The Software Invention Cube: A Classification Scheme for Software Inventions*

Jan A Bergstra
Instituut voor Informatica, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
and
Paul Klint†
Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands

Received 3 July 2007, revised 7 June 2008

The patent system protects inventions. The requirement that a software invention should make ‘a technical contribution’ turns out to be untenable in practice and this raises the question, what constitutes an invention in the realm of software. The authors developed the Software Invention Cube (SWIC), a classification of software inventions and used this classification to explore the meaning of the notions ‘novelty’, ‘inventive step’ and ‘someone skilled in the art’ for software inventions. They came to the conclusion that no meaningful distinction can be made between software invention and software discovery, which is crucial in patent law. They also showed that only in very few cases copyright is an alternative for patents to protect software inventions. In their analysis they made a distinction between ‘software inventionism’ (the point of view that software inventions per se can exist and precede any patenting or any other form of protection) and the techno-political decisions that can be combined with it. The result is a framework that enables reasoning about the software inventions and their potential protection. Hence, the question of what constitutes a software invention and the desirability to protect such an invention in any way is completely decoupled.

This paper is a sequel to the previous paper1 in which a proposal was formulated for an IPR-based software engineering life cycle and it was argued that only when an IPR-based software engineering life cycle is used a rational strategy towards software patenting, software patent licensing as well as IPR defense is possible. Further, an extensive discussion was given regarding the problem of so-called trivial patents. These seem to undermine the vitality and usefulness of the software patenting system. A number of examples of patents and patent applications that may be considered trivial were given. Long term strategies were discussed to remove trivial patents from the scene. Further, a research agenda consisting of a number of promising research questions concerning software patenting was worked out in significant detail and several policy recommendations were made.

Keywords: Software inventions, software engineering lifecycle, software patents, IPR on software

The current paper studies the notion of invention in the realm of software, starting with the colloquial meaning of the word ‘invention’ and moving to legal and other aspects of this notion. The copyright is then compared with patents and a summary of the state of the art in software and how it is documented is also covered. After the analysis of software invention, the paper is concluded with recommendations of this research.

What is an Invention?

Colloquial Meaning

The American Heritage Dictionary of the English language defines the colloquial meaning of the word ‘invention’ as:

(a) The act or process of inventing
(b) A new device, method, or process developed from study and experimentation
(c) Skill in inventing, inventiveness
(d) A discovery, a finding.

Webster Online defines invention as:

(a) Discovery, finding
(b) Productive imagination
(c) Something invented as (i) a product of the imagination; (ii) a device, contrivance, or process originated after study and experimentation
(d) The act or process of inventing

WordNet defines invention as:
(a) The creation of something in the mind
(b) A creation (a new device or process) resulting from study and experimentation
(c) The act of inventing

An invention is a creation of mind, and both the process to arrive at this creation or the capability to create it, are referred to as invention. An invention is also supposed to be new, although it is unspecified for whom. An invention is a novel device, material, or technique. It is also useful to contrast invention with two other highly related words, i.e. discovery and innovation. Although, invention and discovery are synonymous in certain contexts, it is also common to use invention for a creation of the mind and discovery for a novel observation, usually of a natural phenomenon. Reference.com explains the difference between invention and innovation:

‘Following the terminology of political economist Joseph Schumpeter, an invention differs from an innovation. While an invention is merely theoretical (even though it might have been filed with the Patent Office), an innovation is an invention that has been put into practice. However, this conflicts with the theory of social anthropologists and other social sciences researchers. In social sciences, an innovation is anything new to a culture. The innovation does not need to have been adopted’.

It can be inferred from the above that there are three phases that play role in the colloquial meaning of the word ‘invention’:
(a) The capability or skill to invent.
(b) The act or process of invention.
(c) The outcome of this process.

Other phases like, ‘application of the invention’ and ‘adoption of the invention’ belong to the realm of innovation. Another way to represent this distinction is shown in Figure 1. During conceptual phase, a ‘mental concept’ first comes into existence in the mind of inventor and is then gradually refined, made more precise and finally described in detail; it is turned into a ‘described concept’. Next, the merits of described concept, if any, have to be determined. If the outcome is positive, described concept is transformed into a ‘validated concept’ that forms the core of an invention. In subsequent ‘invention’ phase, the IPR and application policy for the invention have to be determined, i.e.,
(a) Keep it secret
(b) Keep it secret and apply it
(c) Publish it
(d) Patent it

In ‘innovation’ phase, the invention is applied and used.

Legal Meaning

Merriam-Webster’s Dictionary of law defines the ‘invention’ as:
(a) A device, process, or discovery under US patent law that is new and useful, that reflects extraordinary creative ability or skill, and that makes a distinct and recognized contribution to and advancement of science.
(b) The act or process of creating such an invention.

Although, the word invention has a colloquial meaning, in the context of law it has been defined as an independent, technical notion. In European Patent Convention (Article 52), a patentable invention is characterized by three properties, i.e., novelty, inventive step, and industrial applicability.

EPC, Article 54 defines ‘novelty’ as:
- An invention shall be considered to be new if it does not form part of the state of the art.
- The state of the art shall be held to comprise everything made available to the public by means of a written or oral description, by use, or in any other way, before the date of filing of the European patent application.
EPC, Article 56 defines ‘inventive step’ as:
An invention shall be considered as involving an inventive step if, having regard to the state of the art, it is not obvious to a person skilled in the art.

EPC, Article 57 defines ‘industrial applicability’ as:
An invention shall be considered as susceptible of industrial application if it can be made or used in any kind of industry, including agriculture.

In order to apply the above definitions to the area of software, several questions arise:
- What is the ‘state of the art’ in software?
- Who is the person ‘skilled in the art’ of software?
- What is a software invention?
- How can a software invention be compared with the state of the art?

Concepts that are relevant for the subject matter of this paper like idea, invention, work and discovery have different meanings in colloquial language, in strict legal texts, and in discussions in the area of philosophy of science. In this paper, they are considered from a multi-disciplinary perspective.

Inventions as a Class of Knowledge
Invention represents a form of knowledge that is usually identified with its readable textual description. An invention is seen as a ‘described invention’. At the opposite end of the spectrum are ‘embodied inventions’ which are described inventions that have been applied in a specific, material, case. Both described and embodied inventions play a role in patenting and future use of Patents Act in the universe. Patenting is about the creation of a body of described inventions and its use, whereas, for instance, infringement deals with the matching between described inventions and embodied inventions. The state of the art contains described inventions as well as embodied inventions though the degree to which is a matter of dispute and development. By default, the term ‘invention’ can be used to denote a described invention. If an embodied invention is meant it will always be made explicit, or be clear from the context.

The counter-intuitive situation on the requirement that an invention be new was pointed out by Reinier Bakels. An invention has been new at the time of its making, later on it is still an invention though not new anymore. The same holds for mathematical theorems.

The advantage of this terminology is that a patent application can be said to contain an invention even if its novelty is contested. It is very much like mathematical theorems: scientific journals prefer new theorems, but the concept of a theorem as such in no way refers to novelty. Similarly, the discovery of radio-activity is still to be classified as a discovery, though, not a new or recent discovery.

One cannot assume that all inventions constitute patentable subject matter by definition. Thus, some inventions may escape patenting, for instance, if regulations have been adapted because the economic or legal effects of patenting certain classes of inventions are considered detrimental. Hence, being an invention is independent of either specific patents or even the existence or operation of entire patenting systems, like being a ‘work’ is independent of the existence of copyright regulations. Moreover, inventions go through a life-cycle, but never cease to exist.

It is the embodiments that may cease to exist, and in some cases all descriptions of an invention may disappear. From a theoretical point of view the questions ‘Was the invention of combustion engine done by Archimedes’, and ‘Was the invention of the combustion engine known to ancient Greeks’ are reasonable historic questions, currently both provided by a negative answer. Such questions are never seriously asked, but do not differ in principle from the reasonable question ‘Was RSA known to the KGB before R, S and A developed it?’.

Discovery versus Invention
In the natural sciences (as opposed to the legal domain) there is a conceptual tool to make distinction between discovery and invention, discoveries are embodied in nature while inventions are embodied in artefacts. Notions such as ‘reality’ and ‘observation’ are used to explain that the instruments, being artefacts, used during experiments lead to valid conclusions that are independent from the actual instruments being used.

In the world of software no plausible distinction between invention and discovery seems possible. Dictionaries fail to make a sharp distinction between these words. It is also artificial to split discovery and invention as the results of science versus those of technology. As a consequence, it seems hard to distinguish algorithmic inventions/discoveries, usually considered as abstract scientific discoveries and thus, being unpatentable, from other software inventions. This explains why proponents of software
patents have a hard time delineating the scope of patentable content. It can be expected that no reliable distinction can be made.

It is commonly considered a good thing that countries maintain groups of public knowledge workers who produce software inventions/discoveries which are published before they can ever be patented. This provides a balancing force against software patenting. Even if those knowledge workers are classified as scientists and are employed by general universities, fail to provide any criterion or deeper grounds as to why certain topics in software engineering are being investigated with the purpose of publication in mind rather than that of patenting. It is the phenomenal effectiveness of the scientific mode of operation which justifies public costs that go with employment of these software engineering scientists as well as with the facilities they need for their work. However, each theme within software engineering can be approached in the publication-oriented scientific tradition as well as in a more closed and patent-oriented commercial R&D tradition and both approaches can lead to valid and useful outcomes. Hence, it can be concluded that neither approach leads to an answer of the discovery versus invention question.

**Copyright versus Patent**

The common view on copyright versus patenting is that copyright protects the 'expression' of an idea, while a patent protects the 'idea itself'. An 'idea' is here understood to include a perspective on a functional application. An intriguing case occurs when both, orthogonal mechanisms are applicable to protect the same artefact.

The first line of defense is to use copyright to protect owner of the expression (which is an embodiment of invention) against IPR violations. The second line of defense is to patent specific ideas. Since legal procedures involving patents imply significant legal risks and associated costs, it will be preferable to use, in case of a perceived violation, the first line of defense (copyright) whenever possible.

This combined usage of copyrights/patents is only relevant for a patent holder who actually exploits the inventions described in his patents. He can only do so by producing embodiments and these are in many cases susceptible to copyright violation. A patent holder who does not own embodiments of inventions described in the patents he owns, cannot resort to copyright protection. This implies that in standard case for which the patent system has been setup, patents and copyright protection go hand in hand. Copyrighting is an indispensable tool because copyright violation is often easier to establish than patent infringement. However, patent protection is unavoidable in all cases where copyright protection falls short.

This standard case does not seem to correspond with the current practice of software protection. Rarely, holders of software patents want to enforce a monopoly on the production and delivery of the inventions described in their patents, but instead they prefer to use copyright to protect one specific embodiment. It is this strange situation where the practice of software patenting strongly departs from philosophical background of patent system that leads to optical illusion that copyrighting suffices.

Patents are intended to create temporary monopolies, but in practice copyrights are misused to create such software monopolies. Given the long duration of copyright protection, this gives undesired and lengthy protection to pioneers in the market. The key problem is excessive length of copyright protection in combination with the fact that copyright protection has not been designed for creating economic monopolies.

**The State of Art in Software**

In software engineering, the software life cycle is a frequently used manner of organizing the software development process. Figure 2 shows simplified version of the life cycle. There are many models for software development but in most models following five phases can be distinguished:

(a) Requirements engineering: Collect the requirements and expectations from the future owners and users of the system.
(b) Design: Translate the requirements in a specification that describes the global architecture and the functionality of the system.
(c) Implementation: Build the system. This amounts to transforming the design into software source code.
(d) Testing: Test that the implemented system conforms to the specification.
(e) Maintenance: Install, maintain and gradually improve the system.
It should be emphasized that the software life cycle covers design and construction of a software product as well as its use. Each phase contains a validation and verification (V&V) sub-phase in which quality of deliverables of those phases is controlled. Also the backward arrows that make this into a real ‘cycle’ is the possibility to become aware in later phases that decisions made in a previous phase have to be revised.

The state of the art in software is the explicit body of knowledge about software engineering that is documented in:

- The Software Engineering Body of Knowledge (SWEBOK)
- University curricula and text books
- Publications by professional organizations such as Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE)
- Web sites and mailing lists related to software projects
- Publicly available results of the software engineering life cycle from software projects. This includes all artefacts of life cycle, including designs, source code, and documentation
- Lectures, courses, and oral presentations about software engineering topics

Software Invention

For each phase of the software engineering life cycle, one can distinguish the following four aspects that play role in inventions:

(a) The capabilities to carry out this phase
(b) The process that is used to produce results of each phase
(c) The tools that are used to support this process
(d) The deliverable of each phase, i.e., output produced in the course of this phase

In principle, inventions are conceivable for all 5 × 4 = 20 phase/aspect combinations and for each phase/aspect combination, three properties, novelty, inventive step, and industrial applicability have to be specified. The final step in this analysis is therefore to investigate IPR implications of each phase/aspect combination. This is achieved by addressing the following five questions for each phase/aspect combination:

(a) What is state of the art and how is it described?
(b) What is expected technical content of a described invention?
(c) What is expected size of an inventive step?
(d) Who are skilled in the art?
(e) How does an infringement look like?

The properties, novelty, inventive step, and industrial applicability are contained in these questions, where industrial applicability is contained in the technical content question. Some authors make a distinction between physical utility, logical utility and application utility, but this is considered more geared towards the US patent system where usability is emphasized as a requirement rather than technical contribution.

By differentiating through life cycle phase it becomes clear how widely the answers to these questions can and probably will differ. A requirements engineer will need skills from psychology, sociology, business administration, formal specification and software engineering, while an implementer needs skills in programming languages, algorithms, software architecture, and software engineering. In other words, software engineering is not a homogeneous skill but is based on several scientific and engineering disciplines.

This differentiation, thus leads to more specific answers and may help to focus on described inventions. For instance, in the description of a new compiler technique it is unnecessary (or rather undesirable) to include usability arguments like ‘this invention can be used in a personal computer, including a hard disk, CD drive, and a network connection ...’ as one so often reads in patent descriptions. A typical example of the latter would be a compilation technique that is specifically aiming at reduction of power usage in mobile devices.
The analysis summarized in Figure 3 shows the Software Invention Cube (SWIC). It shows the five phases of the software engineering life cycle (engineering phase/technical aspect front plane), the four aspects of each phase (technical aspect/legal assessment top plane), and five IPR questions for each phase/aspect pair (engineering phase/legal assessment right plane). In total $5 \times 4 \times 5 = 100$ combinations are contained in the cube that needs analysis.

Exploring the Software Invention Cube

While exploring parts of SWIC, the intention is not exhaustive but to illustrate that the meaning of notions along the legal assessment axis widely vary depending on the location in SWIC. The concepts ‘state of the art’, ‘technical content’, ‘inventive step’, ‘skilled in the art’ and ‘infringement’ are context-dependent and cannot be defined for software as a whole, but the examples show that a precise meaning can be defined in each case.

Requirements Engineering

Requirements engineering is about collecting the requirements and expectations from the future owners and users of a system. It amounts to interview techniques, development of use cases of the envisaged system, informal or formal specification of the results of these interviews or use cases, and consistency checks on these specifications. For all four aspects inventions are conceivable.

Capabilities

Considering the following two hypothetical inventions in requirements engineering/capabilities:

(a) An interview technique and a manner to train these techniques.

(b) A social game that leads to better use cases and user involvement.

The state of the art contained in requirements engineering literature as a whole but has links to psychology, sociology, and may be business administration. The technical content is likely to be rather non-technical, e.g., a method to let prospective users write stories how they expect the system to behave. The inventive step might be the specific format of story cards or use of social techniques. A person skilled in the art will have a background in requirements engineering, and in particular knowledge elicitation and interview techniques. An infringement may, for instance, be an interview technique that uses an essential part of the inventive step.

Tools

In a hypothetical invention, wherein the requirements engineering is a ‘tool to administrate the results of interviews’, the state of art is the part of requirements engineering literature devoted to methods and techniques. The technical content amounts to specific aspects of interviews administrated and handled by the tool. The inventive step might be the specific views given on interviews, e.g., statistics that show how well the use cases have been covered by different user groups. A person skilled in the art might be either a requirements engineer or a tool builder. A typical infringement becomes a tool with same functionality as the inventive step. Here, the copyright can be used to protect the source code of the tool and its documentation. However, the essential idea embodied in the tool can only be protected by a patent.

Implementation

Implementation amounts to building the desired system by manually or (semi-) automatically transforming the design into software source code.

Tools

Considering the following hypothetical invention for implementation/tools:

- A profiling tool that indicates in a program's source code the amount of energy that will be consumed by a program when it is executed on a specific mobile device.
The state of the art consists of compiler construction, electronics, and software engineering. The technical content are techniques to map instruction sequences for a specific device back to energy consumption characteristics of the original source code and the ways to visualize this information. The inventive step is the idea to associate energy consumption with source code. A person skilled in the art is a compiler and writer specialized in energy-aware code generation or an electronics expert. A typical infringement is a tool that implements this idea. In this case, copyright can be used to protect the source code of the tool and its documentation. However, the essential idea embodied in the tool can only be protected by a patent.

**Deliverables**

Considering the following hypothetical invention in implementation/deliverables:

- A system that predicts the Dow Jones index.

The state of the art consists of software engineering and financial mathematics. The technical contents are the mathematical models used to make predictions. The inventive step is the specific mathematical model used and its implementation. A person skilled in the art is a software engineer or financial specialist. A typical infringement is a software system that implements the same mathematical model. The main deliverable of software implementation is source code and copyright can be used to protect it. However, the essential idea embodied in the software can only be protected by a patent.

**Testing**

Testing is used to determine whether or not a system conforms to its specifications.

**Process**

Considering the following invention in testing/process:

- Write test cases that describe the functional behaviour of a system before any coding is done. During implementation, use the test cases to check conformance with the intended system behaviour. This strategy is used in extreme/agile programming.7

The state of the art consists of software engineering, in particular specification and testing. The technical content is the idea to start with test cases rather than to use them later in the life cycle. The inventive step is the reordering of life cycle. A person skilled in the art is a software engineer. A typical infringement is a similar reordering of life cycle.

The actual description of this invention can be protected by copyright. Since this invention is close to a business process, it is not clear whether it can be protected by patent, but this is a matter of intense debate.8 A comparison with the case in implementation/deliverables may be helpful. There, main technical content of the invention is a mathematical model to predict the Dow Jones and its implementation. Here, technical content of invention is only marginal and regards process steps to be taken during testing. Since a business process consists of a sequence of process steps to achieve a business goal, many commonalities between a business process and a testing process can be seen.

**Deliverables**

Considering the following hypothetical invention in testing/deliverables:

- Produce a colour-coded version of the source code, where green source code means well-covered by test cases and red source code means that the code has not been executed by the test cases.

The state of the art consists of software engineering, in particular testing. The technical content is the idea to display test coverage as a colour-coded version of the source code. The inventive step is the use of this colour-coding. A person skilled in the art is a tester. The typical infringement is a testing system that uses colour-coding for presenting test coverage. The actual form of the colour-coded source may be protected by copyright. The idea itself can only be protected by a patent.

**Conclusion**

Enough evidences have been given that software inventions can be classified in a ‘software invention cube’ that distinguishes the phases of software engineering life cycle, aspects that play role in inventions, and IPR views on them. About 100 combinations generated by this cube have widely different characteristics regarding possibilities for protection.

The first conclusion that can be drawn from this analysis is that software inventions do exist. This point of view can be called as ‘software inventionism’. The SWIC can be helpful when used...
systematically for writing patent applications, for organizing prior art databases which are in fact ‘existing invention databases’, for reverse engineering ‘systems’ into constellations of embodied inventions and into families of described inventions. Much of the old heritage will have to be restructured in terms of described inventions for which patents cannot be filed anymore. Looking at existing practice this poses a significant modularization problem that can be solved (at least in principle) by means of a systematic matching with the SWIC.

Most studies related to software patents focus on only a small part of the SWIC. In particular, engineering phase = implementation and technical aspect = deliverables. The SWIC makes clear that significantly more cases have to be considered.

An intriguing self-referential question is whether the software engineering life cycle itself is a software invention. This question can also be asked for the SWIC. Both are scientific results rather than inventions, as a consequence both are not patentable. Hence, main conclusions are:

- SWIC is a new perspective on the subject of IPR on software.
- Software inventions do exist, can be called ‘software inventionism’.
- No meaningful distinction can be made between software inventions and software discoveries.
- Although copyright on software plays a prominent role in the debate on software patents, only in very few cases copyright is an alternative for patents to protect software inventions.
- The SWIC is a proposal for protection systems for software based on clear principles of knowledge organization.
- In first approximation, state of the art in software is contained in the body of knowledge as documented, for instance, in SWEBOK. However, important links exist to other areas such as psychology, sociology, business administration, economics, manufacturing, and electronics. Due to the wide applicability of software and software inventions, this may amount to links to most areas of knowledge.
- Due to the vast amount of knowledge that constitutes state of the art in software, the concept of someone ‘skilled in the art’ is useless. More specifically, it is useless unless this notion is specified in further detail. SWIC provides a possible decomposition of ‘the art’ into manageable sub areas where persons ‘skilled in the art’ can be identified.
- The desirability for protection of software inventions has technical, legal, economic, and even moral aspects. Such techno-political issues have not been explicitly addressed in the current paper.

Acknowledgment

Discussions with Reinier Bakels have drawn the attention to the fact that, contrary to intuition, inventions need not be new from a legal perspective. The comments made by Bronwyn Hall on a draft of this paper are gratefully acknowledged. Susanne van Dam helped to convert this paper to the required format.

References