The software invention cube

Jan Bergstra

ASICT/IvI/FNWI/UvA
janb@science.uva.nl

HOSC: June 15, 2006

INPUT: SWIC Joint work with Paul Klint CWI/UvA

Context: EU project: "Study of the effects of computer implemented inventions". Plus some remarks that I am responsible for myself.

- Software patents: the phrase survives whether we like it or not.
- HOSC 2005: (Jan Bergstra &) PAUL KLINT: patents in the context of the software engineering life-cycle/trivial patents.
- HOSC 2006: JAN BERGSTRA (& Paul Klint): Software invention cube.
- IPR for software stands on FOUR feet:
  - law
  - economics
  - international legislation, conventions and treaties
  - software engineering
- IPR on software UNITES everyone from GPL to closed source writers: IPR on software is important !!
  We only disagree on how and when it should be done.

focus: PATENTS

Conceptual problems are massive:

- what kind of inventions/discoveries require protection
- what can be protected
- what should not be protected
- what is the boundary between copyright and patenting

In software:

- inventions are published and/or patented
- products are copyrighted (e.g. sources, executables, formal specifcations, requirements documents, textbooks)

Already this assumption is controversial. Some define 'invention' via patents: hopeless and needless circularity!

software inventionism (SI)

(software) INVENTIONISM = the COMBINATION of these viewpoints:

- 'software inventions' as a concept are prior to patents and patenting systems of all forms.
- software inventions require systematic analysis (e.g. SWIC)
- software INVENTION = software DISCOVERY.
- copyrights » IPR for software products in (almost) physical form.
- publication and patenting » IPR for software inventions.

constraints on SI:

- Neutral on the economic and ethical aspects of IPR for software.
- assumes a flexible boundary between copyrighting and patenting.
- No claims that software is special, prefix software only removes constraints for compatibility with inventions in other areas.

practical use of the phrase SI

It is implausible to write or say: I am a software inventionist.

But it works to write or say: assuming software inventionism.
This kind of language is not un-common in analytical philosophy.

techno-political software inventionism (TPSI)

= (software) INVENTIONISM plus a number of TP decisions.

- Various options for TPSI, from anti-patent to anti-copyright etc.
- TPSI's are designed and mature through political process, experience and discussion (TPSI life-cycle)
- EU is working on its own TPSI.
- Lawyers and economists: which TPSI to choose.
- Software engineering experts: details and concepts of software inventionism (e.g. the definition of prior art)
- Research on SI and on specific TPSI's to be encouraged.
- Software engineering research agenda: free from political pressures, be it from commercial interests or from open source communities!
- Freedom of thought is somewhat at risk in our circles!

Basic research on software inventions

- what inventions are we talking about?
- what is the distinction between invention and product?
- how to classify inventions?

software invention classification proposal

We propose SWIC. Details in our conference paper
- on the HOSC 2006 CD,
- or on PK's homepage www.cwi.nl/ paulk/patents)

- inventions: much larger scope than products
- most inventions cannot be copyrighted by definition

SWIC: Software Invention Cube

3 dimensional space of inventions: Cube with $4 \times 5 \times 5 = 100$ cells.

- technical aspect
  - capabilities
  - process
  - tools
  - deliverables
- engineering phase
  - requirements engineering
  - design
  - implementation
  - testing
  - maintenance
- legal assessment
  - state of the art
  - technical content
  - inventive step
  - skilled in the art
  - infringement

described invention versus embodied invention

SWIC classifies described inventions. Patent infringements always arise from embodied inventions.

described invention versus embodied invention

SWIC classifies described inventions. Patent infringements always arise from embodied inventions.

Novelty considered optional (and temporary)

Inventions need not be new! Like a mathematical theorem:
a theorem remains a theorem for centuries.
Example: automatic garbage collection (AGC) is an invention. A particular description of AGC can be projected on the 100 cells of the SWIC, many results will be empty or irrelevant. Decomposition into a number of inventions, leading to a potential IPR strategy.

described invention versus embodied invention

SWIC classifies described inventions. Patent infringements always arise from embodied inventions.

Novelty considered optional (and temporary)

Inventions need not be new! Like a mathematical theorem:
a theorem remains a theorem for centuries.
Example: automatic garbage collection (AGC) is an invention. A particular description of AGC can be projected on the 100 cells of the SWIC, many results will be empty or irrelevant. Decomposition into a number of inventions, leading to a potential IPR strategy.

Exclusion of mathematics (from invention) considered futile

Scientific publication, patenting or secrecy are alternatives. No useful selection possible of 'scientific results' (on software engineering) that cannot or should not be considered for patenting. No sound criteria. The distinction between algorithms and programs is artificial and useless (for excluding described inventions from patenting).

What will we do next? (PK & JAB)

Once we know what can be protected with IPR, the next question is: what should be protected? Impact analysis in principle: impact models as a foundation.

- Software technology life-cycle needed: impact of inventions
- Craig A. James 'The care and feeding of FOSS'
- Rogers describes technology life-cycles in general, much work has been done.
- Instantiation for software engineering required.
- Is software innovation life-cycle a useful phrase?

What will we do next? (PK & JAB)

Once we know what can be protected with IPR, the next question is: what should be protected? Impact analysis in principle: impact models as a foundation.

- Software technology life-cycle needed: impact of inventions
- Craig A. James 'The care and feeding of FOSS'
- Rogers describes technology life-cycles in general, much work has been done.
- Instantiation for software engineering required.
- Is software innovation life-cycle a useful phrase?

from invention to innovation

A TPSI regulates the process from invention to innovation. TPSI's can only be compared or assessed if that process is known.

some radical thoughts

- People from law and economy may FAIL to see that software is a revolutionary business:
  - slow communcation $\longrightarrow$ internet
  - documents lost $\longrightarrow$ search engines
  - library overflow $\longrightarrow$ digital library
  - software expensive $\longrightarrow$ FOSS
- Software IPR cannot be based on flawed and ill-understood concepts.
- It will NOT be left to legal and economic specialists.
- Extreme investments will be made if necessary to develop a PROPER IPR system for software.
- People in SE are not lazy.
- If patenting must be reorganized for software that will HAPPEN!
- Software is stronger than both TRIPS and EU. It cannot be regulated by irrational means.