Extreme Programming System Metaphor: A Semiotic Approach

Rilla Khaled, Pippin Barr and James Noble
School of Mathematical and Computing Sciences
Victoria University of Wellington
Wellington, New Zealand
rkhaled, chikken, kjx@mcs.vuw.ac.nz

Robert Biddle
Human-Oriented Technology Laboratory
Carleton University
Ottawa, Canada
robert_biddle@carleton.ca

ABSTRACT

System Metaphor is one of the key practices of Extreme Programming (XP). Unfortunately, the System Metaphor practice is poorly understood, and is the practice XP teams most commonly choose to ignore. We provide a simple, structural model of system metaphors, based upon Peircean semiotics, giving a fundamental account of the way metaphors can contribute to a software system. Using this model, we identify activities that teams can use to develop metaphors for their systems, and techniques for evaluating system metaphors. We hope this analysis will encourage Extreme Programming teams not to abandon system metaphors, but rather, to continue to use metaphors to strengthen their development practices.

1. INTRODUCTION

One of the twelve core practices of Extreme Programming (XP) is the *System Metaphor*. In the glossary of *Extreme Programming Explained*, Kent Beck describes it as:

> *A story that everyone – customers, programmers and managers – can tell about how the system works* [8].

The system metaphor is a means of communicating about the project in terms that both developers and customers will understand, and which does not require pre-existing familiarity with the problem domain [9]. The system metaphor guides the mental models that project members have of the system, and shapes a logical architecture for the system.

Experience with XP shows that the system metaphor practice is the most commonly dropped practice, due to a lack of understanding of how to use it, and the difficulty of *finding* an appropriate metaphor [43, 9, 42]. Martin Fowler sums up a widely held sentiment about metaphor [10, 31, 45, 5] when he says the following in Chapter 1 of *Extreme Programming Examined*:

> *. . . I still don't think I've seen metaphor explained in a convincing manner. This is a real gap in XP, and one that the XPers need to sort out . . .*
>
> *. . . I still haven't got the hang of this metaphor thing. I saw it work, and work well, on the C3 project, but it doesn't mean I have any idea how to do it, let alone how to explain how to do it* [23].

In this paper, we set out to address these issues, our primary aim being how to *understand* system metaphor and secondarily how to *do* it. To understand system metaphors, we have analysed their *structure* using semiotics, the study of signs. In particular, we use techniques from Peirce, and Lakoff & Johnson to develop a formal semiotic model of metaphor in extreme programming. While the model itself is an advance for understanding and reasoning about metaphor, to make system metaphor more accessible on a practical level, we present a set of activities for finding potential system metaphors, and a set of criteria for evaluating them, based on our semiotic analysis but capable of independent application. XP teams can use these activities and criteria to support their development practices, thus profiting from our analyses, without necessarily having to appreciate the semiotic model upon which they are based.

Section 2 of this paper discusses the current state of the System Metaphor practice within the XP community, and existing suggestions for improvement. Section 3 contains a brief introduction to Peircean semiotics, a structural model of metaphor in general, and the application of semiotics in Computer Science. In section 4 we present our structural model of the XP system metaphor, detailing each component by applying it to the Chrysler C3 payroll system metaphor. Section 5 contains a list of process activities for establishing metaphors suitable for use by XP developers as well as various metaphor evaluation considerations. In section 6 we outline future directions for this work and finally in section 7 we present our conclusions.

2. EXTREME PROGRAMMING AND METAPHOR

The System Metaphor practice is a way of explaining the logical architecture of a system by describing it terms of something with which developers and customers are already

familiar [10, 9]. A system metaphor facilitates discussion of the project in language that is accessible to both customers and developers, providing a shared vocabulary for discussing system problems and solutions [5, 43]. For developers, a system metaphor additionally supports consistency in naming elements of their programs, including subsystems, packages, classes, and methods [17].

The paradigmatic XP system metaphor is the PAYROLL SYSTEM IS AN ASSEMBLY LINE metaphor used for the Chrysler C3 payroll system [24, 27]. This metaphor makes extensive use of manufacturing concepts, such as lines, stations, bins, and parts. The C3 metaphor works roughly in the following manner: a person's paycheck is a combination of *parts*, where parts initially relate to hours worked, e.g. basic gross pay. The parts move down the assembly line and are placed into *input bins*, which then supply these parts to *stations*. Each station works in sequence, and *processes* the parts, where processing consists of debiting or crediting further amounts to the initial amount, e.g. income tax, pension, overtime, union dues, and so forth. These processed parts are then placed into *output bins*, and in turn get processed by other stations. The final paycheck consists of an assemblage of all of the output parts resulting from each station in the assembly line [27].

The metaphor plays a role in shaping the "logical architecture" of the system. In *Extreme Programming Explained*, Beck gives explanations of how the metaphor shapes the architecture [8]:

> *The metaphor just helps everyone on the project understand the basic elements and their relationships. Words chosen to identify technical entities should be consistently taken from the chosen metaphor. As development proceeds and the metaphor matures, the whole team will find new inspiration from examining the metaphor.*

Consistent with the XP mind set of avoiding investment in the unknown, the system metaphor is a "cheap" system design, in that it suggests major system components and their interactions. Additionally, good metaphors have generativity, thus allowing people to broach new ideas and questions regarding the system they would not have otherwise raised [44]. Said the C3 developers on the topic of their system metaphor [24]:

> *The team had the benefit of a very rich domain model developed by members of the team in the project's first iteration. It gave the members of the project an edge in understanding an extremely complex domain.*

Yet for all the benefits of metaphors, in practice they are difficult to come up with and to use [43, 9, 42]. Unfortunately, there is little literature available on how to choose and use a metaphor — research on the system metaphor practice is quite sparse, especially when compared to the wealth of research considering other XP practices, such as Pair Programming, Test-Driven Development, or the Planning Game.

It seems that the typical approach for finding a metaphor is by a process of trial and error, to see if the suggested metaphor "fits". Wake also suggests combining metaphors if a single appropriate metaphor cannot be found or alternatively using the "naïve" metaphor, where the system stands for itself. He also suggests dropping the use of the metaphor if it stops working [43].

The system metaphor faces an even deeper problem however, which is the question of whether it really is of any help, especially if the chosen metaphor later turns out to be incorrect or unhelpful [23, 45, 5, 10]. Studies of projects using XP have shown not only that chosen metaphors are usually poor, but also that these poor metaphors are very underutilised during development [42, 25]; this is borne out in our own experience of XP projects in an educational environment [39]. Typical concerns are that the chosen metaphor is too weak, thus not providing any insight into potential architectural plans or providing enough vocabulary, or conversely that the chosen metaphor is too strong, thereby forcing system components into a form that they do not logically "mold" into. Maybe the chosen metaphor is too unfamiliar to the team members to provide any value (which may be the case if an earlier development team established the metaphor). Other causes of worry are that the metaphor is misleading and implies relations that do not exist, and also that the metaphor limits the conception of the system and provides no insight on how to deal with changes once the system needs updating [43, 44].

As well as its necessity being under scrutiny, metaphor is often overlooked. The *Extreme Programming Applied* authors Ken Auer and Roy Miller also seem to feel the same way in the chapter "Painting a Thousand Words":

> *A lot of people doing XP say they haven't really found a good metaphor or that they use metaphors only for certain parts of the system. All of the people we've talked to who don't use a metaphor haven't seen it as a significant problem* [5].

Within XP, then, experience with metaphor is somewhat mixed. Outside XP, however, metaphor is widely recognised as a fundamental part of communication. Not only is metaphor used extensively in literature, art, film and everyday speech, it is a tried and true learning technique which people use very frequently. To harness the benefits of metaphor, in this paper we set out to improve *understanding* of XP metaphor, by drawing upon existing theories of metaphor (in section 3) and then applying those analyses to the use of metaphor in extreme programming (in section 4).

3. SEMIOTICS

Agile software development is a relatively new methodology, forged out of the need for adaptivity, faster software delivery time and communication. In particular it stresses the need for face to face discussions between team members, and

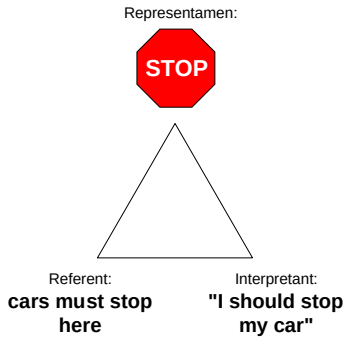

Figure 1: A diagram of the Peircean triad as applied to a stop-sign.

between team and customer instead of relying on formal documentation. For these reasons, semiotics seems like an ideal tool for studying and analysing XP system metaphor, as it is rigourous enough to facilitate structured analysis, while still retaining enough flexibility to recognise that multiple view points exist, or that views change over time.

By applying semiotics to XP metaphors, it is possible to harness a great amount of work done in the *general* field of semiotics. Here, we present only enough background to support our analysis of System Metaphor, as general introductions to semiotics are widely available [16, 15, 19, 18, 21].

Note that we do not expect XP developers to use semiotics directly to analyse their own metaphors. Rather, we will provide a structural model of the System Metaphor practice, which is grounded in semiotics. Currently there is *no* such way to view and understand metaphor, therefore the model will supply guidelines of sorts for developers to understand, develop, and evaluate the metaphors that they build into their systems.

3.1 Peircean Semiotics

Semiotics is the formal study of signs. According to Charles Sanders Peirce, one of the founders of semiotics, a sign is "*something which stands to somebody for something in some respect or capacity.*" [40, v.2 p.228]; more succinctly, Umberto Eco has defined a sign as "*something that stands for something else*" [19]. In other words, a sign can be almost anything — footprints, written words, spoken words, thoughts, images — anything which can mean something to someone.

Peirce proposed a triadic model of the sign. In his view, the sign is divided into three parts: the *object* or *referent*, the *representamen* and the *interpretant*. While Peirce made use of the term *object*, in this paper we shall use the term *referent* to avoid confusion with objects from object-oriented programs. Figure 1 shows the representamen, referent, and interpretant of a traffic "stop" sign.

As is shown in the diagram, the *representamen* is the actual embodiment of the sign. In this case, the sign in the world: a red octagon with the word "stop" written on it in white block letters. The representamen is the part of the sign that

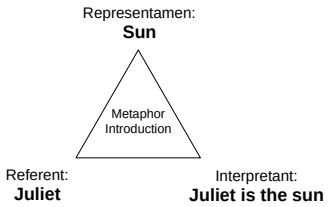

Figure 2: A semiotic model of metaphor introduction.

people encounter and attempt to understand the overall sign through.

The *referent* of the sign is the concept *cars must stop here*. This is the idea that the sign is meant to convey to those who encounter it. Other ways of expressing this are that the referent *represents* that concept or that it *refers* to that concept.

Finally, the *interpretant* of a sign is the thought or concept that occurs in an interpreter's mind when they encounter the sign. Thus, in the figure, the person who has encountered the sign correctly thinks that they should stop their car. There is no guarantee, of course, that this "correct" interpretant will be arrived at. The person could have thought something like "I should stop smoking my cigarette now." Context, convention, and law render this outcome unlikely, however.

One of the characteristics of Peircean semiotics that makes it particularly suitable for modelling metaphor is that interpretants can act as representamens for new signs. This type of process occurs very commonly as it takes place whenever people have "chains" of thoughts. Joined signs indicate the process of refinement of a conveyed concept.

3.2 The Semiotics of Metaphor

As a general figure of speech, "metaphor" has a reasonably broad scope of meaning. For example, *The American Heritage Dictionary of the English Language* defines metaphor as:

> *A figure of speech in which a word or phrase that ordinarily designates one thing is used to designate another, thus making an implicit comparison* [22]

More briefly, Lakoff & Johnson have defined metaphor as "*understanding . . . one thing in terms of another*" [30]. A metaphor sign involves the interaction, in some way, of the *tenor* and the *vehicle* of the metaphor, where the tenor is the thing or concept being described, and the vehicle is the thing or concept that is used to describe the tenor [20]. In the example JULIET IS THE SUN, Juliet is the tenor, and the sun is the vehicle.

We use the Peircean sign to model the parts of the metaphor, shown in figure 2. This *Metaphor Introduction* sign introduces the *metaphor* and places the vehicle and tenor into

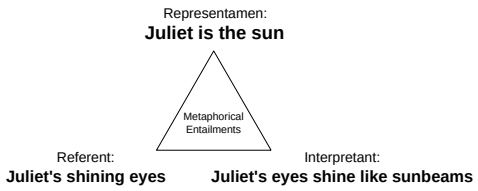

Figure 3: A semiotic model of metaphorical entailments.

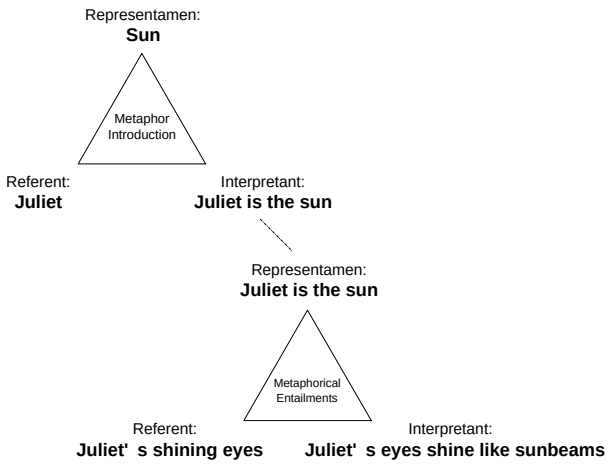

Figure 4: A semiotic model of metaphor showing one particular entailment

context. The *representamen* of the sign consists of the vehicle of the chosen metaphor, as it is easy to imagine that the vehicle "represents" the tenor. In the example, the sun is the representamen. The *referent* of the sign is the tenor of the metaphor, as it is the concept being referred to by the vehicle, and the relationship between the representamen and referent is one of reference. In the example, Juliet is the referent, as the sun represents, refers to, or stands for Juliet. Finally, the *interpretant* of the sign is the complete metaphor, as it is an interpretation that a viewer may arrive at upon encountering the representamen in the context of the referent. It is a *denotative* interpretant, which is to say that its meaning is embodied within its face value. The interpretant in our example ends up as "Juliet is the sun".

Lakoff & Johnson carried out substantial work in the area of metaphor, and introduced the concept of *metaphorical entailments* [30]. A metaphorical entailment is an application of some fact about the vehicle of the metaphor to the tenor of the metaphor. For example, if we consider the metaphor JULIET IS THE SUN, bearing in mind that it was Romeo who made this statement, one of its metaphorical entailments might be that "Romeo's world revolves around Juliet", because elements in solar systems revolve around the sun, and this quality gets transferred to Juliet. Another entailment could be that "Juliet gives Romeo life", as many facets of life are dependent on the sun. While a metaphor makes a *direct* comparison between tenor and vehicle, metaphorical entailments consist of all of the *indirect* resulting qualities we can deduce about the tenor based on the vehicle.

Semiotically, we model these entailments as instances of a second sign, the *Metaphorical Entailment* sign, depicted in figure 3. The *representamen* of the metaphorical entailment sign is the interpretant of the metaphor introduction sign, e.g. "Juliet is the sun". The *referent* of the metaphorical entailment sign is what the representamen *refers to* or *stands for*, typically some characteristic, distinguishing mark or trait of the tenor of the metaphor. The characteristic in this example is "Juliet's shining eyes". The *interpretant* of the metaphor entailment sign is the result of reflecting upon the metaphor, which (in this case) could be part of the meaning Romeo ascribes to the metaphor. This entailment is typically closely tied to a characteristic embodied in the referent. For example, "Juliet's shining eyes", which is a characteristic of Juliet in the opinion of Romeo, is closely related to the statement "Juliet's eyes shine like sunbeams", which is an entailment of the metaphor.

The difference between the characteristic and the entailment is that while the characteristic describes the tenor independently of the vehicle, the entailment explicitly relates qualities of the vehicle *to* the tenor. To contrast the interpretant of this sign with the interpretant of the Metaphor Introduction sign, this interpretant is *connotative*, which means that its meaning is dependent on cultural and personal associations.

Although the figure shows only a *single* characteristic as the referent and a *single* resulting metaphorical entailment, this does not mean that the metaphor just refers to this specific characteristic and entailment. In fact, any other characteristic of Juliet could have replaced the one in the figure. Semiosis (the process of interpreting signs) can be carried out repeatedly within the metaphor, each time with a different referent, or characteristic and resulting entailment. In the example, from the viewpoint of Romeo, one characteristic of Juliet is "Juliet's shining eyes". Equally, the characteristic could have been "Juliet's beauty". If the referent of the sign had indeed been "Juliet's beauty", then the resulting entailment might be "Juliet is radiantly beautiful" as radiance is commonly associated with the sun and its rays.

To summarise, a metaphor begins by describing its tenor in terms of its vehicle, with this implied comparison giving rise to a range of metaphorical entailments. Both the metaphor itself and the entailments can be taken as signs, which we call the metaphor introduction and metaphor entailment signs respectively. The two signs are linked by the chain of semiosis, as the interpretant of the metaphor introduction sign becomes the representamen for the metaphorical entailments. Figure 4 shows our complete structural model of metaphor. Our model shows how metaphors convey meaning: a representamen is used to describe a referent, and they become bonded as an interpretent; this bonding in turn allows characteristics of the original referent to become interpreted in new ways.

3.3 Computing Semiotics

Given that questions of representation and interpretation undergird much of computer science and software engineering, there has been surprisingly little direct application of semiotics to these areas. Much of the research that has been done in this area has been directed to the design and analysis of user interfaces, because these are the most visibly sign-intensive parts of a computer system [7, 36].

3.3.1 User-Interface Semiotics

Metaphors are a very popular approach to user-interface design [11, 14, 13], since being popularised by the Xerox Star and Apple Macintosh [28, 4]. A user-interface metaphor explains some system functionality or structure (the tenor) by asserting its similarity to another concept or thing already familiar to the user (the vehicle). The key to UI metaphors is that the chosen vehicle is something already *familiar* to the user and so the intention is to provide a base level of comfort and knowledge without necessarily understanding the underlying system.

We have conducted a semiotic analysis of user-interface metaphors, again relying on Peircean semiotics [6]. The resulting semiotic structure is very similar to that of figure 4, with a Metaphor Introduction sign giving rise to a series of Metaphorical Entailments. For user interface metaphors, of course, the representamen (vehicle) of the metaphor is typically a graphical icon, such as a file folder or trashcan, while the referent (tenor) of the metaphor is the system function they seek to present to users, such as storing or deleting data.

3.3.2 Semiotics of Programming

Peter Bøgh Andersen and colleagues have conducted extensive work focusing primarily on computer systems in general, as well as on interface design. In his major work, *A Theory of Computer Semiotics*, Andersen develops an extensive theory on how semiotics can be applied to all aspects of computing [2]. The book includes an extensive case study of the application of his methods to a software system for a post office. Other important work by Anderson focusses on whether semiotics is a good approach to human-computer interaction at all [3]. Semiotics has provided the inspiration behind some specialised traditional (i.e. non-Agile, non-Extreme) information systems development and analysis methodologies. Liu and Stamper, for example, describe system design techniques explicitly based on semiotics [33, 34]. Similarly, Marcelo Pimenta and Richard Faust have taken a semiotic approach to requirements gathering [41].

We have also applied semiotics more generally to programming, in particular to the analysis of design patterns. As code structures that stand for design ideas, patterns can be treated directly as signs [37] and this can lead to an effective categorisation scheme for patterns [38].

3.3.3 An Object-Oriented Programming Sign

Semiotics can also be used to describe object-oriented programming [1, 38]. Within the Scandinavian approach, as described by Ole Lehrmann Madsen, Birger Møller-Pedersen and Kristen Nygaard in *Object-Oriented Programming in the BETA Programming Language*, programming is seen as modelling the world:

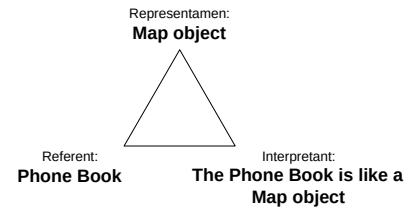

Figure 5: An OOP sign for an Address Book object

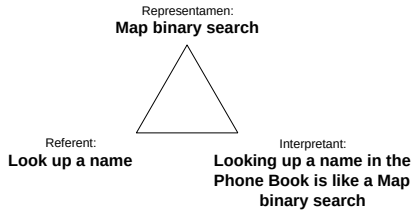

Figure 6: An OOP sign for a LookUp method

> *A program execution is regarded as a physical model, simulating the behaviour of either a real or imaginary part of the world* [32].

Programs are often written in a way that reflects similarities in state and behaviour between program objects and their users' or customers' domains; certainly programmers have been taught to write their programs in this way for quite some time [29, 12].

Eyoun Eli Jacobson explains this type of OO modelling as using objects and classes featuring elements and references within the system and representational world, to represent relevant phenomena and concepts exhibiting similar elements and references within the system and conceptual world [26]. For example, consider a Phone Book object, where a Map object is being used to implement a telephone directory. The Map object represents the Phone Book, while the binary search used within the Map represents looking up something in the book.

We can describe this relationship using semiotics, as shown in figures 5 and 6. We call these signs *"Object-Oriented Programming"* signs, because they capture the basic semiosis that is typically latent in object-oriented programming. In each of these signs, the *representamen* is the program element standing in as a representation (the map object or its lookup method), the *referent* is the domain concept being represented (phone book or lookup) and the *interpretant* of the sign establishes the representational relationship between the program object and the external object — the idea that this particular map object represents that particular phone book; that this particular binary search lookup method represents searching the phone book.

4. A STRUCTURAL MODEL OF THE XP SYSTEM METAPHOR

Extreme Programming system metaphors are, first of all, metaphors, that is, they describe one thing in terms of another. XP system metaphors are a *specialised* case of

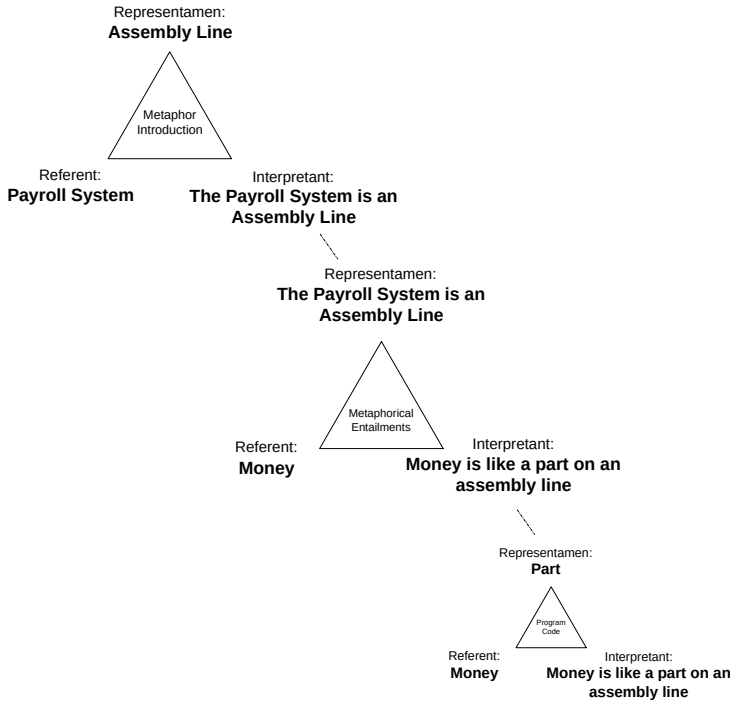

Figure 7: A structural model of XP metaphor showing one particular entailment

metaphor however, in that they specifically serve to describe *object-oriented software systems*. We therefore model XP system metaphors by combining our semiotic models of general metaphors and of object-oriented systems, as shown in figure 7. This model of XP system metaphors consists of three interrelated signs. The first sign introduces the parts of the metaphor; the second sign deals with the entailments of the metaphor; and the third sign represents programming constructs resulting from the metaphorical entailments. This section explains how our model works with respect to an actual system metaphor, the Lines-Parts-Bins-Stations metaphor used in the C3 system [24].

4.1 The Metaphor Introduction Sign

The first sign is the Metaphor Introduction sign (see figure 7). The *representamen* of this sign is the vehicle of the system metaphor as a whole. For example, the C3 system metaphor name is "Assembly Line", so this is the representamen. The *referent* of the Metaphor Introduction sign relates to the problem domain of the system being built, so the referent for the C3 project is "Payroll System" which is the tenor of the system metaphor. Finally, the *interpretant* of the Metaphor Introduction sign is the complete metaphor of the form The Payroll System is an Assembly Line, because this is the interpretation a team member makes upon exposure to the metaphor.

4.2 The Metaphorical Entailments Sign

The second, Metaphorical Entailments sign, models the entailments of the metaphor introduced by the first sign (again, see figure 7). The *representamen* of this sign comprises the complete system metaphor and is the interpretant from the Metaphor Introduction sign — in this case, The Payroll System is an Assembly Line. The *referent* for the Metaphorical Entailments consists of a *characteristic* of the tenor of the metaphor, that is, one of its important components or operations. In the Payroll system example, potential characteristics include "money", "paychecks", and the "pension deduction" operation.

As with the structural model for general metaphor, although there are many potential characteristics, we focus on one characteristic at a time. The *interpretant* for the Metaphorical Entailments sign consists of a comparison of a fact or a concept we associate with assembly lines to the Payroll system, resulting in a statement relating some aspect of assembly lines to the Payroll system, its components, or its functionality. For example, a major concept behind the Payroll system is that paychecks are simply some combination of time, money, and additional adjustment factors. Thinking of potential entailments of the system metaphor upon the Payroll system, one possible interpretant might be "money is like a part on an assembly line". Another might be "a paycheck can be assembled from time and money" and yet another might be "a pension deduction is like a station task on an assembly line". Each of these interpretants will in turn give rise to what we call a *Program Code Sign*, which is the result of combining concepts from the system metaphor with object-oriented programming.

4.3 The Program Code Sign

The Program Code sign, then, is a special case of the OOP sign model presented in the section 3.3.3. As shown in figure 7, each Program Code sign represents an entailment of the metaphor upon a system component or operation. Program Code signs are ultimately embodied within the actual code.

This realisation of metaphorical entailments is described in-

formally in the XP literature. Wake writes in Chapter 5 of *Extreme Programming Perspectives*:

> *...By looking at the objects behind the metaphor, and especially the interactions between those objects, we can get insights into how our system does work and how it should work* [44].

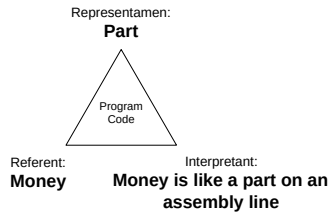

Figure 8: The money part model

The *representamen* of a program code sign, then, is an interpretant from a metaphorical entailment sign. The *referent* of the sign refers to the part of the domain that the concept is being applied to, and the *interpretant* is the result of the application of the metaphor-inspired concept to the domain. Figure 8 shows one example of a Program Code sign from the assembly line metaphor. The representamen of the depicted programming concept model is "Part" and the referent is the domain concept "Money". The interpretant becomes "Money is like a part on an assembly line." This interpretant in fact embodies the meaning of the Program Code sign, which corresponds to the interpretant of a Metaphorical Entailments sign.

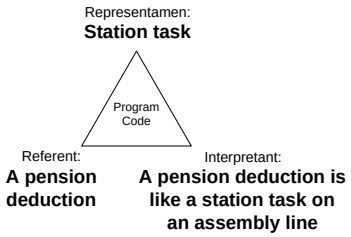

Figure 9: The deduction station model

Figure 9 shows another example of a Program Code sign from the Payroll system. The representamen is "station task", the Payroll domain operation "Pension Deduction" is the referent and the interpretant becomes "A pension deduction is like a station task on an assembly line."

The Program Code signs represent the level at which the metaphor supplies ideas that are directly usable for the system, i.e. the coding level. If the metaphor is rich it will yield a large number of Program Code signs, which in turn provide a vocabulary for describing a domain, and also propose a logical architecture for the system.

5. CHOOSING AND EVALUATING A SYSTEM METAPHOR

The semiotic structural model of system metaphor is an important contribution to the underlying rationale of Extreme Programming, because it makes clear how metaphors work to describe systems. In fact, the model also suggests further ideas, regarding how to *choose* a metaphor as well as techniques for how to *evaluate* a metaphor.

In this section we address these two issues, grounding the discussion with the simple running example: A Bank Account is a Water Reservoir.

5.1 Choosing a metaphor

Choosing a system metaphor appears to be a rather "hit and miss" art form at present. Based on the structural model, we propose a series of activities for supporting XP teams carrying out the Metaphor practice.

5.1.1 Brainstorming potential metaphors

The top-level sign in the structural model represents the system metaphor. The task of initially establishing a metaphor is, given a system to be built (the tenor of the metaphor, the *referent* of the Metaphor Introduction sign), to find a suitable vehicle to describe it (the *representamen* of the sign), that will lead to useful metaphorical entailments (*interpretants*).

The first of stage choosing a metaphor, then, is brainstorming a set of potential metaphors, involving the whole XP team, and especially including the customer representatives. Apart from randomly sparked suggestions, to help inspire potential metaphors, we suggest thinking of ways to explain the system and its intended functionality to an audience of *non-experts*. For example, a non-expert description of a bank account might be "it is a container; different parties can add or remove the contents of the container; and the quantity of the substance in the container is important". A next step might be to imagine possible "surrounding" metaphors that account for this explanation. Some metaphor possibilities for a bank account may include water reservoir, parking lot, office, city, and hospital. Once a sufficient number of suggestions has been made, we suggest choosing three candidate metaphors by group consensus which intuitively seem the most promising. For example, the most suitable metaphors for describing the bank account could seem to be the water reservoir, parking lot, and hospital metaphors.

5.1.2 Brainstorming the entailments of the metaphors

The second level of the structural model represents the entailments of the metaphor. This is the stage at which it begins to be revealed whether or not a metaphor will be applicable to a particular system. Exploring each candidate metaphor relies on semiosis of the Metaphorical Entailments sign and individual entailments — in terms of the model, finding the *interpretants* and *referents* of the Metaphorical Entailments Sign based on the signs representing the whole candidate metaphors.

Continuing from the last activity, we propose brainstorming the entailments of each of the top three metaphors first individually, then again with the entire development team, so that people are able to come up with their own ideas and then think of new ideas inspired by suggestions of other team members. One approach is to try focussing on one part or

characteristic of the system, and then considering it in the light of the metaphor. Considering A BANK ACCOUNT IS A WATER RESERVOIR, possible entailments are that the bank balance relates to the water level; a deposit is like inflow of water; an overdraw limit is like a minimum water level; account management is like water treatment; and so on. This technique can also be applied in reverse, thinking about a potential entailment of the metaphor and finding parallels between the entailment and characteristics of the system to be built. Using the same example as above, the water level of a reservoir rises and falls, just as a bank balance does; and the reservoir needs to monitored to maintain a certain degree of water quality, just as bank accounts need a high level of security management. Less sensible entailments include: since water reservoirs are required to periodically change their water, money should therefore be periodically transferred to another account; and, since water reservoirs also have a maximum capacity to prevent flooding of surrounding land, bank accounts should always contain less than some maximum balance.

After brainstorming the entailments of each candidate metaphor, the team should then be in a position to evaluate each candidate, and therefore choose the metaphor best suited to the next phase of the project.

5.1.3 Culling the group list and finding the winner

The third level of the structural model deals with Program Code signs — that is, with the program that will actually be written. Only some of the suggested entailments will lead to sensible programs, however, and these entailments need to be identified as a team so that everyone has the same understanding of how the metaphors works *for* the system, as well as *how far* the metaphors works.

So, we would encourage the group to evaluate the entailments for each candidate metaphor (as described in the next section), and "cull" the lists for *correctness, consistency* and *coherency* by crossing off entailments that are incorrect with respect to the workings of the system, inconsistent with each other or focus on irrelevant details. Furthermore, the group should keep only the entailments which support and are consistent with the known workings of the system. Working with the list of entailments generated for the water reservoir metaphor, the money transferral entailment and the maximum balance entailment are clearly incorrect, so they should be culled.

At this stage it should be clear how good the three metaphors are with respect to providing a useful vocabulary for the system and explaining the known system components and functionality, while not implying non-existent behaviour. If there is one metaphor that clearly achieves these tasks, it should probably be adopted as the system metaphor. If there is no clear winner and all of the metaphors have their strengths and their weaknesses, we suggest using a combination of more than one metaphor for the system.

5.2 Evaluating a metaphor

Although Metaphor is a key practice of XP, currently there is little guidance about how to evaluate potential metaphors. As we discussed in section 2, amongst XP practitioners it seems that a common way to evaluate metaphors is to keep using them until they seem to stop working. In this section, we present six criteria, derived from the semiotic model, that can be used to evaluate a metaphor — either evaluating a candidate when choosing a metaphor for the first time, or evaluating a metaphor in an existing system.

5.2.1 How is the metaphor good?

Various metaphors can be helpful in different ways. One of the first questions to arise when deciding if a metaphor is good is **whether the entailments of the metaphor contain programmable ideas**, in other words, ideas that can inspire actual code. The Program Code signs inspired by a metaphor should be consistent across a metaphor, to ensure the overall coherency of the metaphor and consequently increase the likelihood of a *shared* understanding of it. In turn, the shared understanding should lead to more coherent, consistent and simple code. If more than one metaphor is used, the consistency applies to *each* metaphor individually. Using the Bank Account example, potentially useful entailments are "Account management is like water treatment" and "An account balance is like the water level".

A second criterion is **whether the metaphor addresses the major system components and their known functionality**. This can be checked by examining the Program Code signs resulting from the metaphor to see whether Program Code signs exist for the major components and also whether the important functionality is described in a Program Code sign. Program Code signs may need to be broken down into smaller models before this can be checked. This criterion is similar to the first in that it also probes the relationship between the metaphor and the system, but it differentiates itself from the first in the level of detail at which the metaphor *represents* the system. From another perspective, while the first criterion is examining an overall "story" that the metaphor suggests, this criterion focusses on "characters". In the Bank Account example, major system components and functionality include *bank accounts, deposits, withdrawals, overdraft*, and *account management facilities*. Some corresponding concepts in the Water Reservoir metaphor are *reservoir, inflow, outflow, minimum water level*, and *treatment*.

Sometimes a metaphor can still be helpful even if it does not fully address either of the above concerns. Considering the metaphorical entailments, a third criterion for a good metaphor is **whether the metaphor entailments provide a vocabulary with which to describe the system**. While the entailments themselves do not necessarily deal with concrete programming issues, they provide a way of describing the workings of the system, which is useful for communication amongst the *entire* development team, which is to say developers *and* clients. For example, some useful language and concepts that the Water Reservoir metaphor provides are *reservoir, water supply, inflow, outflow, minimum water level, treatment*, and *water quality*.

5.2.2 Is the metaphor too poor?

Unfortunately, rich metaphors addressing all major system components and known functionality are few and far between. Often combined metaphors make up for this lack of an all-encompassing metaphor. Yet in order to choose *which*

metaphors to combine, some amount of attention should be paid to the disadvantages of each metaphor.

A fourth criterion, then is to examine the metaphorical entailments to see **which system components and interactions are left undescribed by the metaphor**, in other words, which parts of the system the metaphor does not explain. To avoid confusion, each undescribed component or required functionality should be addressed either by another metaphor or by means of explicit description. A metaphor that is not contributing much may need to be dropped, as too many metaphors complicate the shared understanding of the system.

For example, the Water Reservoir metaphor uses the idea of an inflow to represent a deposit. But while a deposit into an account is a *discrete* event, water inflows for a reservoir take place *continuously* from surrounding water sources, other reservoirs or perhaps storm water, therefore the idea of a discrete deposit is left undescribed. Furthermore, while the party that deposited money could be *anyone* in the case of a bank account, in a water reservoir, the water sources are very restricted.

5.2.3 Is the metaphor misleading?

In contrast to poor metaphors, some metaphors describe too much. Metaphors of this type are divisible into two kinds. Metaphors of the first group simply cover too much ground, and are addressed by the fifth criterion, considering **whether the metaphorical entailments imply non-existent system components or non-existent behaviour**. While it is almost impossible to avoid this to some extent, the metaphor should not imply *too much* that is incorrect. As every incorrect entailment is excluded from the metaphor, progressively the *intuitability* of the metaphor decreases, where intuitability describes the extent to which behaviour and components can be guessed or predicted. In turn this makes the metaphor less helpful. One example of non-existent behaviour implied by the Water Reservoir metaphor is that when the bank account balance reaches a certain level, it will be unable to accept more money, given that water reservoirs have a maximum capacity and risk flooding if they become too full. Another example of non-existent behaviour is that money must be "thrown away" periodically to keep the remaining money "fresh", given that water in reservoirs is drained every so often to maintain a certain level of water quality.

The second sort of misleading metaphor is characterised by being overly strong, and is addressed by the sixth (and last) criteria: **whether the metaphorical entailments make the system more complicated than it needs to be**. Metaphors exist to *enhance* system understanding, not to obscure it. For example, a water reservoir typically needs pumping stations, turbidity monitoring, filtering, the addition of chlorine and fluorine, earthquake protection, and involves concepts such as water pressure, salinity, quality, and so on — none of which has any obvious counterpart in the world of banking; and efforts, say, to add "money pumps" to a banking system to keep the "money pressure" high enough would be quite misguided. The heart of XP is a simple and flexible system architecture, therefore an overly complicated metaphor should be avoided at all costs. It is not clear that the Water Reservoir metaphor really enhances *any* understanding of the system, as all it seems to offer is a set of synonyms for some system components and behaviour, and yet it does not provide a full set. Furthermore, the team probably has more knowledge of bank accounts than they have of water reservoirs.

6. FUTURE WORK

So far we have successfully applied our model to certain XP metaphors to obtain useful analyses. The metaphor choosing and evaluation techniques have also proven themselves to be useful. We plan to conduct further case studies with different metaphors, ranging from metaphors originating from completed projects, ongoing projects and "example" metaphors described in XP texts. Our investigation will take place in an academic setting, both as research and as part of one of our 3rd year undergraduate computer science cources, and also within industry, upon XP teams *and* customers, as some of our other recent work has concerned the role of the XP customer [35]. Another plan for this work is to develop the metaphor choosing process activities into a set of patterns.

7. CONCLUSION

On the topic of metaphor, Martin Fowler says the following [23]:

> *"This is a real gap in XP, and one that the XPers need to sort out".*

In this paper, we have attempted to close this gap. Our major contribution is a structural model for the XP system metaphor, based on Peircean semiotics. This model considers metaphor as three distinct signs: the denotative metaphor introduction; the connotative metaphorical entailments; and the resulting program code. The semiotic model provides a rigourous way to *examine* the system metaphor and its subsequent parts. It imparts a structure to our reasoning about metaphor, which as a consequence improves our *understanding* of metaphor's workings.

Based on the structural model, we suggest a series of activities for establishing and refining metaphors to support XP teams. We also developed six evaluation criteria for system metaphors that focus on the identification of *good*, *poor*, and *misleading* metaphors. We hope this analysis, and the techniques based upon it, will aid Extreme Programming developers in their use of the System Metaphor practice, and consequently improve the systems built using that practice.

8. REFERENCES

[1] P. B. Andersen. A Semiotic Approach to Programming. In *The Computer as Medium*, pages 16–67. Cambridge University Press, 1993.

[2] P. B. Andersen. *A Theory of Computer Semiotics.* Cambridge Series on Human-Computer Interaction. Cambridge University Press, 1997.

[3] P. B. Andersen. What Semiotics Can and Cannot Do for HCI. In CHI'2000 Workshop on Semiotic Approaches to User Interface Design., 2000.

[4] Apple Computer, Inc. Staff. *Macintosh Human Interface Guidelines.* Addison-Wesley, 1992.

[5] K. Auer and R. Miller, editors. *Extreme Programming Applied*, chapter 23: Overtime Is Not the Answer. Addison-Wesley, 2002.

[6] P. Barr. A Semiotic Model of User-Interface Metaphor. In *Virtual, Distributed and Flexible Organisations: Studies in Organisational Semiotics - 3*. The 6th International Workshop on Organisational Semiotics: Virtual, Distributed and Flexible Organisations, 2003.

[7] P. Barr, R. Biddle, and J. Noble. Icons R Icons. In *User Interfaces 2003: Fourth Australian User Interface Conference*, pages 25–32, 2003.

[8] K. Beck. *Extreme Programming Explained*, chapter Glossary. The XP Series. Addison-Wesley, 2000.

[9] K. Beck. The Metaphor Metaphor. Invited speaker at OOPSLA, 2002.

[10] K. Beck, A. Cockburn, and L. Bossavit. System Metaphor. `http://c2.com/cgi/wiki?SystemMetaphor`, 2003. (Various other anonymous authors.).

[11] A. Blackwell. *Metaphor in Diagrams*. PhD thesis, University of Cambridge, September 1998.

[12] D. W. Brown. *An Introduction to OBJECT-ORIENTED ANALYSIS: Objects and UML in Plain English*. John Wiley & Sons, 2002.

[13] J. M. Carroll, R. L. Mack, and W. A. Kellogg. Interface metaphors and user interface design. In M. Helander, editor, *Handbook of Human-Computer Interaction*, pages 67–85. Elsevier Science Publishers, 1988.

[14] J. M. Carroll and J. C. Thomas. Metaphor and the cognitive representation of computing systems. *IEEE Transactions on Systems, Man. and Cybernetics*, 12(2):107–116, March/April 1982.

[15] P. Cobley, editor. *The Routledge Companion to Semiotics and Linguistics*. Routledge, London, 2001.

[16] P. Cobley and L. Jansz. *Semiotics for Beginners*. Icon Books, Cambridge, England, 1997.

[17] W. Cunningham. System Of Names. `http://c2.com/cgi/wiki?SystemOfNames`, 2003.

[18] A. Easthope and K. McGowan, editors. *A Critical And Cultural Theory Reader*. Allen & Unwin, 1992.

[19] U. Eco. *A Theory of Semiotics*. Indiana University Press, 1976.

[20] U. Eco. *Semiotics and the Philosophy of Language*. Indiana University Press, 1986.

[21] A. Edgar and P. Sedgwick, editors. *Key Concepts in Cultural Theory*. Routledge, London, 1999.

[22] A. H. Editors, editor. *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company, 4th edition, 2000.

[23] M. Fowler. *Extreme Programming Explained*, chapter 1: Is Design Dead? The XP Series. Addison-Wesley, 2001.

[24] R. Garzaniti, J. Haungs, and C. Hendrickson. Everything I Need to Know I Learned from the Chrysler Payroll Project. In *SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum)*, pages 33–38. ACM Press, 1997.

[25] J. Herbsleb, D. Root, and J. Tomayko. The eXtreme Programming (XP) Metaphor and Software Architecture. Technical report, Carnegie Mellon University, 2003.

[26] E. E. Jacobsen. *Concepts and Language Mechanisms in Software Modelling*. PhD thesis, University of Southern Denmark, 2000.

[27] R. Jeffries. Lines Stations Bins Parts. `http://c2.com/cgi/wiki?LinesStationsBinsParts`, 1999.

[28] J. Johnson, T. L. Roberts, W. Verplank, D. C. Smith, C. Irby, M. Beard, and K. Mackey. The Xerox Star: A retrospective. *IEEE Computer*, 22(9), 1989.

[29] G. Korienek and T. Wrensch. *A Quick Trip To Objectland*. Prentice-Hall, 1991.

[30] G. Lakoff and M. Johnson. *Metaphors We Live By*. The University of Chicago Press, 1980.

[31] P. Lappo. No pain, no XP: Observations on teaching and mentoring extreme programming to university students. In *Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*, pages 35–38. Universitá di Cagliari and Free University of Bolzano-Bozen, 2002.

[32] O. Lehrmann Madsen, B. Møller-Pedersen, and K. Nygaard. *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley, 1993.

[33] K. Liu. *Semiotics in Information Systems Engineering*. Cambridge University Press, 2000.

[34] K. Liu, R. J. Clarke, P. B. Andersen, and R. K. Stamper, editors. *Organizational Semiotics: Evolving a Science of Information Systems, IFIP TC8 / WG8.1 Working Conference on Organizational Semiotics: Evolving a Science of Information Systems, July 23-25, 2001, Montréal, Québec, Canada*, volume 227 of *IFIP Conference Proceedings*. Kluwer, 2002.

[35] A. Martin, J. Noble, and R. Biddle. Being Jane Malkovich: A Look Into the World of an XP Customer. In M. Marchesi and G. Succi, editors, *Extreme Programming and Agile Processes in Software Engineering, 4th International Conference, XP 2003, Genova, Italy, 2003 Proceedings*, Lecture Notes in Computer Science. Springer, 2003.

[36] M. Nadin. Interface design: A semiotic paradigm. *Semiotica*, 69(3):269–302, 1988.

[37] J. Noble and R. Biddle. Patterns as Signs. In B. Magnusson, editor, *16th European Conference on Object-Oriented Programming*, pages 368–391, 2002.

[38] J. Noble, R. Biddle, and E. Tempero. Metaphor and metonymy in object-oriented design patterns. In *Proceedings of Australian Computer Science Conference (ACSC)*. Australian Computer Society, 2002.

[39] J. Noble, S. Marshall, S. Marshall, and R. Biddle. Less extreme programming. In *The Proceedings of the Sixth Australasian Computing Education Conference (ACE2004)*. Australian Computer Society, 2004.

[40] C. S. Peirce. *Collected Papers*. four volumes. Harvard University Press, 1934–1948.

[41] M. S. Pimenta and R. Faust. HCI and Requirements Engineering - Eliciting Interactive Systems Requirements in a Language-Centred User-Designer Collaboration: A Semiotic Approach. *SIGCHI Bulletin*, 29(1), January 1997.

[42] J. Tomayko and J. Herbsleb. How Useful Is the Metaphor Component of Agile Methods? A Preliminary Study. Technical report, Carnegie Mellon University, 2003.

[43] W. C. Wake. *Extreme Programming Explored*, chapter 6: What is the System Metaphor? The XP Series. Addison-Wesley, 2002.

[44] W. C. Wake and S. A. Wake. *Extreme Programming Perspectives*, chapter 5: The System Metaphor Explored. The XP Series. Addison-Wesley, 2003.

[45] D. West. Metaphor, Architecture, and XP.