From: Timothy Flynn [e-mail redacted]
Sent: Monday, September 27, 2010 9:55 PM
To: Bilski_Guidance
Subject: Request for comments regarding subject matter eligibility

In response to the USPTO's solicitation of public comment concerning *Interim Guidance for Determining Subject Matter Eligibility for Process Claims in View of Bilski v. Kappos ,* as an experienced practitioner and independent developer in the field of software engineering I would like to offer the following comments. These comments are intended as a response to the third question posed in the solicitation which I repeat here for reference :

> *3. The decision in Bilski suggested that it might be possible to ''defin[e] a narrower category or class of patent applications that claim to instruct how business should be conducted,'' such that the category itself would be unpatentable as ''an attempt to patent abstract ideas.'' Bilski slip op. at 12. Do any such ''categories'' exist? If so, how does the category itself represent an ''attempt to patent abstract ideas?''*

It is my contention that there does indeed exist a broad category of patent applications which constitute attempts to patent abstract ideas. Further the USPTO has made a practice of granting such patents over the past several years. This is the category of software patents. A few specific examples of patents which fall into this category are the following :

- Patent Number 6,711,293 *Method and Apparatus for Identifying Scale Invariant Features in an Image and Use of Same for Locating an Object in an Image*
- Patent Number 5,960,411 *Method and System for Placing a Purchase Order via a Communications Network*
- Patent Number 6,941,275 *Music Identification System*
- Patent number 6,125,447 *Protection Domains to Provide Security in a Computer System*

These patents, and many others like them represent attempts to patent abstract ideas, specifically mathematical algorithms and processes for manipulating abstract data. To see that this is true we shall need to develop an understanding of the nature of abstraction. Abstraction is one of the principle foundations on which mathematics and its embodiment in computer software is based.

Let's begin with the simplest of mathematical concepts, that of a natural number: 1, 2, 3, etc. Such a number represents a mathematical concept known as an equivalence class. An equivalence class is a set of objects defined by some property, rule or condition. The number 2 represents the equivalence class of all sets which have 2 elements. This is the fundamental operation of abstraction. It allows us to now think in terms of the abstract concept 2 instead of having to consider all sets with 2 elements as distinct unrelated things. The abstraction does not represent all knowledge about all sets with 2 elements but rather represents that property which all such sets share.

Abstraction is a hierarchical process. It does not stop at one level. Rather higher level abstractions are built on top of lower level abstractions which are in turn constructed on still lower level abstractions and so on until one arrives at the level of concrete physical things which exist in the real world. Such physical things are no longer abstract, but everything else is.

Two other notions closely related to that of abstraction are that of *concept* and that of *representation*. We will take the term concept to refer to the most general thing which can be imagined, discussed or processed. Concepts in and of themselves have no concrete physical reality in the physical world (ie. they are abstract). In order for a concept to acquire some manner of existence it must be represented, ultimately in some concrete physical form. When a concept exists only within the mind of a person we will speak of a *mental representation*. Though the detailed nature of such mental representations is not yet well understood we can assume that it involves some combination of dynamic electrical and chemical signals within the brain. In order for a concept to enter into the shared consciousness of multiple persons it must be transformed into some other physical representation such as sound waves corresponding to some spoken utterance or photons reflected off of writing on a piece of paper. We will refer to this type of representation as a *linguistic representation*. Similarly in order to be processed by a computer system the concept must again be translated into some representation (again involving dynamic electromagnetic signals) which can be manipulated by the hardware of the computer. This we will refer to as a *computational representation*.

Now in this chain of events where a concept is transformed first from a mental to a linguistic then finally to a computational representation the concept itself retains its identity. To take a specific example consider the concept whose linguistic representation is given by the word *dog*. In the brain this concept is represented through some set of neural signals. When spoken it is represented by vibrations of gas molecules and when stored in computer memory it is represented by a collection of switch states which represent the sequence of numbers 100, 111, 103 in base 2 (these numbers represent the encoding of the characters d, o and g in the most commonly used character encoding). Despite the great dissimilarity between these 3 physical representations, they all refer to the same concept. This is the essence of abstraction. We agree through some convention that these 3 representations actually refer to the same abstract concept.

A very important point to understand is that such abstractions do not have meaning solely through some human caprice or convention. Rather mathematics allows rigorous statements to be made about such abstractions which can give them a sort of reality which reminds us of certain aspects of physical reality while retaining significant differences with respect to that domain.

One such rigorous mathematical statement or theorem which has proved to be of the utmost importance to the development of digital computers holds (stated informally) that there exists a certain class of mathematical objects, known as Universal Turing Machines (UTM's), which possess the property of being able to perform any computation when provided with the appropriate set of instructions. It is possible to construct many different physical representations of these UTM's. In other words there exist many physical systems which when configured with the appropriate set of initial

conditions and assuming an appropriate mapping between physical variables and mathematical objects are able to simulate the behavior of UTM's and hence perform any computation which can be performed (provided the size of the problem does not exceed the memory capacity of the physical system). One such physical system is the CPU of a modern computer. The role of a computer is therefore not to be a physical machine as such but rather to be a physical representation of a mathematical abstraction.

Computer software constitutes the sets of instructions which are needed to allow computer hardware to perform any computation. This software, being an abstract concept, also exists in different forms or representations. At one level of abstraction it is a sequence of numbers which when fed into the computer cause it to perform the desired computation. This representation is that of machine executable object code. Programmers rarely deal with this representation. Instead they create software using a language which, though its meaning is rigorously defined, uses common English words and punctuation to represent the instructions which, after an appropriate change of representation, will be fed to the computer. This representation, known is source code, is thus a much more abstract representation of the already abstract object code representation. At this level we also see that a computer program is nothing more than a set of instructions which could, in principle, be performed by a human being with pencil and paper.

It is therefore clear that any computer software is an abstract mathematical concept and hence, since an idea is nothing more than the mental representation of a concept, an abstract idea.

In light of this argument one is left to wonder how there could be such disagreement on the abstract nature of computer software. I think the misunderstanding can be traced to several causes. First the subject matter of mathematics is not always understood to encompass the totality of all abstractions. People often think of mathematics as being limited to the study of numbers and geometry when in reality it is that of abstract concepts. Mathematics should not be distinguished from other forms of abstract human reasoning on the basis of its subject matter but rather on the basis of the rigour that it affords. This definition is not arbitrary for any attempt to limit the subject matter of mathematics to certain classes of abstractions fails because mathematical statements about the allowed subset are found to imply equally rigorous statements about a larger set of abstractions once appropriate concept mappings are defined. This misconception has led some to deny the fact that all computer programs are in fact mathematical objects.

Another source of error is that to the uninformed computer software does not seem abstract. As I write these lines I am interacting with a software program which seems very real to me. It possesses a quality of predictability and definiteness similar to that possessed by real physical things. This predictability is however not the result of the software being a real physical thing but rather of the rigorous nature of mathematical reasoning on which the program, as an abstract mathematical object, is based.

Timothy G. Flynn
Endicott, NY