Impreciseness and Its Value from the Perspective of Software Organizations and Learning

Grigori Melnik, Michael M. Richter

Department of Computer Science, University of Calgary
Calgary, Canada
melnik@cpsc.ucalgary.ca; richter@informatik.uni-kl.de

Abstract. When developing large software products many verbal and written interactions take place. In such interactions the use of abstract and uncertain expressions is considered advantageous. Traditionally, this is not the case for statements which are imprecise in the sense of being vague or subjective. In this paper we argue that such statements should not only be tolerated but, often, they can be very useful in interaction. For this purpose we relate abstraction, uncertainty and impreciseness to each other by investigating the differences and common properties. We also discuss the relation to the use of common sense implementations in Artificial Intelligence. The introduction of degrees of impreciseness leads to the question of finding an optimal level. This is interpreted as a learning problem for software organizations. The success can be measured in terms of different cost factors. The design of evaluation experiments is shown as an interdisciplinary task.

1 Introduction

Traditionally, software engineering as an engineering discipline has always required preciseness in requirement specifications, design, implementation, and other stages of the lifecycle. This is what many software engineering methodologies advocate (e.g., Waterfall, Spiral, Cleanroom etc.) and it is what formal methods[1] are for. For example, when dealing with requirements, the classic software engineering textbooks teach us that "the specification document should not include imprecise terms (like *suitable*, *convenient*, *ample*, or *enough*) or similar terms that sound exact (e.g. because they use numbers) but in practice are equally imprecise, such as *optimal* or *98 percent complete*." [7]

This is strongly related to the current debate on which approach is preferable – tayloristic or agile. We will not make any principle statements but rather take the position of those who think that the truth is somewhere in between those extremes and one

[1] A software development method is "formal" if it is a formal system in the sense of mathematical logic. This means that there is a formula language with a precisely defined syntax, there is a fixed meaning to the formulae, and there is a calculus to analyse or transform the formulae without recurring to the meaning (adopted from [3]).

has to decide how much of each approach to use based on a particular situation (application domain, relationship to the customer, development environment etc.)[2].

There are reasons for formal rules. Some of them are legal, others are pragmatic/practical. For example, in traditional software engineering, the specification document is essential for both testing and maintenance. "Unless the specification document is precise, we cannot determine whether the specifications are correct, let alone whether the implementation satisfies the specifications." [7].

We would like to emphasize that the concept (object) of impreciseness is used in various ways by different people. Impreciseness could mean several things:

a) being abstract (by concealing low level details and focusing on high level views of the problem or the solution);

b) being incomplete (by telling one part of the story)[3];

c) being uncertain (by expressing probabilities for events, beliefs, estimates and so on);

d) using informal and vague expressions (i.e. expressions without clearly defined semantics).

Ordinarily, the first three types – abstract, incomplete, and uncertain – have merits in themselves and are adequate in many situations (last not least in the development of large-scale software). The last type (informal/vague) does not have common recognition and is often considered to be a flaw. However, conversations among team members and with the customer often include such informal, vague, and imprecise expressions. In the context of the present paper, we understand the term "impreciseness" as a quality of lacking precision and using vague or subjective expressions.

Our recommendation is that instead of attempting to eliminate impreciseness or at worse to tolerate it (if it cannot be eliminated), the teams should learn how to use impreciseness properly and to their advantage.

In the next sections, we discuss such potential advantages and also what it means to use impreciseness properly. We will also examine the relationship between impreciseness and organizational learning. For this purpose, we will have to look at the impreciseness-abstraction relationship and impreciseness-uncertainty relationship.

2 Scenarios

In order to illustrate our arguments and to motivate the use of impreciseness, abstraction and uncertainty, let us consider several hypothetical scenarios in the context of developing software for civil engineers and architects. Any other group of customers could have been taken. We emphasize here, however, that much of the need for being

[2] Barry Boehm and Richard Turner discuss this balance between agility and Taylorism by defining "home grounds" – where each of the approaches is most comfortable (for more information, we refer the reader to [1].

[3] Incompleteness can only be improved by getting more information from the customer. This is a problem of dialogs with which we will not deal here.

informal results from the fact that most customers are not computer scientists or software engineers and they deal with non-formalized problems.

Scenario A: Abstract Expressions

A requirement *"to connect to a certain database"* is abstract, but, nevertheless, has a clear meaning. A more detailed requirement would be *"to connect to a DB2 database via JDBC Level 3 application driver"*. Abstraction is useful because it allows us to focus on the problem at large and not on the particular, low-level details (the specific database driver in this case).

Scenario B: Uncertain Expressions

An instruction *"to budget for 3 to 5 person months"* to complete a task is clearly an uncertain expression. However, it still has a defined meaning. To make it more certain, one may use a phrase like *"budget for 3½ to 4½ person months*". The reason for being uncertain is that we presently have only estimates for the budget. Closer investigation may lead to more accurate estimates.

Scenario C: Imprecise Expressions

Suppose we have to present features of the building construction to an architect. A non-functional aspect of this could be a requirement *"to make it user-friendly and understandable to architects*". This is clearly vague and can be interpreted in a variety of ways. To interpret this phrase properly, it is necessary that one has background knowledge about the types of features architects like and consider user-friendly. A somewhat more precise requirement would be *"to use a graphical user interface like XYZ"*, where XYZ is some common package known to architects. It is not possible to make the statement *"user-friendly for architects"* precise in every respect. It is also not necessary because that would require a complete understanding of the architecture domain. To interpret such requirements one needs a developer who has some knowledge and experience in the area of usability and human-computer interaction. The advise to utilize an interface in a certain style makes it somewhat more precise but, still, does not have a well-defined, unambiguous meaning.

Through these three examples we observed various degrees (levels) of abstraction, uncertainty and impreciseness. These levels are not independent of each other, because impreciseness may only become expressible (visible) at a certain level of detail. For example, the term "*user-friendly*" makes sense only when you come to the level when the user is involved. At the highest level, in a conversation, such statement would not be considered as a useful contribution but rather as a distracter. Unnecessary statements in a conversation are often more confusing than helpful. A statement only makes sense and will be considered as constructive when one comes to such level of details of talking about the user.

On the other hand, not everything should be expressed in an imprecise way. There are cases when preciseness is necessary. An example of such requirement could be a

due date or if the use of this particular XYZ interface is prescribed (because, for example, the company of the customer is required to use this interface by the contract). It is often difficult to say which level of impreciseness is the most useful one (and it is not even clear what *useful* means). We explain this first informally and will later relate this to cost functions.

If we consider *useful* as *valuable*, in our scenario it may be, that mentioning the phrase *"to use an interface like XYZ"*, in fact, provides no additional information (value) to an experienced developer.

3 Formal Aspects

Despite the fact that we are talking about abstract, uncertain or imprecise concepts, it is still necessary to introduce formal notions when discussing such concepts. We can draw a parallel with the theory of probabilities that deals with uncertainties but is, nevertheless, a formal mathematical discipline.

When considering all three types (abstractions, uncertainties and impreciseness), there exist partial orderings in the sense of *more* or *less*. Furthermore, with each abstract and each uncertain expression, there is a set of all possible interpretations associated with it. This set constitutes the meaning (the set theoretic semantics) of the expression. In particular, a set of all interpretations for an abstract concept is comprised of instances of all possible details. A set of all interpretations for an uncertain concept consists of all data points within the range specified by uncertainty (estimate). Similar well-established interpretations are given by probability distributions.

In contrast, one cannot associate a precise set of all possible interpretations for an imprecise concept, because a definition of such set is needed but unavailable.

In cognitive science this was well-realized several decades ago. Instead of defining a set of all possible interpretations, a notion of "category" was introduced (see [2, 5, 4, 6] for example). Bruner, Goodnow, and Austin, in their influential book "A Study of Thinking" [2] discuss ways in which people organize knowledge:

> *"We begin with what seems to be a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions. But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our environment. The resolution of this seeming paradox – the existence of discrimination capacities, which, if fully used, would make us slaves to the particular – is achieved by man's capacity to categorize. To categorize is to render discriminably different things equivalent, to group the objects and events around us into classes, and to respond to them in terms of their class membership rather than their uniqueness."* [2]

A category does not have a definition but it has prototypical and other members. What is regarded as prototypical depends upon the social conventions of the people involved in the communication. Other members of the category may exist and they are related to some prototype via certain links or patterns termed as "properties". The

use of categories in human conversations requires certain flexibility and intelligence of the participants in order to interpret the category in their own contexts. E.g., the expression "*programmer who has experience with architects*" may have Bill as a prototype in the company. This category may have Mary as another non-prototypical member because she joined Bill during two such projects.

Categorization reduces the complexity of the environment. For example, when one uses a phrase "*we should do it in an object-oriented way*", the statement itself is imprecise. However, there are certain concepts associated with the category "*object-oriented programming*" (objects, classes, state, behaviors, encapsulation, inheritance, polymorphism, etc.), which, for an experienced software engineer is quite sufficient at a certain stage of the conversation. Any attempt to make this more precise (by considering all flavors and details of object-oriented languages/methodologies) will result in an endless endeavor, will still be incomplete, and will not be helpful.

In addition, categorization reduces the need for constant learning – we do not need to be taught about novel objects if we can categorize them. There are possibly hundreds of various implementations of relational databases. However, the category of relational databases allows us to understand the way a new database (which we never heard of) works because we associate it with the properties of relational databases (relations, tuples, keys, relationships, etc.)

This is closely connected to the problem of using common sense expressions, which occur in almost every human conversation. In the field of artificial intelligence, the attempts to formalize common sense have been less than successful, as can be seen, for instance, from the Cyc project[4]. In this project, started in 1984, the researchers set out to build a huge knowledge base system containing all concepts, facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday human life. During the development of such knowledge base, it became clear that the killing factor was not the number of information units but the number of relationships among them. Therefore, the original ambitious goal was reduced significantly to more manageable but still useful tasks – the current implementation of the Cyc knowledge base consists of thousands of "microtheories", which are focused on a particular domain of knowledge, a particular interval in time, or a particular level of detail.

A possible consequence for formalizing common sense (concepts, facts etc.) in software engineering will be to develop similar such microtheories. Because software engineering is so complex that it spans all aspects of modern life, it would also require a vast number of such microtheories. Besides the sheer skepticism that this approach can ultimately work, we cannot even wait for this to be done[5]. Thus, success in such endeavor is simply impossible.

The next important question is whether we should use imprecise expressions at all. Instead of using an informal/imprecise expression associated with a category, the alternative is to use precise statements that describe one or more member of that category. In other words, we would have to select one of many possible interpretations of an expression. That would be justified if this interpretation (the selected one) is the only one that should be used for solving a specific task. This would require that we can foresee all of the future aspects in the context which we do not know yet. In addi-

---

[4] See http://www.cyc.com and http://www.opencyc.org.

[5] In case of the Cyc project, it has been going on in various incarnations for over 20 years now.

tion, it would not require any use of human intelligence. This would result in difficulties in dealing with even small errors. In the earlier example of the qualification of the programmers Bill and Mary, a discussion about what abilities are precisely needed would be endless and useless.

There is a debate in IT industry on how many details should be planned ahead of time. The Cleanroom advocates not only encourage planning upfront but also performing a formal validation of that plan. The newly emerged fleet of agile methods does not share this view. In contrast, they emphasize that developing software in the real world involves gracefully and continuously adapting to change. Agile methods encourage teams to concentrate on clean code that works and the value delivered to the customer rather than the process itself. Having said that, agile methods do not reject the concept of planning. Agilists perform planning rigorously but only within a given iteration (which is normally short – 2-4 weeks).

The situation for imprecise concepts is even more challenging because, as can be seen from our previous discussion, it is easier to make a plan more detailed than it is to make an imprecise/subjective statement more precise.

4 Learning

The last problem which remains open is to find out which level of impreciseness should be chosen. For this purpose we have to point out, what is considered as more or less useful. We have discussed this from the practical point of view earlier and will connect this question with quantifiable experiments now.

If a certain expression is too imprecise, then important issues (that can be made precise and are necessary to know) may be lost. If it is too precise, then certain possible interpretations (that are useful) may be missed.

If we are talking on a very high level, we need many expressions in order to come to the category that we want to describe. On the other hand, if we are communicating on the very detailed (low) level, we encounter a difficult task of synthesis in order to reach the concepts we are interested in. To illustrate this, let us consider a very simple example. Suppose the team leader is in the process of talking to the audience that knows graphics tools quite well and knows, in particular, about the valuable properties of the tool XYZ. Now the team leader wants to communicate the following message:

1) *"All components using graphics in the style of XYZ were very well received by the new customer"*.

Instead of using this statement she could also have said[6]:

2) *"All components using tools of the kind c_2, c_5, c_9 and c_{14} were very well received by the new customer";*

3) *"All components that were implemented for architects and took care of aesthetic aspects were very well received by the new customer"*.

[6] Here we assume that they are logically equivalent; we also consider c_2, c_5, c_9 and c_{14} as a specific subset of the set of available tools $\{c_1,\ldots c_n\}$.

Description (2) is too detailed and description (3) uses terms which are too general. In addition, both descriptions do not deliver the same message if we want to emphasize the use of "*graphic tools for architects*". At each level, we associate many concepts and instances with terms mentioned on this level. For example, in description (3), when one mentions "*for architects*", this could also mean office tools, etc. Or, if we say "*aesthetic aspects*", this could mean general principles, not necessarily related to architect's job. In description (2), with tool c_2, for example, we can associate its producer, the price or other things.

Description (1), though being imprecise, is the most appropriate one because it is the only one that generates the intended associations – the audience associates the useful graphics tools for architects with XYZ style. The common background here is the fact that all participants are aware of the fact that "*XYZ style is used for the architect tools*" and they know what is important about this style. This is related to the fact that statements which are equivalent but use different wordings, may generate very different cognitive associations. Indeed, much of the advantage of using certain imprecise expressions has just such motivation.

The possible cost of choosing the wrong level of communication is twofold: 1) the time consumed by the discussions, and 2) the number of errors due to misinterpretations. Both are common measures in software engineering.

Discovery of the right level that minimizes these costs is a learning process.

The problem here is that the level of impreciseness is not the only influence factor for these costs to occur. To investigate these factors, experiments are needed. Performing an experiment would require: 1) to name all the major influence factors for the cost; 2) to ensure that all other influence factors are invariant during the experiment.

It is further important to notice that this refers to all participants of the conversation – which eventually extrapolates to the entire organization as more and more people get involved in conversations. This clearly becomes a social process. Therefore, the expertise from sociology, social psychology and behavioral studies is required for designing, implementing and evaluating such experiments.

5 Common Background

In any conversation it is necessary that participants understand each other well – at least to some degree. When talking about precise statements, it often occurs that terms are being interpreted by people differently. This is even the case in such areas where it is least expected, like banking, for example. In converting currencies different stock exchanges refer to different points of time that are, however, not mentioned explicitly. Another example are recommendations of brokers. They evaluate shares by different methods, so "*strong buy*" can mean something different in London and New York [9]. In the context of our paper, we would rather consider "*strong buy*" as an imprecise statement.

For comparable reasons, in the database theory the concept of the mediator has being vigorously discussed for more than a decade. Originally, the task of the mediator

was to provide syntactical translation. It turned out, however, that considering the semantics was even a more significant issue [10].

In order for people in a conversation to understand all notions in the same way, certain semantics must be used. This semantics can be regarded as a common background of the group. In particular, common terminology is a must.

In a conversation that uses imprecise statements, this requirement of the common background would be impossible to satisfy because the precise semantics is simply not available. Therefore, somewhat weaker requirement is wanted. Such a requirement cannot refer to the interpretations of possible semantics themselves. The only way to deal with this problem is to refer to the cost that arises when people use their subjective interpretations (which we call "personal semantics"). At the end of the previous section, we gave examples of possible costs measures.

The formation of the common background is again a social process that we correspondingly regard as a learning process of the organization. Therefore, our argumentation for the involvement of the interdisciplinary experts in experimentation stands here as well.

The problem of common background and common knowledge has attracted substantial attention in the past.

Unfortunately, many projects on designing and deploying experience and knowledge bases to create and sustain common background in organizations suffered from the lack of user involvement and initiativeness. This mistake of "build it and they will come" is common. If the organization recognizes informal conversations as a useful tool, than it is clear than the common background for this has to be formed using informal conversations.

In a more general perspective, the common background can be regarded as a special facet of organizational identity, spirit, traditions, and culture.

6 Summary

In this paper we have considered the notion of impreciseness and related it to abstraction and uncertainty. Our main concern was the use of such concepts in conversations in organizations that develop software. We argued that the lack of preciseness was not necessarily something that should be avoided, but, in fact, could be turned in an advantage. We have also related this discussion to the problem of formalizing common sense. An attempt to do so is utopian.

However, impreciseness has to be used properly. For that reason, we introduced a partial ordering which led to different levels of impreciseness. An ideal level would be the one that encompasses a set of interpretations precise enough for everybody in the conversation to utilize their own intelligence and abilities to solve the problem; and at the same time, high (imprecise) enough to be accepted by everybody.

Achieving such level is a social process. In the context of software engineering, it is regarded as a learning software organization process. The use of imprecise statements also requires a common background that can emerge through a learning process.

For evaluating the success of such learning processes we suggested the use of classic measures of software engineering – the time (effort) spent and the number of errors due to misinterpretations.

We pointed out that defining and implementing proper experiments cannot be done naively and will require participation of interdisciplinary experts (especially, sociologists and organizational psychologists).

7 Outlook of the Future Work

In the previous sections we have analyzed the role of imprecise statements in conversations among software developers. We have also pointed out situations where the use of such expressions may be to an advantage. This forms a necessary foundation for future experiments that will be conducted to substantiate our view. We hope that this ignites a discussion and leads to generation of new ideas.

The design and execution of such experiments is the next step of the current investigation. We are well conscious of the fact that the design of these experiments will not be trivial. The type of experimental work to be done can be outlined as follows:

1. Record a conversation between two randomly selected software developers without interfering.
2. Extract the imprecise statements from this conversation.
3. Collect initial set of metrics (elapsed time, the number of imprecise statements, the usage frequencies, the number of requests for clarification, etc.).
4. Independently confront both participants with these statements, asking the following 2 questions:
    a. What is your understanding of this particular imprecise expression?
    b. Why did not you use more precise statements in the current context?
5. Analyze participants' responses and get a qualitative insight how and why they used the terminology and expressions. It will be deduced whether the result was useful, or leading to errors, or needing additional sessions.
6. Repeat the experiment with two other people conducting a conversation about the same task (topic), during which participants are instructed not to use any imprecise statements.
7. Collect the same set of metrics as in step 3.
8. Perform comparative analysis.

Certainly, this experiment needs to be repeated with other participants. In addition, an advice of a professional psychologist is required, in order to ensure that there are no other influence factors and to enhance the external and internal validities of the study.

References

1. Boehm, B., Turner, R. Balancing *Agility and Discipline: A Guide for Perplexed*. Boston, MA: Addison-Wesley, 2003.
2. Bruner, J., Goodnow, J., Austin, G. *A Study of Thinking*, New York, NY: Wiley, 1956.
3. Hussmann, H. "Indirect Use of Formal methods in Software Engineering". Online http://www.inf.tu-dresden.de/ST2/ST/papers/icse17-ws.pdf. Last accessed March 1, 2004
4. Lakoff, G. *Women, Fire and Dangerous Things: What Categories Reveal about the Mind*. Chicago, IL: The University of Chicago Press, 1987.
5. Lakoff, G., Johnson, M. *Metaphors We Live By*. Chicago, IL: University of Chicago Press, 1980.
6. Mervis, C., Rosch, E. "Categorization of Natural Objects". *Annual Review in Psychology*, 32: 89-115, 1981.
7. Schach, S. Object-oriented and Classical Software Engineering, 5/e. New York, NY: McGraw-Hill, 2002, p.36.
8. Stroustrup, B. "Interview to Artima, Sep, 2003". Online http://www.artima.com/intv/abstreffi.html Last accessed March 1, 2004.
9. Wache, H. *Semantische Mediation für heterogene Informationsquellen*. Akademische Verlagsgesellschaft Aka GmbH, Dissertationen zur Künstlichen Intelligenz, Berlin (in German). Online http://www-agki.tzi.de/grp/ag-ki/download/2003/wache03.pdf. Last accessed March 1, 2004.
10. Wiederhold, G. "Mediators in the Architecture of Future Information Systems". *IEEE Computer*, 25(3): 38-49, 1992.