The Software Invention Cube: a Classification Scheme for Software Inventions

Jan A. Bergstra
Informatics Institute, University of Amsterdam
and Faculty of Philosophy, University of Utrecht
www.science.uva.nl/~janb

and

Paul Klint
Centrum voor Wiskunde en Informatica (CWI), Software Engineering Department and
Informatics Institute, University of Amsterdam
www.cwi.nl/~paulk

Received July 3, 2007
Revised June 7, 2008

The patent system aims at protecting 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. We develop the Software Invention Cube (SWIC), a classification of software inventions and use this classification to explore the meaning of the notions 'novelty', 'inventive step' and 'someone skilled in the art' for software inventions. We come to the conclusion that no meaningful distinction can be made between a software invention and a software discovery, a distinction that is crucial in patent law. We also show that only in very few cases copyright is an alternative for patents to protect software inventions.

In our analysis we make 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. Note that we completely decouple the question of what constitutes a software invention and the desirability to protect such an invention in any way.

Disclaimer. This work was partly carried out during the ongoing project *Study of the effects of allowing patent claims for computer-implemented inventions*, a study initiated by the European Commission and carried out by MERIT (University of Maastricht, Netherlands), Centre of Intellectual Property Law CIER (University of Utrecht, Netherlands), Centrum voor Wiskunde en Informatica (Amsterdam, Netherlands), Telecommunication Engineering School at the Universidad Politécnica de Madrid (UPM), Spain and Centre for Research on Innovation and Internationalization (CESPRI) at Bocconi University, Milan, Italy.

The opinions expressed in this publication are those of the authors and do not necessarily reflect in any way opinions of the European Commission or any of the partners in the above mentioned consortium.

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

1 Background

As part of a three year European Commission (EC) study on the effects of software patents on innovation we are involved in a multi-disciplinary effort to understand the effects of software patents. These effects are studied from legal, economical, and computer science perspectives.

This paper is a sequel to our previous paper[1]. In that paper 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 was 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.

The goal of the current paper is to study the notion of *invention* in the realm of software. In Section 2 we start with the colloquial meaning of the word 'invention' and gradually move on to legal and other aspects of this notion.In Section 3 we compare copyright with patents. In Section 4 we give a summary of the state of the art in software and how it is documented. The main topic of the paper is presented in Section 5 where we analyze what a software invention is. In Section 5 we formulate the conclusions and recommendations of this research.

2 What is an invention?

2.1 Colloquial meaning

Let us first consider the colloquial meaning of the word *invention*. The American Heritage Dictionary of the English Language[2] defines invention as follows (we do not consider meanings that are not relevant for the current discussion):

1. The act or process of inventing.
2. A new device, method, or process developed from study and experimentation.
3. Skill in inventing; inventiveness.
4. A discovery; a finding.

Webster Online[3] defines invention as

1. Discovery, finding.
2. Productive imagination.
3. Something invented as (1) a product of the imagination; (2) a device, contrivance, or process originated after study and experimentation.
4. The act or process of inventing

WordNet[4] defines it as:

1. The creation of something in the mind.
2. A creation (a new device or process) resulting from study and experimentation.
3. The act of inventing

An invention is a creation of the 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: *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. We quote Reference.com to explain the difference between invention and innovation[5]:

> *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.*

We infer from the above that there are three phases that play a role in the colloquial meaning of the word 'invention':

1. The capability or skill to invent.
2. The act or process of invention.
3. The outcome of this process.

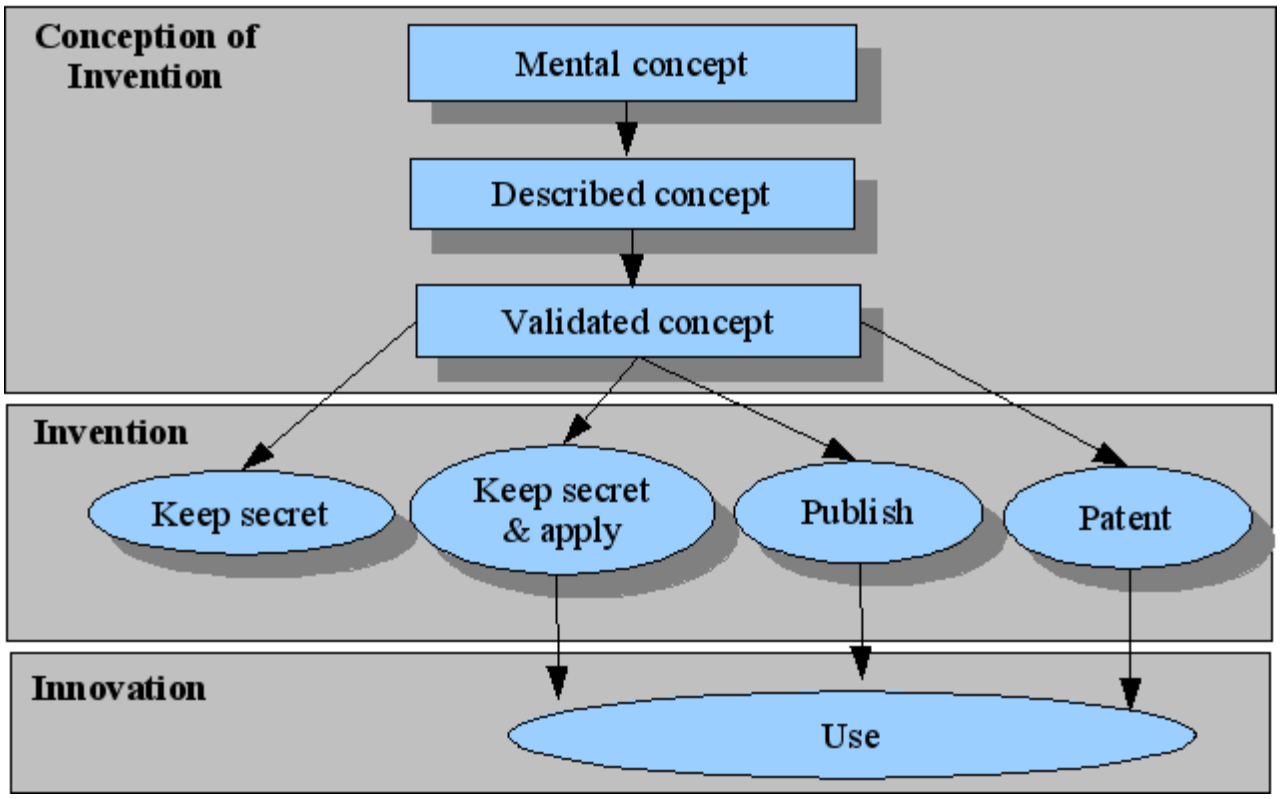

Fig. 1: From concept to innovation

Other phases like

4. Application of the invention.
5. Adoption of the invention.

belong to the realm of innovation. Another way to represent this distinction is shown in Figure 1. During the conceptual phase, a *mental concept* first comes into existence in the mind of the inventor and is then gradually refined, made more precise and finally described in detail; it is turned into a *described concept*. Next the merits, if any, of the described concept have to be determined. If the outcome is positive, the described concept is transformed into a *validated concept* that forms the core of an invention. In the subsequent *invention* phase, the IPR and application policy for the invention have to be determined:

- Keep it secret.
- Keep it secret and apply it.
- Publish it.
- Patent it.

In the *innovation* phase, the invention is applied and used.

2.2 Legal meaning

Merriam-Webster's Dictionary of Law[6] gives the following definition for invention:

1. A device, process, or discovery under U.S. 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.
2. The act or process of creating such an invention.

Although the word invention has a colloquial meaning, in the context of the law it has been defined as an independent, technical notion. The European Patent Convention (Article 52) a patentable invention is characterized by three properties[7]: novelty, inventive step, and industrial applicability. We give some brief excerpts from the European Patent Convention to clarify these notions. 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 we will have to answer several questions:

- What is the 'state of the art' in software?
- What is a 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?

We will return to these questions in the following sections. 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. We do not take the legal meaning of these concepts as leading. The contribution of our paper is that we consider them from a multi-disciplinary perspective. We are aware of (and take full responsibility for) the danger that our presentation can be criticized from strictly mono-disciplinary perspectives. Making a connection between the variations in meaning of these concepts is likely to be a serious challenge from the perspective of philosophy of law and is outside the scope of this paper.

2.3 Described 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*: described inventions that have been applied in a specific, material, case. Formidable questions arise here like: 'what inventions are embodied in an Airbus A 380?' or 'Are any inventions embodied in the human body?'. Patenting and the future use of patents act in a universe where both described and embodied inventions play a role. 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 we will use the term 'invention' to denote a described invention. If an embodied invention is meant this will always be made explicit, or be clear from the context.

Following the EPC we will explicitly not insist on the requirement that an invention be new. This counter-intuitive situation was pointed out to us by Reinier Bakels[8]. 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.

We do not 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.

Summing up: 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 the combustion engine done by Archimedes', and 'Was the invention of the combustion engine known to the 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?'[9].

2.4 Discovery versus invention

In the natural sciences (as opposed to the legal domain) there is a conceptual tool to make a 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. As we have seen in Section 2.1, 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. We expect 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, that fails 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 that justifies the public costs that go with the employment of these software engineering scientists as well as with the facilities that 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 we must conclude that neither approach leads to an answer of the discovery versus invention question.

3 Copyright versus patent

Before we further elaborate the main theme of this paper -understanding what software inventions are- a small digression on Intellectual Property Rights (IPR) is appropriate. 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. We have discussed this extensively in a previous paper[1]. An intriguing case occurs when both, orthogonal, mechanisms are applicable to protect the same artefact.

A first line of defense is to use copyright to protect the owner of the expression (which is an embodiment of the invention) against IPR violations. A 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 the 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. Only 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 the philosophical background of the patent system that leads to the 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 the excessive length of copyright protection in combination with the fact that copyright protection has not been designed for creating economic monopolies.

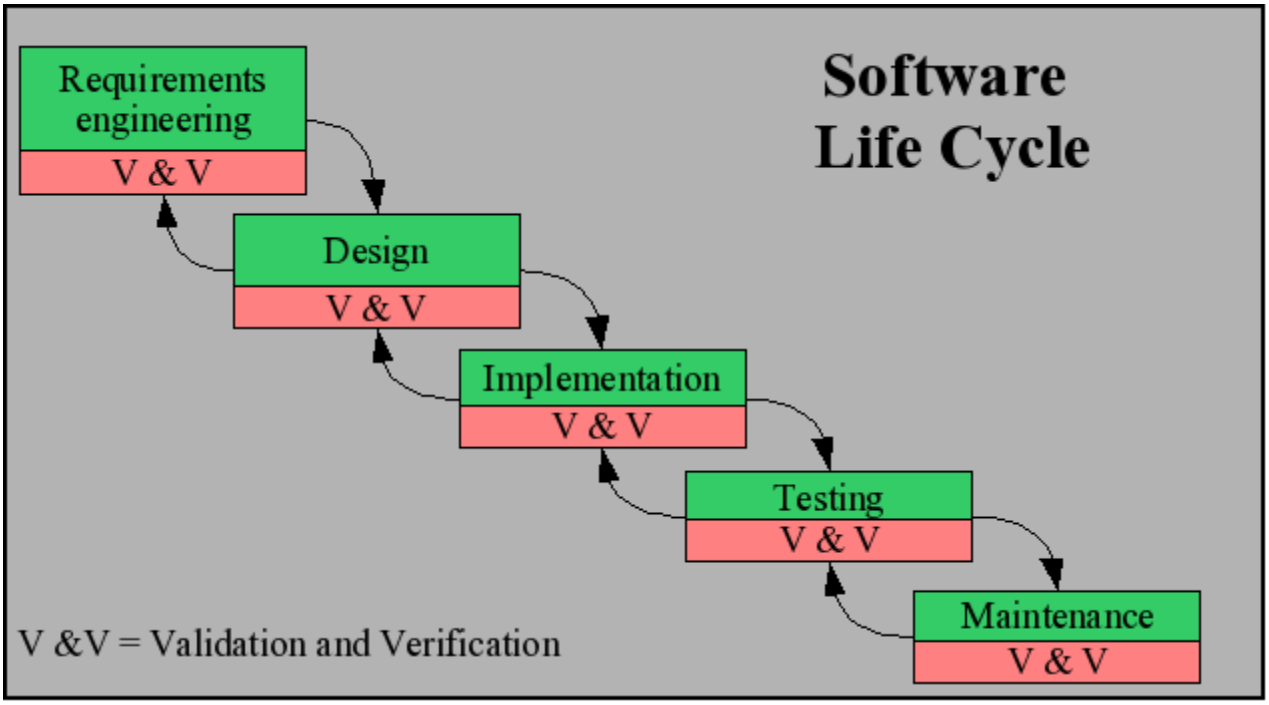

Fig. 2: The software life cycle

4 What is the state of the art in software?

In software engineering, the software life cycle is a frequently used manner of organizing the software development process. Figure 2 shows a strongly simplified version of the life cycle taken from a standard textbook[10]. There are many models for software development but in most models one can distinguish the following five phases:

1. Requirements engineering: collect the requirements and expectations from the future owners and users of the system.
2. Design: translate the requirements in a specification that describes the global architecture and the functionality of the system.
3. Implementation: build the system. This amounts to transforming the design into software source code.
4. Testing: test that the implemented system conforms to the specification.
5. 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 the quality of the deliverables of those phases is controlled. Also note the backward arrows that make this into a real 'cycle': it is possible 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)[11].
- University curricula and text books.
- Publications by professional organizations such as Association for Computing Machinery (ACM)[12] and the Institute of Electrical and Electronics Engineers (IEEE)[13].
- 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 the life cycle, including designs, source code, and documentation.
- Lectures, courses, and oral presentations about software engineering topics.

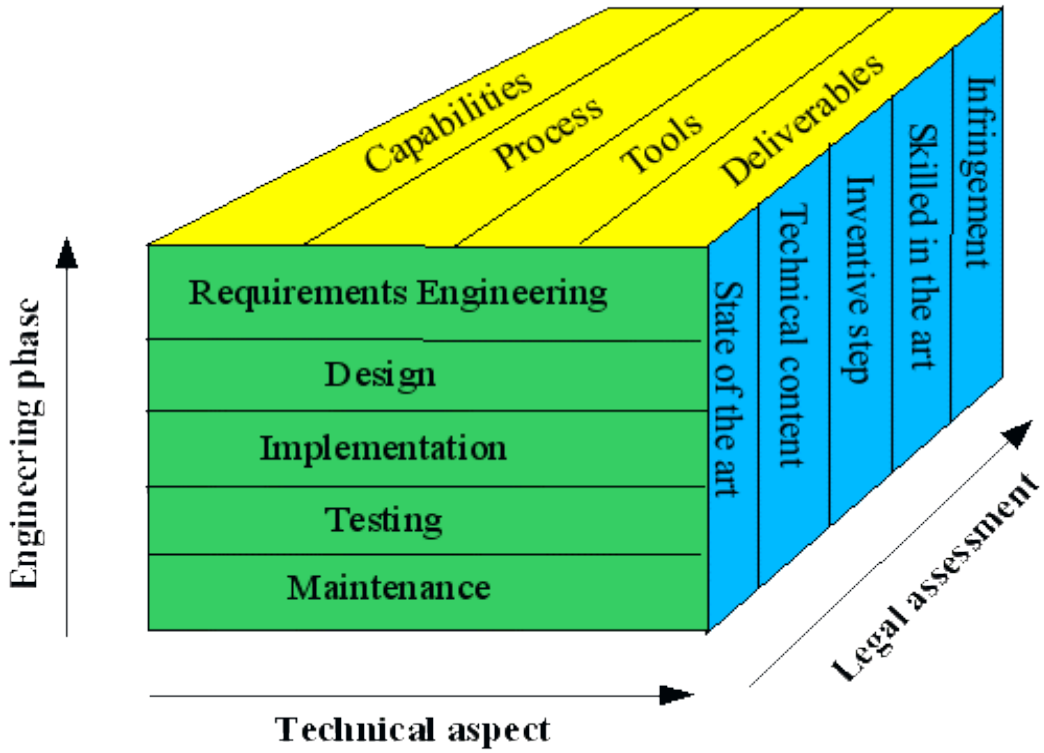

Fig. 3: The Software Invention Cube (SWIC)

5 What is a software invention?

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

1. The *capabilities* to carry out this phase.
2. The *process* that is used to produce the results of each phase.
3. The *tools* that are used to support this process.
4. The *deliverable* of each phase, i.e., the 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, the three properties novelty, inventive step, and industrial applicability have to be specified. The final step in this analysis is therefore to investigate the IPR implications of each phase/aspect combination. This is achieved by addressing the following five questions for each phase/aspect combination:

1. What is the state of the art and how is it described?
2. What is the expected technical content of a described invention?
3. What is the expected size of an inventive step?
4. Who are skilled in the art?
5. 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 our technical content question. Observe that other authors make a distinction between physical utility, logical utility and application utility[14], but we consider this as more geared towards the US patent system where usability is emphasized as a requirement rather than technical contribution.

By differentiating by life cycle phase it becomes clear how widely the answers to these questions can and probably will differ. A requirements engineer will need skills coming from psychology, sociology, business administration, formal specification and software engineering, while an implementor 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 a 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. Unless, of course, the invention is specifically related to these usability aspects. A typical example of the latter would be a compilation technique that is specifically aiming at reduction of power usage in mobile devices.

This analysis is summarized in Figure 3 that 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 the five IPR questions for each phase/aspect pair (Engineering phase/Legal assessment right plane). In total 5 * 4 * 5 = 100 combinations are contained in the cube that needs analysis. We will now discuss some representative cells of the cube.

6 Exploring the Software Invention Cube

We will now briefly explore some parts of the Software Invention Cube. Our intention is not to be exhaustive but to illustrate that the meaning of the notions along the Legal Assessment axis widely vary depending depending on the location in the SWIC. The concepts *state of the art*, *technical content*, *inventive ste*p, *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.

6.1 Requirements Engineering

As already mentioned, 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.

6.1.1 Requirements Engineering/Capabilities

Consider the following two hypothetical inventions in Requirements Engineering/ Capabilities (Here and in the following paragraphs we will suggest several hypothetical inventions which we characterize in a very global manner by omitting actual details. In a real case, such details would be essential.):

- An interview technique and a manner to train these techniques.
- A social game that leads to better use cases and user involvement.

The state of the art is contained in the 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 the story cards or the 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.

Discussion It will be clear that only very specific parts of the above inventions can be protected by copyright. For instance, the layout of story cards. The obvious protective measure is a patent to protect the idea on which the invention is based.

6.1.2. Requirements Engineering/Tools

Consider the following hypothetical invention in Requirements Engineering/Tools:

- A tool to administrate the results of interviews.

The state of the art is that part of the requirements engineering literature that is devoted to methods and techniques. The technical content amounts to the specific aspects of interviews that are administrated and handled by the tool. The inventive step might be the specific views that are given on the 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 is a tool with the same functionality as the inventive step.

Discussion 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.

6.2 Implementation

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

6.2.1 Implementation/Tools

Consider the following hypothetical invention in 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 the 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 writer specialized in energy-aware code generation or an electronics expert. A typical infringement is a tool that implements this same idea.

Discussion 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.

6.2.2 Implementation/Deliverables

Consider 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 content are the mathematical models used to make predictions. The inventive step is the specific mathematical model used and its specific 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.

Discussion 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.

6.3 Testing

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

6.3.1 Testing/Process

Consider the following invention in Testing/Process:

- Test first: write test cases that describe the functional behavior of a system before any coding is done. During implementation, use the test cases to check conformance with the intended system behavior. This strategy is used in Extreme/Agile Programming[15].

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 this reordering of the life cycle. A person skilled in the art is a software engineer. A typical infringement is a similar reordering of the life cycle.

Discussion 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 a patent, but this is matter of intense debate.[16] A comparsion with the case in 6.2.2 may be helpful. There, the main technical content of the invention is a mathematical model to predict the Dow Jones and its implementation. Here, the technical content of the invention is only marginal and only regards process steps to be taken during testing. Since a business process consists of a sequence of process steps to achieve a busines goal we see many commonalities between a business process and a testing process.

6.3.2 Testing/Deliverables

Consider the following hypothetical invention in Testing/Deliverables:

- Produce a color-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 color-coded version of the source code. The inventive step the use of this color-coding. A person skilled in the art is a tester. A typical infringement is a testing system that uses color-coding for presenting test coverage.

Discussion The actual form of the color-coded source may be protected by copyright. The idea itself can only be protected by a patent.

7. Conclusions

We have given evidence that software inventions can be classified in a Software Invention Cube that distinguishes the phases of the software engineering life cycle, aspects that play a role in inventions, and IPR views on them. The 100 combinations generated by this cube have widely different characteristics regarding possibilities for protection.

The first conclusion that we can draw from this analysis is that software inventions do exist. We call this point of view *software inventionism*. Note that we decouple this observation from any *techno-political considerations* whether software inventions should be protected or not.

Given, the Software Invention Cube, we are now in the position to answer the four questions that we raised in Section 2.2:

- What is the *state of the art* in software?
- What is a 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?

As our analysis shows, these questions are very hard to answer in general since they strongly depend on specific circumstances. However, in a structured context, like the SWIC, specific answers are possible as has been shown by our examples.

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 Software Invention Cube. Both are scientific results rather than inventions, as a consequence both are not patentable.

Our main conclusions are the following:

- To the best of our knowledge the Software Invention Cube is a new perspective on the subject of IPR on software.
- Software inventions do exist, we call this 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, we have shown that only in very few cases copyright is an alternative for patents to protect software inventions.
- Protection systems for software should be based on clear principles of knowledge organization. The Software Invention Cube is a proposal for such a principle.
- In a first approximation the state of the art in software is contained in the body of knowledge as documented, for instance, in SWEBOK[11]. 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 the 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 (that part of) the art' can be identified.

The *desirability* of the protection of software inventions has technical, legal, economic, and even moral aspects. We have explicitly not addressed such techno-political issues in the current paper.

Acknowledgments

Discussions with Reinier Bakels have drawn our 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 us to convert this paper to the required format.

References

1. J.A. Bergstra and P.Klint. About 'trivial' software patents: the IsNot case. Science of Computer Programming, **16**(3) 264-285, 2007.
2. The American Heritage Dictionary of the English Language, 2000.
3. Webster online, http://websteronline.com, 2005.
4. Princeton University. Wordnet 2.0. wordnet.princeton.edu 2003.
5. Reference.com. http://Reference.com.
6. Merriam-Webster's dictionary of law. Merriam-Webster, Inc., 1996.
7. E.A. van Nieuwenhoven Helbach and J.L.R.A. Huydecoper and C.J.J.C. van Nispen, *Industriële Eigendom*, volume Bescherming van technische innovatie, Kluwer, 2002. In Dutch.
8. R. Bakels, Private Communication, 2005, See also http://www.ivir.nl/publications/formerstaff/bakels.html.
9. R. Rivest, A. Shamir and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM,* **21**(2): 120-126,1978.
10. H. van Vliet. Software Engineering: Principles and Practice, Wiley, second edition, 2000.
11. Software engineering body of knowledge (SWEBOK). http://www.swebok.org, 2004.
12. Association for Computing Machinery (ACM). http://www.acm.org.
13. Institute of Electrical and Electronics Engineers (IEEE). http://www.ieee.org.
14. R. Plotkin. From idea to action: toward a unified theory of software and the law. *International Review of Law, Computers & Technology*, **17**(3), November 2003.
15. K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.
16. Meurer, Michael J., Business Method Patents and Patent Floods. Washington University Journal of Law and Policy, Forthcoming Available at SSRN: http://ssrn.com/abstract=311087