed cognition” (e.g., [19]) of the entire constellation of people and artifacts that together produce a solution. This imposes many additional requirements on representations, such as their “openness” to viewing by more than one person simultaneously and their ability to support the sorts of exchanges and handoffs that occur among individuals. Especially important is the support representations provide for establishing “common ground” [6], i.e., the state that we agree on what we are referring to, and know that we agree.

It has often been proposed that certain types of representations used in software engineering have advantages for collaborative work. Object-oriented (OO) representations, for example, have properties that are theorized to facilitate communications among those involved in collaborative development tasks [29], and there is some evidence that use of OO representations

reduces the need for clarification in the discussion of design ideas [18].

Collections of representations

Most software engineering tasks are sufficiently complex that many representations are used in concert. A wide variety of types of diagrams, specification languages, programming languages, and so on have evolved to meet various needs. Much of the collaborative work is done in the context of meetings, formal or informal, as the participants discuss documents, diagrams, and so on, and the issues they raise [26, 27, 34]. This typically occurs in many iterations, as ideas “move” from the more stable media such as paper, whiteboard, or computer screen to the transitory verbal medium of conversation, then back to more stable media as issues are resolved. Much of the crucial work occurs during these periods of discussion when the primary representational medium is spoken language. The properties of the specific types of representations that are rendered in this verbal medium can be expected to influence the course and the success of the collaborative task as do the properties of other representations. The ease and precision with which other sorts of representations can be translated into verbal representations, and with which conclusions represented verbally can be translated into a more stable medium would also be expected to have a substantial impact on the process.

This section has focused on the critical role of representations in software engineering. I turn now to a discussion of certain types of metaphorical representations and research that bears on their suitability for collaborative software engineering tasks.

Metaphorical Representations

Metaphors are widely used to understand new or difficult domains in terms of things already understood. They can be very important and illuminating in bringing to light important characteristics one might not otherwise be able to express or even think about [2]. But they can also shape thinking and discussion in ways that close off options and neglect critical aspects of the objects under consideration [30]. This is particularly true of metaphors, as opposed to deliberate comparisons in the form of analogy [14], since metaphors are often used implicitly to establish a conceptual foundation for the lesser-known domain.

Informal observation, case studies (e.g., [24]) and interviews [36] all indicate that metaphorical language is used in various software engineering tasks. For example, communication between components or processes is sometimes spoken of as if physical objects like “packets” are sent, received, handed off, torn apart, damaged, or lost. This is clearly a metaphorical characterization of the

true physical situation, but as metaphors go, it sticks fairly close to the facts.

Other sorts of metaphor venture further from the literal. A process may “look at” the contents of a message, “recognize” a problem, “decide” to “take action.” Even further from the literal are descriptions of what a process “thinks” is happening, what it is “trying to do,” what things it “knows” about and doesn’t “know” about, and so on. These latter anthropomorphic metaphorical expressions are examples of what is often called “naïve psychology.”

Naïve psychology as a specific cognitive capability

There is considerable evidence that naïve psychology is a “special” sort of metaphor, in that it exploits a specific cognitive capability that evolved as a result of natural selection pressures favoring prehuman ancestors that were able to navigate the social world more effectively than their peers [5, 8, 22, 25]. This particular type of social reasoning ability emerges, without any special instruction, in virtually all intact humans. In a course of development which is increasingly well understood, children develop first a “naïve physics” that supports reasoning about physical objects and their movements. Somewhat later, “naïve psychology”, or “theory of mind” [23] emerges, allowing them to reason in sophisticated ways about people, based on what they infer a person’s state of knowledge and motivation to be. It appears to involve brain processes quite distinct from those used to reason about behavior in physical terms [13, 22].

Much is also understood about the mechanics of naïve psychology. It involves a particular mode of reasoning in which behavior is explained in terms of the beliefs and desires (or near-synonyms, such as “wants,” “knowledge,” and so on) of the one whose behavior is being explained (whom I’ll call the “behavior”) [5, 32]. For example, a typical explanation of why a developer selected a particular data structure might invoke the developer’s desire to perform certain manipulations on the data, and a belief that this particular data structure would allow efficient access to and storage of the data for these manipulations. Explanations can, of course, be much more elaborate, but they all tend to be cast in beliefs and desires or their near-synonyms.

One of the important component skills that allows naïve psychology to function is the ability to infer the behavior’s current state of knowledge and current desires and what further behavior these are likely to produce [11, 23]. People are very skilled at such things as keeping track of what information a behavior has been exposed to, inferring what the behavior must have known in order to have performed certain actions, and what a behavior must be trying to do, given its behavior and current knowledge.

Naïve psychology and software behavior

One of the fundamental challenges in software engineering is the extreme complexity of software [3]. Many kinds of artifacts are structurally complex, but there is little doubt that compared to any other type of artifact ever created, large computer programs (during execution) are by far the most behaviorally complex. One might speculate then, that a specific cognitive capacity that evolved in order to deal with behavioral complexity might be a powerful tool for understanding the behavior of software.

The characteristics of naïve psychology are well-suited to some kinds of tasks. It is a highly abstract representational system. In saying, for example, that a software component A “knows” about another component B, the nature of the relationship is only very incompletely specified. The statement implies merely that the behavior of A may be in some way contingent on the function or behavior of B. The nature of the contingency is unspecified. Similarly, if A is said to “believe” that a particular event has occurred, this statement implies that A may behave differently than it would if A did not “hold” this “belief” about the event. If, on the other hand, A has no beliefs one way or the other about some event, then A’s behavior cannot be (directly) contingent on the event. Statements about desire are similarly abstract. If we assert that A “wants” to communicate with B, for example, we don’t know if A will actually attempt to communicate with B, or if so, how, or with what message, or with what success.

The component skills of someone competent in naïve psychological reasoning are also matched, at least superficially, to many tasks in software engineering. In designing or understanding a complex, multi-component system, it is extremely important to be able to keep track of what state each component is in, what it is “trying” to do when its behavior causes a fault, what it “knows” about the state of other components, what it “knows” about various protocols, external entities, and so on. If the skills of naïve psychology allow an engineer to keep track of this sort of information with little effort, because of a built-in capability, it would be advantageous indeed.

Another of the interesting properties about beliefs and desires is that they are always *about* something. [4, 10]. In ascribing a belief *p* to behavior A, I am not saying something only about A. Rather, I am asserting some sort of relationship between A and the object of belief *p*. So, for example, if I say that component A believes a particular file is corrupted, this asserts something about A’s current state, but also about a particular sort of relationship that A has to the file. A description purely in terms of A’s state (e.g., variable X has value ‘0’) does not convey such information. Statements about what a component knows, believes, is trying to achieve, and so

on are very compact, abstract expressions of potentially very complex relationships that would be difficult to fully express in other, non-anthropomorphic terminology.

Finally, naïve psychological representations may be particularly well-suited to collaborative tasks. The highly developed cognitive capacity that underlies such representations is virtually universally shared. Collaborators can follow each other's reasoning quickly, with relatively little effort, and can rely on being understood. Particularly in teams with diverse training and experience, having some such representational system for establishing common ground is essential for successful collaboration [6].

I have argued in this introduction that metaphorical representations, particularly naïve psychological representations, have properties that make them appealing in some ways as a representational system for software behavior, both because of a match (albeit, perhaps a superficial one) with some important tasks, and because they make use of a specific, powerful, cognitive capability. The specific research questions are presented in the next section.

Research Questions

This research seeks to answer four basic questions about the use of metaphorical representations of system behavior in a collaborative software engineering task. Together, the answers should give a good indication of how extensively they are used, and whether the use is superficial or is an essential feature of the intellectual work. The first and most basic question is

1. How frequent are metaphorical descriptions of behavior, as compared to other modes of describing system behavior? Of the metaphorical descriptions, how many use naïve psychology?

The next question concerns how the naïve psychology representations are used. A "placeholder" hypothesis would assert that naïve psychology representations are primarily temporary placeholders for literal descriptions. Under this hypothesis, naïve psychology descriptions would be relatively frequent in the early stages of analysis, when there is the greatest need for placeholders for as-yet-undetermined literal terminology. As the analysis progresses and literal terms become increasingly available, the proportion of naïve psychology descriptions would decrease relative to literal descriptions. If they do not, then naïve psychology presumably serves some other role that continues to be important throughout the analysis. More specifically, the question is

2. Does the proportion of naïve psychology descriptions decrease over the course of each domain analysis?

The use of metaphorical vocabulary may reflect only a choice of convenient words, without actually reflecting the use of an underlying metaphorical representation. If this is the case, one would expect to find the metaphorical descriptions distributed more-or-less randomly throughout the sessions. If, on the other hand, use of metaphorical vocabulary actually reflects occasions when the participants are using a metaphor for problem solving, one would expect to see sequences of descriptions all drawn from the same type of metaphor, shifting eventually as other metaphors or literal descriptions are brought to bear. If behavior descriptions are not randomly distributed, but rather tend to occur in same-metaphor sequences, this would tend to indicate the "mere word choice" hypothesis is false. The question to be addressed:

3. Do metaphorical representations tend to occur in sequences, or is their use distributed randomly throughout the sessions?

Finally, if the descriptions of software or system behavior are actually making use of a specific cognitive system specialized for social intelligence, then one might expect to find other linguistic traces of the use of this specialized system. In particular, one might expect to see the software or system "personified," i.e., spoken of as if it were a person. There are many ways one might test for evidence of personification. I elected to use a very simple, straightforward, and conservative one. In each verb phrase describing system behavior, if the word used to denote the actor is clearly one typically associated with a person and not a "thing," this is a form of personification. So, for example, the behavior description "I just loop through the list" is clearly a personification, referring to the software's behavior as if the actor were a person ("I"). The non-personified alternative, of course, would be "It just loops . . ." The clearest evidence of personification is when the actor is in the first (I, me, we) or second (you) person. Third person pronouns may or not represent personifications, since "they" can refer to persons or things, and "he" and "she" can by convention be used for things (as when ships are referred to as "she"). Counting the occurrences of first and second person pronouns as actors in behavioral descriptions of the software is a simple, clear, and conservative way of searching for evidence of personification. If such occurrences are frequent, then the total frequency of personifications must be at least that great. So the research question is

4. How frequently is the subject of the software or system behavior description rendered in the first or second person?

METHOD

The empirical approach used in this study was a content analysis of conversation during nine domain analysis meetings.

Commonality Analysis Sessions

The data for this study were taken from sessions of a form of collaborative domain analysis called commonality analysis [35]. The commonality analysis process consists of a series of facilitated sessions in which domain experts collaboratively identify and express what is common among all members of a family (called “commonalities”) and what differs from one member to another (“variabilities”) as well as the range of values the variabilities can assume (“parameters of variation”). At the participants’ discretion, many forms of representation are used in these sessions, including data-flow diagrams, formal logic, finite state machines, and so on. The final document expressing the results of the analysis is in English (rather than a formal notation), and is structured in several sections listing the commonalities, variabilities, and the parameters of variation.

Three series of commonality analysis meetings in a large telecommunications company were selected for this study. These analyses were conducted by groups of domain experts, all of whom were quite experienced in the domain to be analyzed, and a trained moderator, who was not a domain expert. The moderators strove to capture, not dictate, the ways in which the domain experts expressed their ideas. We can assume, therefore, that the ways in which these experts talked to each other reflect their ordinary modes of talking and thinking. The basic task in commonality analysis is to identify what is common and what varies from family member to family member in the domain, so that the full variety of family members can be quickly constructed as needed. In the course of these discussions, many descriptions of what existing and future family members do and how they work were generated.

Commonality analyses typically involve at least a half-dozen sessions of 1-2 hours. Complex domains require many more sessions. Three analyses were chosen for this study. *Signal1* was a very high-level analysis of the domain of message-passing protocols in telephony. The analysis team consisted of five domain experts at the beginning. Two of them had worked together before, the others had not. All were from different organizations within the company. Two left the analysis task between the “early” and “middle” sessions. *Signal2* was another, more detailed, analysis by the same three remaining team members, conducted shortly after *Signal1* ended. This analysis covered a small portion of the domain in *Signal1*, but in much greater detail. I treated it as a separate analysis because it was considered to be separate by the participants and because it produced a separate output.

The *Revision* analysis was a completely unrelated analysis by a non-overlapping team from an organization not represented in the *Signal* analyses. The task was a revision of an already-completed analysis of switch maintenance software, i.e., software that allows hardware and software to be maintained or upgraded while the switch is operating, ensuring it remains in a “safe” condition. The purpose of this analysis was to address change requests submitted by users of the original analysis. The team had two members who remained throughout, but was joined in the “middle” session by two other domain experts whose skills were needed for a time.

For each domain analysis, three sessions were chosen, one to represent the early stages, one to represent the middle stages, and one to represent the late stages. In the case of *Signal1* and *Signal2*, the early session was the very first one in the analysis. The team had already been together to receive domain analysis training, and the sessions had a high content of analysis work. For the *Revision* analysis, the early session was the second in the series, since I was not able to videotape the first. In all cases, the middle session was one that had equal numbers of sessions before and after it. The late session was one of the last 2-3 sessions in each analysis.

Trained and experienced moderators who were not domain experts moderated all sessions. *Signal1* and *Signal2* had the same moderator, *Revision* had a different moderator.

All sessions were videotaped, with the consent of the participants. All descriptions of system (i.e., hardware and/or software) behavior were transcribed. Each verb phrase describing some action of the system was counted as a separate behavior.

Each behavior was then categorized with respect to the *type of description* (see Appendix A). The categories were

1. Specialized vocabulary (SV): literal (non-metaphorical) descriptions that are accepted computer science terms, such as *call*, *assign*, *block*, *loop*, *execute*, *parse*, and so on. Also included were words that were appropriated by the group and assigned their own specialized meaning, as indicated by their inclusion in a “dictionary” developed as a standard part of the commonality analysis.
2. Physical movement (PM): descriptions that taken literally indicate physical movement, such as *get*, *go*, *give*, *hand off*, *come back*, and so on. Also included were words associated with physical causality, such as *have an effect*, *cause something*, *start*, *contain*, and *do work*.
3. Perceptual (P): descriptions that indicate simple sensing of a condition or “looking” through

something, such as *look, see, find, search, check*. Also included were words indicating simple, “mechanical” responses to input, such as *pick* and *select*.

4. Naive psychological (Npsy): descriptions that involve beliefs or desires, such as *want, try, think, know, seem, say, trust, use, give up*. Also included were other words that clearly indicated a naive psychological view, such as *following* a rule (as opposed to executing an algorithm), or *remembering* what happened.
5. Other (O): descriptions that fell into none of the above categories.

Note that no behavioral descriptions were included in the analysis if they could be reasonably interpreted solely as behaviors of people or groups of people. Only descriptions of the behavior of the system were included. So, for example, the sentence “We loop through the list that’s in the specification” was included because clearly the analyst is not “looping” through the list, the software is. On the other hand, in “We want to loop through the list” the “want” would be excluded since a reasonable interpretation is that it is the analysts who “want.” The software does the “looping,” however, so “loop” would be included (coded as an SV type of description).

Determining whether the subject of the sentence was first, second, or third person was straightforward. The pronouns “I,” “me,” and “we” were taken to indicate first person, while “you” was taken to indicate second person. All occurrences of named people or groups, or the pronouns “they,” “it,” “them,” “he,” “she,” and so on were categorized as third person.

RESULTS

A total of 1796 behavior descriptions were extracted from the nine domain analysis sessions.

Frequencies of Description Types

The overall distribution of the types of behavior descriptions is given in the following table:

Specialized Vocabulary (SV)	483
Physical Movement (PM).....	676
Perceptual (P)	163
Naive Psychological (Npsy)	425
Other (O)	49

Overall, then, the descriptions cast in terms of naive psychology comprise a substantial portion, about 24%, of the total of behavior descriptions. If we add together all of the metaphorical descriptions, i.e., Physical Movement, Perceptual, and Naive Psychological, there were a total of 1264 such descriptions as compared to 483 “literal” descriptions, i.e., those that used a more or less well-defined computer science or domain-specific vocabulary.

In terms of percentage, about 70% of all behavior descriptions were metaphorical.

Changes in Frequency Distributions Over Time

Figures 1 through 3 show the percentages of Npsy and SV descriptions from early to late sessions within the three analyses. The percent of Npsy descriptions remains substantial throughout. It dipped below 20% in only one session (where they accounted for about 16% of all descriptions). Clearly, there is no downward trend in the proportion of Npsy descriptions. In fact, the percentage of Npsy descriptions *increased* from Early (22%) to Middle (23%) to Late (30%) when the data are aggregated across all three domain analyses.

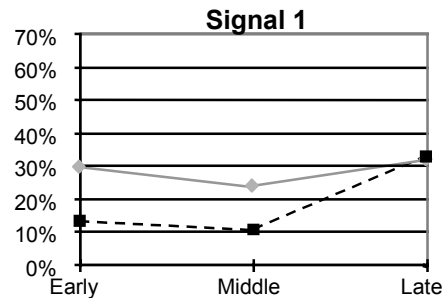
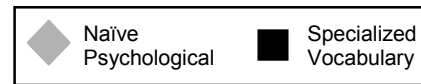


Figure 1. Percent of NP and SV descriptions in Signal 1 sessions.

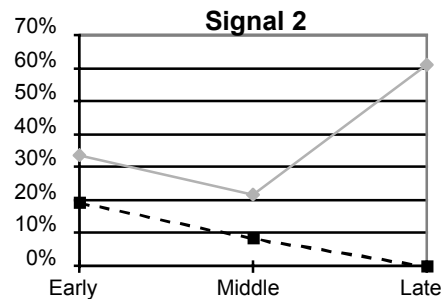


Figure 2. Percent of NP and SV descriptions in Signal 2 sessions.

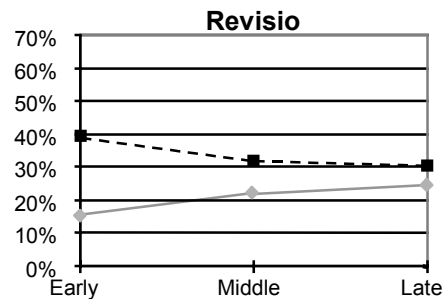


Figure 3. Percent of NP and SV descriptions in Revision sessions.

Sequential Order of Description Types

The hypothesis investigated was that behavioral descriptions using each type of metaphor would occur in sequences rather than be randomly distributed through each session. This hypothesis was tested using log-linear modeling on a table of transition frequencies (see Appendix A for details of the analysis and results). Table 1 shows the frequencies of transitions among the types of descriptions.

ANTECEDENT (Transition <i>from</i>)	CONSEQUENT (Transition <i>to</i>)				
	NPsy	O	P	PM	SV
NPsy.....	129	11.....	31.....	160.....	91
O.....	10.....	7	2.....	19.....	10
P.....	45.....	4.....	39	47.....	28
PM.....	154.....	18.....	49.....	323	130
SV.....	84.....	9.....	41.....	124.....	222

Table 1. Frequencies of transitions from Antecedent type of description to Consequent type of description. Frequencies are summed across all sessions.

The results of the analysis show that the tendency of behavior descriptions to follow descriptions of the same type is highly statistically significant. The frequencies of transitions on the diagonal (shown in **bold**) in Table 1 is much greater than would be predicted by chance (see Appendix A for further explanation).

First and Second Person Descriptions

The overall distribution of first, second, and third person actors in behavioral descriptions is shown in the following table:

person	signal1		signal2		revision		Tot.	
first	89	18%	27	13%	279	26%	395	22%
second	93	19%	77	38%	349	32%	519	29%
third	317	64%	99	49%	466	43%	882	49%
total	499	100%	203	100%	1094	100%	1796	100%

Table 2. Distribution of 1st, 2d, and 3d person.

The ordinal position of these frequencies is consistent across the three analyses. The sum of first and second person actors is never less than one-third of all actors in an analysis, and comprises a majority of actors in the other two. Overall, about one-half of all actors in the behavioral descriptions are in the first or second person.

DISCUSSION

Taken together, the results provide powerful support for the idea that metaphorical representations played a major role throughout the domain analysis sessions studied. In fact, they represent a majority of all behavior descriptions. Naïve psychological descriptions, which are arguably the “least literal” of the description types studied, do not

appear to be mere placeholders for other, more literal, descriptions, since their proportion did not decrease over time within analyses. Rather, they appear to fulfill some need that persists throughout each analysis. Nor do the metaphorical descriptions appear to merely reflect moment-to-moment choice of vocabulary, since descriptions within a single metaphor tend to occur in sequences. There is little reason to expect this non-random distribution unless the participants are adopting and staying within a particular metaphorical representational system before moving on. Finally, there is additional evidence that the participants are actually using the naïve psychology metaphor, since they “personify” the software, as indicated by use the first and second person to represent the software as actor, in about half the descriptions (see also [36]).

These results need to be interpreted cautiously, of course. The data all came from domain analysis sessions, and domain analysis is a particularly abstract task, where one is describing a whole family of applications. The software in all cases was large, highly complex telecommunications software. It may be that with smaller programs, or with less abstract tasks, there would be less of a tendency to bring metaphorical descriptions to bear. Additionally, although the subjects came from several different development organizations, they were all from the same company. The results might be different in other corporate and engineering cultures. Studies of other tasks, people, and settings are clearly needed.

Abstraction and Speed

It would not be surprising, however, if the complexities of reasoning about state, behavior, faults, interactions, scenarios constructed on the fly, and so on, virtually *require* the use of naïve psychological “social intelligence.” People find it relatively easy to keep track of what other people know about a complicated situation, what these other actors are trying to do, how they are likely to react to new information, what knowledge and intentions can be inferred from their behavior, and so on. The enormous cognitive demands of collaborative software engineering tasks may simply require developers to use these intellectual resources, especially since this may be the only powerful representation and reasoning system shared by all members of diverse teams. Collaboration also demands that participants reason very quickly about extraordinarily complicated systems, e.g., to construct and walk through a novel scenario, to understand the implications of a change, and so on, at conversational speed. Synchronous collaborative work imposes a fast pace that is unlike the self-paced tempo of individual work. It may be extremely difficult to achieve this level of abstraction and speed in collaborative work in any other way. Again, it will require additional studies of other tasks, people, and settings to address this question.

Fusing the Organization and the Artifact

The specific way in which the domain analysts personified the software shows another important feature of naïve psychology that may be important for collaborative work within a large organization. They tended to identify the software with the person or team that produced it, as when remarking that “we’re assuming *you* are left in a valid condition, once *we* stop.” Notice it is the system, not the people, that are “left in a condition” and that “stop.” Yet the verbal “fusing” of the people and the part of the system they created results in a personification that is far from arbitrary, but rather helps the participants stay constantly aware of the mapping of system functionality onto the development organization. In fact, there was at least one occasion when temporary confusion resulted over whether a particular behavior under discussion was something that a team of people performed, or whether it was something the software those people designed performed.

This apparent “fusion” or identification of people with their artifacts is a powerful acknowledgement of the importance of “Conway’s Law,” i.e., that the structure of the software reflects the structure of the organization [7]. People appear to take advantage of this mapping in the “fused” representation that provides a very compact way of keeping track of critical organizational information, as well as information about the system, in a single structure.

Naïve Psychology and Standard Representations

The relationship of naïve psychological representations to other, more standard, software engineering representations is also an important issue. Representations must often be moved from one medium to another [19, 27] as when, for example, part of a diagram is discussed (diagrammatic to verbal) or the results of the discussion are captured in the diagram (verbal to diagrammatic). If this translation is difficult, one might expect errors to creep in with some frequency. If the words used to discuss an issue cannot accurately express the situation, or if the consensus achieved in a discussion cannot be fully expressed in a more durable representation, the results are unlikely to be satisfactory.

This line of reasoning has several implications. First, there might be significant advantages for using “standard” representations that are as compatible as possible with naïve psychology. One might make the case, for example, that agent representations are a good fit [33], or even that OO representations, since objects have both state and behavior, fit well with naïve psychological capabilities. On the other hand, functional representations are relatively incompatible with naïve psychology. Future work should examine the discussions around representations of various sorts to see if naïve psychology is often used in conjunction with “incompatible”

representations, and if so, whether the translations to and from “talk” are difficult or error-prone.

The second implication is that it is important to achieve a better understanding of the semantics of naïve psychological representations. If we had a clearer idea of what it means to say, e.g., that component A should, in a particular scenario, “know” that component B is not functioning properly, then we would be in a better position to see if this “knowledge” is fully implemented in the detailed design and code.

A third, and somewhat more radical, possibility would be to design specification languages that are either directly based on naïve psychology, or that are designed to be easily translated. As I mentioned in the introduction, it has occasionally been suggested that anthropomorphic talk should be somehow banned because it is sloppy and imprecise. Perhaps we should take the opposite tack and adjust our standard representational machinery to accommodate the modes of reasoning and representing that actually are used by software engineers.

A Cautionary Note

It is important to point out that the capacity for naïve psychology evolved in order to understand human minds, not software. Human minds and software differ in many obvious and significant ways, regardless of the programming paradigm used. For example, we tend to think of beliefs as atomic, i.e., if I ascribe a belief *p* to a human, I assume they will *always* act as if they believe *p*, until they (atomically) cease to believe *p*. Software may or may not be consistent, perhaps acting sometimes as if it believes *p*, and other times as if it doesn’t (if, e.g., some functions check a particular piece of state, but others were written incorrectly and do not). Similarly, if I ascribe a desire to a human, I expect them to seek an alternative way to fulfill it if one specific path is blocked. Again, software is often not that smart. An interesting research question for the future is to try to determine if these sorts of assumptions actually lead to significant numbers of software errors.

ACKNOWLEDGEMENTS

I would like to thank the commonality analysis participants and moderators for allowing me to videotape their sessions. I also thank Beki Grinter for comments on an earlier draft of this paper, and Audris Mockus for comments as well as advice on the statistical analyses (any errors or omissions are solely the responsibility of the author, of course).

REFERENCES

- [1] T. Ball and S. Eick, "Software Visualization in the Large," *IEEE Computer*, Vol. 29, No. April, 1996, pp. 33-43.

- [2] R. Boyd, "Metaphor and Theory Change: What is "Metaphor" a Metaphor for?," *Metaphor and Thought*, A. Ortony ed., Cambridge University Press, Cambridge, UK, 1993, pp. 481-542.
- [3] F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, 1987, pp. 10-19.
- [4] P. Churchland, "Eliminative Materialism and the Propositional Attitudes," *Journal of Philosophy*, Vol. 67, No. February, 1981, pp. 67-90.
- [5] A. Clark, "From folk psychology to naive psychology," *Cognitive Science*, Vol. 11, No. 1987, pp. 139-154.
- [6] H.H. Clark and S.A. Brennan, "Grounding in Communication," *Perspectives on Socially Shared Cognition*, L.B. Resnick, J.M. Levine and S.D. Teasley ed., APA Books, Washington, DC, 1991, pp. 127-149.
- [7] M.E. Conway, "How Do Committees Invent?," *Datamation*, Vol. No. April, 1968, pp. 28-31.
- [8] L. Cosmides and J. Tooby, "Origins of domain specificity: The evolution of functional organization," *Mapping the Mind: Domain Specificity in Cognition and Culture*, L.A. Hirschfeld and S.A. Gelman ed., Cambridge University Press, New York, 1994, pp. 85-116.
- [9] R.S. Day, "Alternative representations," *The psychology of learning and motivation*, G. Bower ed., Academic Press, Orlando, FL, 1988, pp. 261-305.
- [10] D.C. Dennett, "Intentional Systems," *Journal of Philosophy*, Vol. 8, No. 1971, pp. 87-106.
- [11] D.C. Dennett, "True believers: The intentional strategy and why it works," *The Intentional Stance*, ed., MIT Press, Cambridge, MA, 1987, pp. 13-35.
- [12] E. Dijkstra, "On the cruelty of really teaching computer science," *Communications of the ACM*, Vol. 32, No. December, 1989, pp. 1398-1404.
- [13] P.C. Fletcher, F. Happe, U. Frith, S.C. Baker, R.J. Dolan, R.S.J. Frackowiak and C.D. Frith, "Other minds in the brain: A functional imaging study of "theory of mind" in story comprehension," *Cognition*, Vol. 57, No. 1995, pp. 109-128.
- [14] D. Gentner, "Are Scientific Analogies Metaphors," *Metaphor: Problems and Perspectives*, D.S. Miall ed., The Harvester Press, Sussex, England, 1982, pp. 106-132.
- [15] T.R.G. Green, "Cognitive dimensions of notations," *Proc. Fifth Conference of the British Computer Society*, Cambridge University Press, 1989, pp. 443-460.
- [16] T.R.G. Green and M. Petre, "When Visual Programs Are Harder to Read than Textual Programs," *Proc. Sixth European Conference on Cognitive Ergonomics*, 1992, pp.
- [17] T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework," *Journal of Visual Languages and Computing*, Vol. 7, No. 1996, 1996, pp. 131-174.
- [18] J.D. Herbsleb, H. Klein, G.M. Olson, H. Brunner, J.S. Olson and J. Harding, "Object-oriented analysis and design in software project teams," *Human-Computer Interaction*, Vol. 10, No. 1995, pp. 249-292.
- [19] E. Hutchins, "The Technology of Team Navigation," *Intellectual Teamwork*, J. Galegher, R.E. Kraut and C. Egido ed., Lawrence Erlbaum, Hillsdale, NJ, 1990, pp. 191-220.
- [20] R. Jeffries, A.A. Turner and P.G. Polson, "The processes involved in designing software.," *Cognitive skills and their acquisition.*, J.R. Anderson ed., Erlbaum, Hillsdale, NJ, 1981, pp. 255-283.
- [21] J.H. Larkin and H.A. Simon, "Why a diagram is (sometimes) worth ten thousand words.," Vol. 11, No. 1987, pp. 65-100.
- [22] A.M. Leslie, "Pretense and representation: The origins of "theory of mind",," *Psychological Review*, Vol. 94, No. 4, 1987, pp. 412-426.
- [23] A.M. Leslie, "ToMM, ToBy, and agency: Core architecture and domain specificity," *Mapping the Mind: Domain Specificity in Cognition and Culture*, L.A. Hirschfeld and S.A. Gelman ed., Cambridge University Press, New York, 1994, pp. 119-148.
- [24] K.H. Madsen, "A Guide to Metaphorical Design," *Communications of the ACM*, Vol. 37, No. 12, 1994, pp. 57-62.
- [25] S. Mithen, *The Prehistory of the Mind*, Thames and Hudson, Ltd., London, 1996.
- [26] G.M. Olson, J.S. Olson, M.R. Carter and M. Storosten, "Small group design meetings: An analysis of collaboration," *Human-Computer Interaction*, Vol. No. 1992, pp.
- [27] C. Potts and L. Catledge, "Collaborative Conceptual Design: A Large Software Project Case Study," *The Journal of Collaborative Computing*, Vol. 5, No. 1996, pp. 415-445.
- [28] R.S. Rist, "Plans in Programming: Definition, Demonstration, and Development," *Proc. Empirical Studies of Programmers*, Ablex, 1986, pp. 28-47.

- [29] M.B. Rosson and S.R. Alpert, "The cognitive consequences of object-oriented design," *Human-Computer Interaction*, Vol. 5, No. 1990, pp. 345-379.
- [30] D.A. Schon, "Generative Metaphor: A Perspective on Problem-Setting in Social Policy," *Metaphor and Thought*, A. Ortony ed., Cambridge University Press, Cambridge, UK, 1993, pp. 138-163.
- [31] H.A. Simon, *The sciences of the artificial*, The MIT Press, Cambridge, MA, 1981.
- [32] S. Stich, *From Folk Psychology to Cognitive Science*, MIT Press, Cambridge, MA, 1983.
- [33] M.D. Travers, *Programming with Agents: New Metaphors for Thinking about Computation*, Doctoral Dissertation, Cambridge, MA, Massachusetts Institute of Technology, May 3, 1996.
- [34] D.B. Walz, J.J. Elam and B. Curtis, "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration," *Communications of the ACM*, Vol. 36, No. 10, 1993, pp. 62-77.
- [35] D.M. Weiss, "Commonality Analysis: A Systematic Process for Defining Families," *Proc. ARES Workshop on Development and Evolution of Software Architectures for Product Families*, 1998, pp.
- [36] J. Weitzenfeld, T. Reidl, C. Chubb and J. Freeman, "The Use of Cross- Domain Language by Expert Software Developers," *Journal of Metaphor and Symbolic Activity*, Vol. 7, No. 3 & 4, 1992, pp. 185-195.

APPENDIX A: CATEGORIZATION AND STATISTICAL ANALYSIS

This appendix presents some details of how the behaviors were categorized, and about the statistical analysis underlying the conclusions in the section on "Sequential Order of Description Types."

Behavior Categorization

The sentence containing each verb phrase was entered in a cell in a spreadsheet. Once all the verb phrases had been entered, all information permitting identification of the phrase with a particular meeting or condition was hidden. This was done to ensure that the coder's expectations could not influence the categorization. The sentences were then ordered randomly so that any "drift," or unintentional change in the categorization procedure over time would produce only random noise rather than systematically affecting the results.

A preliminary pass through part of the data was made in order to identify ambiguous cases where categorization proved difficult. A set of rules was developed in order to handle these ambiguous cases consistently. In all, there were over two pages of such rules (which are available from the author). This is sufficient to establish a reasonable level of confidence that the categorization procedure was consistent and unbiased.

Statistical Analysis

A log-linear model of the frequencies in Table 1 was constructed, including the main effects and the cells on the diagonal as factors. The diagonal cells represent the transitions from a description type to itself. Therefore, testing this model is a way of determining whether, overall, description types tend to repeat themselves.

The results (Table 2) show that the diagonal cells have a higher frequency than would be expected based on the marginal frequencies alone (chi-square = 139, one degree of freedom, $p \ll .001$). The metaphorical descriptions are clearly not distributed randomly throughout each session, but tend to occur in sequences of a single type.

	Df	Deviance	Resid .Df	Resid Dev	Pr(Chi)
NULL.....	24.....	1845			
antecedent ...	4.....	830.....	20	1015	0.0000
consequent ..	4.....	827.....	16	188	0.0000
ante = cons. 1.....	139.....	15	15	49	0.0000

Table 3. Results of log-linear model of transition frequencies in Table 1.

A further analysis was conducted to rule out one alternative explanation of these findings. It is possible that the results reported above are due entirely to

individual preferences of the meeting participants. So, for example, if one speaker prefers "physical movement" verbs, and another prefers "naïve psychological" verbs, then the overall data would tend to indicate that verb types repeat themselves simply because speakers tend to utter several verbs during each conversational turn. Thus, the individual preference could be mistaken for an overall tendency to stick with a single metaphor.

In order to test this alternative explanation, a table of transition frequencies was constructed using only those transitions where the speaker changed, discarding transitions between consecutive descriptions uttered by the same speaker. This generated the following table:

ANTECEDENT (Transition <i>from</i>)	CONSEQUENT (Transition <i>to</i>)				
	NPsy	O	P	PM	SV
NPsy	35	2	9	42	20
O	5	2	1	8	6
P	14	1	10	15	7
PM	42	8	16	90	39
SV	34	5	10	34	85

Table 4. Transition frequencies where speaker changes from one description to the next.

Again, constructing a log-linear model with main effects and the cells on the diagonal as factors, the results show that the diagonal factor is highly significant (chi-square = 52, one degree of freedom, $p \ll .001$).

	Df	Deviance	Resid .Df	Resid Dev	Pr(Chi)
NULL	24.....	537			
antecedent	4.....	231.....	20	307	0.0000
consequent ...	4.....	234.....	16	72	0.0000
ante = cons..1.....	52.....	15	15	21	0.0000

Table 5. Results of log-linear model of transition frequencies in Table 4.

Thus, while there may be a tendency for speakers to prefer a particular type of behavior description, that preference cannot account entirely for the overall tendency for description types to occur in sequences.