Code as a Self-contained Patent Algebra

There are two reasons shareware can be considered immoral. Both reasons have the same root: an attempt to shame people into doing what is not in their self-interest.

On one hand, good programmers who could be earning high salaries from employers using their talents efficiently are wasting their talents writing free code in the hopes of meager payments (because they have been convinced it is the right thing to do). Businesses that can get hold of this code for free are supposed to be shamed into voluntarily paying for it (businesses are not easily shamed).

Smart shareware developers give away the code, with enough mystery in terms of how to use it properly, to charge big bucks for support. Shareware code can be purposely enigmatic and riddled with bugs.

The intellectual property distribution turns the immoral business of shareware into its very utopian vision. Independent developers can develop the best possible modules, objects, and agents and they will get paid based on the number of people and businesses that choose to use them.

Individuals and firms have free access to safe, quality code – with no charge ever. Most importantly, a single non-redundant code base develops for the entire world. Of course, attaining the latter goal is impossible, but reaching for the stars can take us very far.

There are many programming languages and many design patterns. Protocols of object and agent interactions, or even subroutine calls, are not universal. Every entity has its own traditions based on legacy code and the way things were always done. Hot shot programmers bring their own ideas into the mix, straining interfaces and increasing the probability of error.

Companies that spend $20,000 per line of code to ensure that each line is flawless and follows the latest software engineering guidelines succeed at the expense of being late to market with a huge deficit to overcome. The real tragedy is that the

same $20,000 is being spent for the same line of code, in the same method, in the same object, in many places over time and space.

With the intellectual property distribution, software engineering, as a discipline, becomes integrated with library science. AI tools can help the search process when the number of ways an object can be classified are in the hundreds or thousands. The VSGs and Federation Library will establish conventions and conform software IP to those conventions.

Code cannot be patented that provides no added functionality, utility, speed, or space savings over patented code. Aesthetics, such as color, cannot be patented, but a method to set aesthetics in an object can be, if not already included.

When a software system is patented, all objects within the system must be patented as non-artistic content or patented applications with a product code. Typically, the high-level object will be patented. Otherwise, why bother writing the code in the first place.

Objects patented at the next level down are bonuses, possibly reused in other applications around the world, bringing in IP royalties as the VIP Treasury releases new money into the economy. Patenting some new methods in existing objects or some new arguments to existing methods can bring in profit as well. Finding bugs in patented code transfers ownership of the repaired line and newly added conditional code.

When patenting code, instantiation physically removes all unused methods and attributes. Rarely used methods in frequently used objects will have no ownership in typical instantiations.

When the time comes, it is hard to say what languages will rise to the top. Java is a clear favorite today for almost all non-speed-critical applications. C++ can be used to patent the machine code for any hardware device, using device-specific names for registers and other hardware-specific interfaces. The translator, likely patented in Java, converts the patented C++ to the actual machine code.

An unexpected, yet desired outcome of software patents is for others to compile the code and distribute the application more efficiently than the original developers. This relieves the original developers of the distribution and sales burden and increases royalties if the more efficient distribution exceeds the 5% cost from the competing product ID.

More so than any other engineering discipline, the intellectual property distribution favors at-home software developers working within the guidelines set by the VSGs and Federation Library to write the objects and agents of the future and be handsomely rewarded when firms use their code. Revenge of the shareware writers.