Robert A. Gehring

Keine Patentlösung für Software

Im vorliegenden Aufsatz[1] soll ein wichtiger Teilaspekt der Interaktion von Patentrecht und Technik näher beleuchtet werden: das Problem, bei umfangreicher Patentierbarkeit von Software unabsichtlich bestehende Patentrechte zu verletzen. Es wird aus der Sicht der Softwareentwicklung diskutiert, wie es geschehen kann, dass Patentrechte unabsichtlich verletzt werden, welche Konsequenzen das für den Entwickler bzw. die Entwicklung von Software im Allgemeinen, haben kann und wie man den bestehenden Risiken in der Praxis begegnen kann. Ein besonderes Augenmerk gilt dabei den Konsequenzen für die Entwickler von Freier und Open Source Software.

Softwareentwicklung als Konfigurierung einer Universalmaschine

Ausgangspunkt der Betrachtungen ist die Entwicklung von Standardsoftware, die sich abstrakt als die Bewältigung folgender Aufgabe beschreiben läßt:

> *„Configure machine M to produce effects R in domain D."*[2]

Software ist dazu da Probleme zu lösen. Diese Probleme bestehen in einer mehr oder minder genau bestimmten Problemdomäne D. Die Software bewirkt in einer Universalmaschine M (z.B. Standard-PC) eine wohldefinierte Folge von Systemzuständen, die als Programmablauf bezeichnet wird. Dabei auftretende Effekte R, d.h. konkrete Verhaltensweisen der Maschine M in Interaktion mit ihrer Umwelt, sollten deterministisch zur Lösung der ursprünglich gestellten Aufgabe führen; andernfalls ist die Software fehlerhaft.

In der Lösungsidee, d.h. der Spezifikation, werden die notwendigen Effekte noch unabhängig von einer konkreten Formulierung entworfen. Anschließend wird der Lösungsweg in einer Programmiersprache so formuliert, dass der Compiler daraus die dem Computer verständlichen Anweisungen generieren kann (Abb. 1).

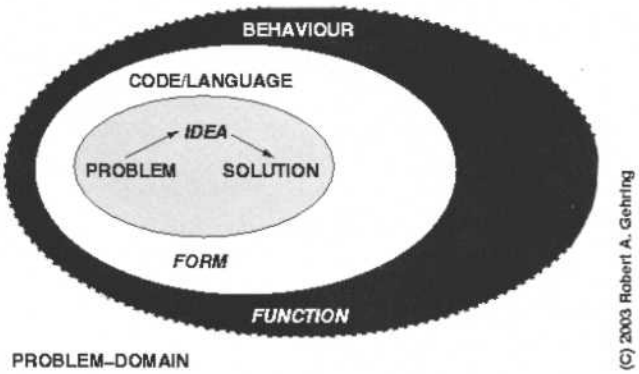

Abbildung 1: Schritte der Softwareentwicklung

Jede Informatikerin und jeder Informatiker wird mit einer solchen Vorstellung von Softwareentwicklung vertraut sein. Von rein fachlicher Warte betrachtet haben Software-Patente hier keinen Platz – ein ingenieurtechnisches Paradigma mit rechtlichen Problemen, wie wir sehen werden.

Patentschutz für Software

Für Software stehen prinzipiell zwei Schutzinstrumente[3] des Immaterialgüterrechts zur Verfügung: Im Urheberrecht wird die Form von Software geschützt, d.h. der je konkrete Ausdruck der Lösungsformulierung; das Patentrecht schützt hingegen die funktionalen Aspekte, nicht jedoch die zugrundeliegenden Ideen als solche.

Insofern Patentschutz für ein bestimmtes Verhalten eines Computers gewährt wird, ist üblicherweise die Rede von Software-Patenten.

In den letzten drei Jahrzehnten wurde, ausgehend von Gerichtsentscheiden im Vereinigten Königreich und in den USA, in zunehmendem Maße Gegenstände für patentierbar erachtet, die alltägliche Probleme der Softwareentwicklung betreffen: Zahlenkonversion, Sortierverfahren, Datenkomprimierung, Speicherverwaltung u.v.m.[4]

Der Regelkreis aus Patentanmeldern, Patentämtern und Patentgerichten generiert immer neue Patente mit sinkender durchschnittlicher Qualität[5], ein »Patentdickicht« (Shapiro 2000), in dem man sich schnell verirren kann.

Softwareentwickler stehen also grundsätzlich immer vor dem Problem, dass eine von ihnen entworfene Lösung Patente verletzen könnte.

Software-Patente als Problem der Softwareentwickung

Da es sich beim Patentschutz um ein gesetzlich geschütztes Exklusivrecht handelt, kann der Patenthalter jedem anderen die wirtschaftliche Verwertung des geschützten Gegenstandes verbieten.[6] Bereits in die Programmentwicklung geflossene Investitionen wären damit verloren, gibt es doch keine praktische Möglichkeit einen Patentinhaber zur Lizenzierung seiner geschützten Ideen zu zwingen. Selbst eine in Aussicht gestellte Lizenzierung kann ökonomisch sinnlos sein, wenn die Lizenzgebühren den zu erwartenden Gewinn übersteigen. Jedenfalls

steigt der Druck auf denjenigen, der auf die Lizenz angewiesen ist, auch zu Bedingungen zu lizenzieren, die normalerweise inakzeptabel wären, denn ohne die Lizenz kann das Produkt nicht vermarktet werden – ein „hold-up"-Problem entsteht (Shapiro 2000).

Um eine solche „hold-up"-Situation nach Fertigstellung der Software zu vermeiden, muss im Prinzip der gesamte Entwicklungsprozess einer Vermeidungsstrategie unterworfen werden. Das unvermutete Auftauchen eines bestehenden Patentschutzes, der zu einer „hold-up"-Situation führen könnte, wäre dann im Grunde genommen als ein fachlicher Fehler bei der Bestimmung der Requirements zu interpretieren.

Ausgehend von der entworfenen Funktionalität der Lösung muss ermittelt werden, ob Patentschutz für die Gesamtlösung oder für Teillösungen besteht. Gelangt man zu einer solchen Erkenntnis, müssen ggf. Lizenzverhandlungen aufgenommen werden oder die Softwareentwicklung muss abgebrochen werden, bevor unnötige Kosten entstanden sind.

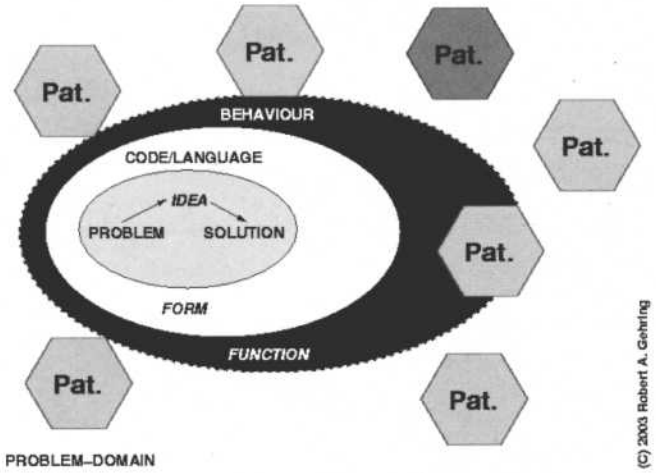

Abbildung 2: Das Patent-Problem

In Abbildung 2 sind zwei Typen von Patenten zu sehen: Die heller gezeichneten stellen jene Patente dar, die in die Problemdomäne fallen und bereits recherchierbar sind. Der zweite Typ von Patenten, in der Grafik dunkler dargestellt, wird im folgenden Abschnitt diskutiert.

Grenzen der Patentrecherche

Der zweite Typ von Patenten, in der Abbildung dunkler gezeichnet, ist noch nicht recherchierbar, da es im Laufe des Patentanmeldungs- und -erteilungsprozesses eine Phase gibt, in der Patentanmeldungen geheimgehalten werden. Sollte ein Patent später erteilt werden, tritt erneut das oben bereits geschilderte Problem mit den Exklusivrechten des Patentinhabers auf: Bereits in die Softwareentwicklung geflossene Aufwendungen werden unter Umständen wertlos. Im Falle der nicht recherchierbaren Patente ist das aber ein unvermeidliches Risiko. Wie hoch dieses Risiko ausfällt, hängt im Wesentlichen von der Granularität der erteilten Patente ab.

Granularität ist kein Terminus aus der Sprache der Patentjuristen. Mit Granularität soll hier die Größe des funktionalen Anteils an einem Softwaresystem gemeint sein, der durch ein Software-Patent abgedeckt wird. Je feiner die Granularität, desto kleiner der funktionale Anteil, und umgekehrt. Im Extremfall, bei sehr feiner Granularität, können ein paar Zeilen Quellcode (LOC) in einem zig-Millionen-LOC-Projekt von der Wirkung eines Patents erfasst sein.[7]

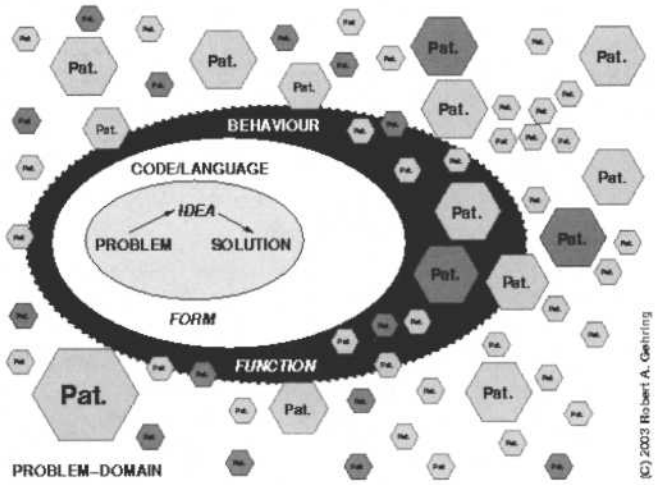

Abbildung 3: Die Patent-Gefahr

Hat man nur ein einzelnes Patent im Blick, mag das unproblematisch, ja vorteilhaft erscheinen. Hat man es mit sehr vielen Patenten feiner Granularität zu tun, stellt sich die Situation dar wie in Abbildung 3 gezeigt: Würde man auf die patentgeschützte Funktionalität verzichten (müssen), würde ein System ähnlich einem „Schweizer Käse" mit vielen „funktionalen Löchern" entstehen.

Wie in Abbildung 3 gezeigt, geht mit feinerer Granularität eine zunehmende Anzahl von Patenten einher. Das Risiko, dass eines oder mehrere Patente den Lösungsbereich innerhalb einer bestimmten Problemdomäne tangieren, steigt analog.

Beim zu verzeichnenden exponentiellen Wachstum der Patentanmeldungen kann heutzutage nicht mehr davon ausgegangen werden, dass nennenswerte Bereiche der Softwareentwicklung nicht betroffen wären. Faktisch ist man inzwischen dort angelangt, dass „alles von Menschenhand Geschaffene unter der Sonne"[8] patentierbar ist.

Ausflüchte und Auswege

Eine grundsätzliche Lösung der genannten Probleme würde wohl nur eine tiefgreifende Reform des Patentwesens leisten können. Oder besser noch, es wäre ein eigenes Schutzrecht für Software zu schaffen, das die Spezifika ihrer Entwicklung und ihres Einsatzes angemessen berücksichtigt. Eine solche Lösung ist jedoch nicht in Sicht. Softwareentwickler, die hier und heute mit dem Problem leben müssen, sind auf pragmatische Auswege und Ausflüchte angewiesen, wie im Folgenden diskutiert.

Ignoranz

Ignoranz ist hier wortwörtlich zu verstehen: Der Softwareentwickler kümmert sich schlicht und einfach nicht um bestehende oder zukünftige Patente. Stattdessen entwickelt er/sie die Soft-

ware so wie gelernt: Ausgehend von der Problemanalyse wird eine Lösungsidee entwickelt und anschließend so implementiert wie es naheliegend ist – ohne Rücksicht auf Patente.

Ob ein solches Vorgehen sinnvoll ist, hängt von der individuellen Risikobewertung ab. Ist das Risiko, vom Patentinhaber belangt zu werden, gering, kann Ignoranz der Weg zum Erfolg sein. In vielen Fällen wird das Risiko in der Tat sehr gering sein, wie sich zeigen lässt.

Wenn A für B Software entwickelt, für die C ein einschlägiges Patent hält, und beide, A und B, sich rational verhalten, werden sie stillschweigend auf die Einholung der Lizenz verzichten und zum gegenseitigen Vorteil das Geschäft tätigen. Wird das Programm im Intranet eingesetzt, wird C von der unlizenzierten Nutzung ihres „geistigen Eigentums" in der Regel gar nichts mitbekommen können. Und wo kein Kläger ...

Ignoranz funktioniert, wenn die „Kosten" unbemerkt auf Dritte abgewälzt werden können, was in der Praxis häufig der Fall sein dürfte. Solange weder A noch B dazu übergehen, die Software an Dritte weiterzugeben, fällt die Risikobewertung eindeutig zugunsten der Ignoranz aus.

Cross-Lizenzen und Patent-Pools

Cross-Lizenzen sind gegenseitige Abkommen zweier Unternehmen, in denen die Bedingungen für die Nutzung der Patente – ausgewählter oder aller – aus dem Patentportfolio des jeweils anderen Unternehmens ausgehandelt sind. Charakteristisch ist, dass die Vertragsbedingungen so lange beliebig ausgestaltet werden können, wie keine Wettbewerbsbehinderung entsteht.

Patent-Pools stellen eine Erweiterung dieses Modells dar, bei der mehrere Unternehmen Schlüsselpatente halten, die zur Anwendung einer bestimmten Technologie unverzichtbar sind. In einem solchen Falle ist es für die Unternehmen kostengünstiger, statt wechselseitige, individuelle Verhandlungen mit allen Unternehmen führen zu müssen einen Patent-Pool einzurichten.

Die Crux beider Ansätze ist darin zu sehen, dass Unternehmen ohne eigene Patente in der Regel ausgeschlossen sind. Ohne Patentschutz steht zwar die von ihnen entwickelte Technologie den Patentbesitzern möglicherweise lizenzfrei zur Verfügung, umgekehrt ist das jedoch nicht der Fall. Vielen Unternehmen bleibt deswegen nichts anderes übrig als sich ihrerseits mit Patenten zu 'bewaffnen', wollen sie nicht in absehbarer Zeit aus dem Markt gedrängt werden.

Davon hat allerdings die Open Source/Free Software-Community nichts, schließt doch ihr offenes Entwicklungsmodell die Anmeldung von Patenten weitestgehend aus.

Liberal Licensing

Im Rahmen des Eclipse-Projekts zur Erstellung einer Open Source-Entwicklungsumgebung hat IBM einen neuen Lizenztyp eingeführt, der hier als „liberal licensing" bezeichnet werden soll und der einige Ähnlichkeit mit dem Ansatz für 'Open Software Patents' aufweist (s.u.).

Der Grundgedanke der Eclipse-Lizenz[9] besteht darin, dass alle Projektteilnehmer sich wechselseitig verpflichten, bestehende Patentansprüche nicht gegen die entwickelte Software durchzusetzen:

> *„Subject to the terms of this Agreement, each Contributor hereby grants Recipient a nonexclusive, worldwide, royaltyfree patent license [...]. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder."*

Im Unterschied zur GNU Public License (GPL) und anderen Free Software/Open Source-Lizenzen, die urheberrechtlich wirken, erstreckt sich diese Freistellungsklausel nur auf das Programm. In der Folge ließen sich keine Programmteile zu anderen Zwecken lizenzfrei wiederverwenden.

Das Free Software/Open Source-Software-Patente-Dilemma wird wiederum nur symptomatisch und nicht grundsätzlich überwunden. Inwieweit dieses Beispiel Schule machen wird, bleibt abzuwarten.

Dual Licensing

Dual Licensing ist eher aus dem Bereich der urheberrechtlichen Lizenzierung bekannt denn von Patentlizenzen. Beim Dual Licensing macht der Rechteinhaber dahingehend von seinen Exklusivrechten Gebrauch, dass er Nutzungsrechte in Abhängigkeit von Zweck und Gestalt der Nutzung kostenpflichtig oder kostenfrei gestattet.

Das wohl prominenteste Beispiel für Dual Licensing im Bereich der Software-Patente ist die »RTLinux Open Patent License, version 2.0« der Firma FSMLabs. Die genannte Lizenz bezieht sich auf das U.S.-Patent 5,995,745, das dem FSMLabs-Entwickler Victor Yodaiken am 30. November 1999 zugesprochen wurde.

Gemäß der »RTLinux Open Patent License, version 2.0« darf die patentierte Erfindung unter bestimmten Bedingungen, d.h. insbesondere im Zusammenhang mit der Entwicklung freier Software, ohne Zahlung von Lizenzkosten genutzt werden:

> *„[U]se of the Patented Process is permitted, without fee or royalty, when used: A. By software licensed under the GPL; or B. By software that executes within an Open RTLinux Execution Environment – whether that software is licensed under the GPL or not."*

Für alle anderen Nutzungsformen entfällt die „automatische" Freistellung von Lizenzzahlungen.

Aus Sicht der Open Source/Free Software-Entwickler ist eine solche Lizenz natürlich ein Schritt in die richtige Richtung. Da es aber grundsätzlich ins Ermessen des Lizenzgebers gestellt ist, die Lizenzbedingungen für zukünftige Nutzungen zu ändern, ist wirkliche Rechtssicherheit nicht gegeben.

Der Berliner Ansatz zu "Open Software Patents" (Gehring 2000)

Es gibt Ähnlichkeiten zwischen dem Berliner Ansatz zu "Open Software Patents" und dem „Liberal Licensing". Allerdings ist

ersterer deutlich breiter konzipiert worden. In Anlehnung an die „Vererbung" der Lizenzrechte und -pflichten, wie sie aus der GPL bekannt ist, werden Lizenznehmer zur lizenzkostenfreien Zurverfügungstellung von Nutzungsrechten aus Patenten verpflichtet, wenn sie selbst Urheberrechte und/oder Patentrechte an der betroffenen Software in Anspruch nehmen wollen. Man kann sich das Ergebnis als universellen Patent- und Urheberrechtspool vorstellen, innerhalb dessen ein Waffenstillstandsabkommen gilt und je mehr Teilnehmer dieser Pool hat, desto weiter reicht der „allgemeine Landfrieden".

Auf lange Sicht könnte es für die beteiligten Entwickler und Unternehmen sinnvoll sein, auf diese Weise die Kosten für die Beilegung von Streitfällen zu verringern.

Ein letzter Ansatz bleibt vorzustellen, der eine auf die Entwicklung quelloffener Software zugeschnittene Modifikation des Patentrechts vorschlägt.

Das Quelltextprivileg (Lutterbeck, Horns & Gehring 2000)

In einem Gutachten für das Bundeswirtschaftsministerium untersuchten Lutterbeck, Horns & Gehring (2000) die Auswirkungen des Patentschutzes für „Computer-implementierte Erfindungen" auf die Entwicklung von quelloffener Software im Allgemeinen und mögliche Implikationen für die Sicherheit der Informationstechnologie im besonderen.

Die Autoren stellen fest, dass kleine und mittlere Unternehmen ebenso wie die Entwickler quelloffener Software durch das bestehende Patentsystem benachteiligt werden. Hinzu kommt, dass sicherheitsrelevante Softwaretechnologien sich nicht so weit verbreiten wie es im öffentlichen Interesse wünschenswert wäre. In der Schlussfolgerung wird die Einführung eines Quelltextprivilegs empfohlen:

> *„Der Umgang mit dem Quelltext von Computerprogrammen muss patentrechtlich privilegiert werden. Das Herstellen, Anbieten, in Verkehr bringen, Besitzen oder Einführen des Quelltextes eines Computerprogrammes in seiner jeweiligen Ausdrucksform muss vom Patentschutz ausgenommen werden. (Quelltextprivileg)"*[10]

Das Quelltextprivileg, wie vorgeschlagen, würde den Softwareentwickler zu einem Teil von Klagedrohungen wegen mittelbarer und/oder unmittelbarer Patentverletzung entlasten. Aber auch bei diesem Vorschlag handelt es sich um eine Symptombehandlung, die allenfalls einige wenige Schwächen des Patentsystems lindern hilft. Gesund wird der Patient dadurch noch lange nicht.

Fazit

Bis in die späten sechziger Jahre konnte nach allgemeiner Auffassung, die von den seinerzeit führenden Lehrbüchern geteilt wurde,[11] Software nicht patentiert werden.

Software wurde seinerzeit in der Hauptsache gebündelt mit Hardware geliefert, im Quelltext, um den Kunden die Gelegenheit zu geben, den Code ihren Bedürfnissen entsprechend anzupassen. In der Zwischenzeit haben sich die Vermarktungsstrategien für Software verändert, aber die Prinzipien der Softwareentwicklung sind gleich geblieben. Das Prinzip der Quelloffenheit von Software, das seinerzeit der Normalfall war, ist in dieser Zeit dabei Märkte neu zu besetzen. Die Entwickler stoßen auf die gleichen Schwierigkeiten, die seinerzeit Wissenschaft und Praxis zu den Warnungen vor Software-Patenten veranlasst hatten.

Es bleibt eine ernüchternde Einsicht: Eine Patentlösung für die durch 'Software-Patente' verursachten Schwierigkeiten bei der Softwareentwicklung ist derzeit nicht in Sicht.

1 *Der Aufsatz basiert auf einem Vortrag, den der Autor am 9. Juli 2003 im Wissenschaftszentrum Berlin gehalten hat.*
2 *Kovitz, 1999, S. 25.*
3 *Auf die Möglichkeit des Gebrauchsmusterschutzes (sog. kleines Patent) wird nicht eingegangen. Auch gewährt das hier nicht behandelte Markenrecht Schutz für Namen und Kennzeichnungen.*
4 *Mehr zu den historischen Hintergründen im Aufsatz von Kretschmer und Gehring im vorliegenden Heft.*
5 *Wenn auf der einen Seite die Menge der angemeldeten und erteilten Patente je Mitarbeiter des Patentamtes stetig steigt, steht zur Prüfung der Patentanmeldung zwangsläufig weniger Zeit zur Verfügung. Parallel dazu wächst jedoch die Menge des publizierten Wissens (in Patentschriften und anderswo), d.h. der Umfang der ‚prior art'–Prüfung müsste eigentlich zunehmen. Setzt man das ins Verhältnis und geht man davon aus, dass die Zeit der Prüfung einer Patentanmeldung irgendwas mit der Qualität des Patents zu tun hat, muss man schließen, dass die Durchschnittsqualität sinkt.*
6 *In 'common law'-Systemen spricht man von property rights im Gegensatz zu liability rights. Erstere gewähren ein Verbotsrecht, letztere nur einen Entschädigungsanspruch.*
7 *Adaptiert wird ein Begriff der Granularität von Hohmann, 2003, S. 19, der mit Granularität »the level of work performed by a component« bezeichnet.*
8 *So der U.S. Supreme Court in Diamond v. Chakrabarty, 447 U.S.303 (1980).*
9 *Zur Lizenz: http://www.eclipse.org/legal/cpl-v10.html.*
10 *Lutterbeck, Horns & Gehring, 2000, S. 9.*
11 *Tapper, 1983, S. 2, Fn 4; Bender, 1988, Abschnitt 3.02(4), S. 39.*

Literatur

D. Bender (1998): Computer Law. Software Protection, Matthew Bender, New York.

R.A. Gehring (2000): Der Berliner Ansatz zu Open Software Patents. Beitrag zur Konferenz über wirtschaftspolitische Aspekte der Patentierung von Software, Bundeswirtschaftsministerium, Berlin, 18. Mai 2000, *http://ig.cs.tuberlin.de/ap/rg/2000-05/Gehring-OpenSoftwarePatents052000.pdf* [24 Aug 2003].

L. Homann (2003): Beyond Software Architecture. Creating and Sustaining Winning Solutions, Addison–Wesley, Boston u.a..

B.L. Kovitz (1999): Practical Software Requirements, Manning, Greenwich, CT.

B. Lutterbeck, A.H. Horns, R.A. Gehring (2000): Sicherheit in der Informationstechnologie und Patentschutz für Softwareprodukte – ein Widerspruch? Gutachten für den Bundeswirtschaftsminister, Berlin, *http://ig.cs.tu-berlin.de/forschung/IPR/LutterbeckHornsGehring-KurzgutachtenSoftwarePatente-122000.pdf* [28 Aug 2001].

C. Shapiro (2000): Navigating the patent thicket: Cross licenses, patent pools, and standard–setting. Competition Policy Center. Paper CPC00 011.

C. Tapper (1983): Computer law, 3. Aufl., Longman, London and New York.