Freie Universität Berlin – Institut für Informatik

Proseminar: Informationsethik

Dozent: Dr. Sandro Gaycken

Sommer Semester 2011

Ethische Grundlagen von Softwarekontrolle

Christopher Patton

cjpatton@ucdavis.edu

Immatrikulationsnummer: 4430339

10. August 2011

Inhaltsverzeichnis

1 Einleitung

In seiner Rede „Open Code and Open Societies“ hat Lawrence Lessig das folgende Zitat von Thomas Jefferson vorgelesen:

„If nature has made any one thing less susceptible then all others of exclusive property, it is the action of the thinking power called an idea, which an individual may exclusively possess as long as he keeps it to himself … That ideas should freely spread from one to another over the globe, for the moral and mutual instruction of man, and improvement of his condition, seems to have been peculiarly and benevolently designed by nature, when she made them, like fire, expansible over all space without lessening their density at any point, and like the air in which we breathe, move, and have our physical being, incapable of confinement, or exclusive appropriation. Inventions then cannot, in nature, be a subject of property.“ (Lessig 2000, 7).

Jefferson spricht hier über die wesentliche Natur von Ideen. Ideen funktionieren seiner Meinung nach gleichermaßen wie Luft. Es ist ein Material, das man nicht kontrollieren kann. Wenn eine Idee geäußert wird, ist sie frei und von allen erhältlich. Heutzutage haben wir uns entschlossen, dass wir Ideen tatsächlich kontrollieren können. Infolgedessen gibt es eine ganze Menge an Methoden, andere Leute von diesen Ideen auszuschließen. In diesem Zusammenhang ist Software ein sehr interessantes Problem. Software spielt heutzutage eine enorme Rolle nicht nur in der Wissenschaft, sondern auch in der Gesellschaft sowie Ökonomie. Software in der Form von Quellcode oder Maschinenbefehlen wird als geistiges Eigentum betrachtet und wird mit einem Copyright versehen. Software kann aber auch als eine Erfindung angehen werden und man kann Software patentieren lassen, obwohl nicht direkt. Dieses zweideutige Verständnis von Software führt zu mehreren ethischen Problemen. Zuerst wird das grundlegende Problem formuliert. Die ethischen Grundlagen werden danach erklärt, die wir uns im Zusammenhang mit diesem Problem überlegen werden. Dann werden die Funktionsweisen des Copyrights und des Patents erklärt. Obwohl diese Schutzmittel in den meisten Ländern im Prinzip gleichermaßen funktionieren, haben sie zum Teil von Regierung abhängige Verschiedenheiten. (Urheberrecht in Deutschland ist beispielsweise juristisch unterschiedlich mit Copyright in den USA.) Um diese technischen Aspekten auszuschließen, werden nur Beispiele aus den USA in dieser

Arbeit verwendet. Es wird danach versucht, die Kontrolle von Software ethsich zu begründen; zum Schluss wird der soziale Umgang dieses Themas besprochen. Die zwei Arten von Softwareschutz liegen in Konflikt miteinander; vielleicht ist es der Fall, dass die ethischen Probleme gelöst werden können, wenn wir eine dieser Arten Softwarekontrolle nicht mehr ins Spiel lassen.

2 Problemerklärung

Schwierig bei der ethischen Analyse von Softwareschutz ist vor allem die große Mannigfaltigkeit von Software, die es gibt. Heutzutage entwickelt man Software, der Probleme in fast jedem Bereich behandelt. Wir müssen irgendwie Software kategorisieren, um mit diesem Problem überhaupt umzugehen. Vernünftig wäre die Motivation oder Absicht hinter einem Softwareprojekt zu betrachten, wobei man zwischen abstrakten und konkreten Motivationen unterscheiden würde. Ein Unternehmen könnte zum Beispiel eine Website entwickeln lassen, die Informationen über die Firma darstellt und somit eine Informationshotline ersetzt. Der konkrete Grund dafür wäre Zugriff auf Informationen für Kunden zu beschleunigen. Die Website ersetzt aber auch einen Arbeitsplatz und spart Geld für die Firma, das als eine abstrakte Absicht des Unternehmens betrachtet werden kann. Man könnte alternativ Code auf reinen wissenschaftlichen Grund entwickeln. Software hat in der Bioinformatik beispielsweise außer der Unterstützung von Wissenschaftlern keine Bedeutung. In einem Softwareprojekt hat der Code ein konkretes Ziel; seine abstrakte Absicht wäre aber das allgemeine wissenschaftliche Kenntnis zu erweitern. Softwareschutz ist ein anderer bedeutungsvoller Bereich der Informatik, der auch wissenschaftliche Aspekten erfordert, nämlich Verschlüsselungsalgorithmen. Solche Algorithmen haben aber nur Bedeutung in der Informatik selbst und können in der Regel nicht als die Motivation der spezifischen Anwendung angesehen werden, sondern eher als das abstrakte Ziel, Datenschutz im allgemeinen zu verbessern. Man könnte letztens vielleicht behaupten, dass es manchmal eine künstlerische Absicht hinter Software gäbe. Ein Videospiel ist ein gutes Beispiel für ein Programm, das sehr viele künstlerische Elementen einfügt und das gleichzeitig erscheint eine zweideutige Motivation zu haben. Man könnte vielleicht die Absicht der Entwicklung eines Videospieles auf reinen ökonomischen Grund reduzieren; dies ist allerdings möglich mit fast allen Formen von Software. Man kann

sich aber vorstellen, wie ein Videospiel auf künstlerischen Grund entwickelt werden kann.

Wir haben somit vier Hauptmotivationen hinter der Entwicklung von Software: Geld zu sparen, Wissenschaft und Wissenschaftler zu unterstützen, Datenschutz, und Kunst. Diese Gründe motiviert die Idee von Softwareschutz. Wenn man zum Beispiel ein Datenschutzprogramm entwickelt, lohnt es sich auch, die Funktionsweise des Programms zu schützen, denn das Programm wäre sonst effektiv nutzlos. In der Entscheidung, welchen Schutzmittel zu verwenden, Patent oder Copyright, überlegt man sich hauptsächlich aber nicht nur den Grund für Schutz, sondern auch die ethische Wirkung seiner Entscheidung.

3 Ethische Grundlagen

Wir überlegen uns diese Ideen im Zusammenhang mit den zwei gegenüber liegenden Systemen der Ethik, die im Kurs besprochen wurden, nämlich Utilitarismus und Deontologie. Der wesentliche Unterschied zwischen den beiden entspricht dem Konflikt zwischen dem Guten gegenüber dem Recht. In einem System spielt eins davon die Hauptrolle und das andere nicht. Es gibt eine ganze Menge an Mischformen von Utilitarismus und Deontologie, die versuchen, diesen Konflikt vernünftig zu lösen; im Zusammenhang mit dieser Arbeit will ich aber nur in Frage stellen, ob die verfügbaren Schutzmittel für Software, nämlich Patent und Copyright, überhaupt ethisch zu begründen sind. Dies lässt die Aufgabe übrig, diese verschiedenen Schutzmittel entweder am Utilitarismus oder an der Deontologie anzupassen.

3.1 Utilitarismus

Die Grundlage von Utilitarismus ist das von Jon Stewart Mill genannte Greatest Happiness Principle oder das Prinzip der größten Glück. Im Grunde genommen ist das Ziel von Utilitarismus gleichzeitig die allgemeine Glück zu maximieren und die Behinderung der Glück zu minimieren:

“According to the Greatest Happiness Principle, as above explained, the ultimate end, with reference to and for the sake of which all other things are desirable (whether we are considering our own good or that of other people), is an existence exempt as far as possible from pain, and

as rich as possible in enjoyments, both in point of quantity and quality … This, being according to the utilitarian opinion, the end of human actions, is necessarily also the standard of morality; which may accordingly be defined, the rules and precepts for human conduct by the observance of which an existence such as has been described might be, to the greatest extent possible, secured to all mankind; and not to them only, but, so far as the nature of things admits, to the whole sentient creation.“ (Mill 1879, 5)

Mill behauptet damit, das einzige Ziel im Leben ist die Glück von man selbst und von allen anderen zu maximieren und die Taten oder Empfindungen die Glück behindern auszuschließen. Eine Entscheidung ist nur dann berechtigt, wenn sie das vollzieht. Darunter versteht Mill ein moralisches System; dieser Erklärung fehlt nur ein Verständnis von Glück. Es gibt viele Erfahrungen oder Aktivitäten, die ein empfindungsfähiges Wesen auf verschiedene Gründe vergnüglich finden kann. Essen, Sex, Trainieren, Lesen, Schreiben, Musik Komprimieren sowie anderen zu helfen oder glücklich zu machen sind Tätigkeiten, die alle Glück erzeugen können. Mill ordnet diese Empfindungen in Schichten nach der Intelligenz die man braucht, von den Tätigkeiten Glück zu gewinnen (Mill 1879, 4). Taten, die eine höhere Fähigkeit um die Folgen der Tat zu vergnügen benötigen, erzeugen mehr oder „bessere“ Glück als diejenigen, wobei eine niedrigere Fähigkeit notwendig sind: „It is better to be a human being dissatisfied than a pig satisfied; better to be Socrates dissatisfied than a fool satisfied“ (4). Glück ist deshalb für Mill relativ. Logischerweise ist diese Relativität notwendig, um sich für eine Glück über eine andere zu entscheiden.

3.2 Deontologie

Unter Deontologie versteht man wiederum ein System, das das Recht statt der Glück maximiert. Um uns zu entscheiden, ob eine Tat berechtigt oder nicht ist, betrachten wir nicht die Folge der Tat, sondern die Tat im Zusammenhang mit einem bestimmten moralischen System. Diese Theorie liegt gegenüber Consequentialism und gleichermaßen Utilitarismus. In der reinen Deontologie lässt man die Folge einer Tat, egal wie „gut“ sie ist, keine Rolle in der Entscheidung spielen: „Deontologists of all stripes hold that some choices cannot be justified by their effects – that no matter how morally good their consequences, some choices are morally forbidden … For deontologists, what makes a choice right is its conformity with a moral norm“

(Alexander 2008, 4). Deontologie ist deswegen von einem normalisierten moralischen System abhängig, bzw. auf ein moralisches System basierend. Diese Idee liegt im Gegensatz zu Utilitarismus, in dem die Moral eine Folge des Prinzips der größten Glück ist. Diese Idee lässt die Frage übrig, woher kommt dieses moralisches System? Eine solche normalisierte Moral folgt, könnte man sagen, aus der Gesellschaft. In der Diskussion von Softwareschutz gehen wir aber von einer wohldefinierten Moral aus; wir müssen von daher keine Gedanken darüber machen.

Wir werden uns nur auf die zwei grundlegenden Agent- und Patient-Centered (oder Täter- und Opfer-zentrierten) deontologischen Theorien beschränken. In Täter-zentrierter Deontologie beachtet der Täter seine moralischen Obligationen bei jeder Entscheidung: „At the heart of agent-centered theories (with their agent-relative reasons) is the idea of agency … Our categorical obligations are not to focus on how our actions cause or enable other agents to do evil; the focus of our categorical obligations is to keep our own agency free of moral taint“ (Alexander 1879, 5). In Opfer-zentrierter Deontologie nimmt der Täter die Rechte des Opfers in Betracht. Diese Theorie basiert auf die Idee, dass jeder Opfer das Recht hat, in einer Tat nicht verwendet zu werden, um positive Folgen zu ergeben: „Although all patient-centered deontological theories are properly characterized as theories premised on people's rights, perhaps the most plausible version posits, as its core right, the right against being used only as means for producing good consequences without one's permission“ (Alexander 1879, 10).

4 Software Schutzmöglichkeiten

Die juristische Grundlage von Softwareschutz ist das internationale Übereinkommen TRIPS (Trade-Related Aspects of Intellectual Property Rights)[1], das 1994 von den meisten westlichen Ländern unterschrieben wurde. Dieses Dokument definiert Quell- und Objektcode als geistiges Eigentum, der unter der Berner Übereinkunft von 1886, bei der die USA ab 1988 Mitglieder war[2] (Deutschland schon früher), mit einem Copyright versehen werden kann (TRIPS §1 art. 10). Unter diesem Übereinkommen sind Programme eigentlich in allen Formen (auch Maschinencode) als geistiges

1 Text zum TRIPS-Übereinkommen: http://www.wto.org/english/docs_e/legal_e/27-trips_01_e.htm
2 Berne Convention Implementation Act of 1988: http://www.copyright.gov/title17/92appii.html

Eigentum betrachtet: „Compilations of data or other material, whether in machine readable or other form, which by reason of the selection or arrangement of their contents constitute intellectual creations shall be protected as such“ (TRIPS §1 art. 10). Das heißt, Software, die man in einer bestimmten Weise oder zu einem bestimmten Zweck zusammensetzt, gilt als geistiges Eigentum. Obwohl Software Copyright unter TRIPS ziemlich klar und verständlich ist, bezieht sich das Dokument nicht direkt auf Softwarepatenten: „Subject to the provisions of paragraphs 2 and 3 [of this article], patents shall be available for any inventions, whether products or processes, in all fields of technology, provided that they are new, involve an inventive step and are capable of industrial application“ (§5 art. 27). Patentieren von Software ist deshalb unter dem TRIPS-Übereinkommen sehr interpretierbar und diese Lage hat zu interessanten Folgen im Bezug auf Softwarepatenten geführt; dazu kommen wir aber erst später.

4.1 Copyright

Das Copyright ist im Zusammenhang mit Software eine Art Kontrolle, wo der Entwickler die Verwendung eines bestimmten Computerprogramms (in der Form von Quellcode, Maschinencode, usw.) direkt kontrolliert. Das Programm wird unter TRIPS mittels einer Lizenz geschützt, die als ein Vertrag zwischen dem Entwickler und Enduser eines Programms fungiert und erklärt, wie die Software verwendet werden kann. Der Entwickler bestimmt durch diese Lizenz (gleichermaßen wie andere Arten geistiges Eigentums), ob der User das Programm kopieren oder weitergeben darf, wie oder wie oft der User das Programm benutzen kann, ob der Quellcode veröffentlicht wird oder nicht, usw. Durch die Lizenz kontrolliert man tatsächlich jede mögliche Aspekt der Software. Lawrence Lessig würde dies als perfekte Kontrolle bezeichnen: „Now this radical increase in control gets justified in the United States under the label of 'property': under the label of protecting property against theft. The idea has emerged that any use of copyrighted material contrary to the will of content controller is now theft. That perfect property is the ideal of intellectual property. That perfect control is its objective.“ (Lessig 10). Im Zusammenhang mit Software ist heutzutage die Tendenz, zwischen Eigentum und geistigem Eigentum nicht zu unterscheiden; d.h., wir behandeln in der Gesellschaft geistiges genau so wie physisches Eigentum.

4.1.1 Microsft-EULA

Eine sehr gewöhnliche Softwarelizenz ist die Microsoft-End User Software Agreement (EULA) für Windows Vista Starter.[3] Diese proprietäre Lizenz, die die Verwendung des Betriebssystems bestimmt, wurde mit der Ansicht geschrieben, dass sich geistiges und materialistisches Eigentum voneinander nicht unterscheiden. Die Lizenz vermeidet im Grunde genommen zwei Verwendungen: sie bestimmt erstens, dass eine Kopie der Software auf einen einzelnen Rechner installiert werden kann und jeweils von einem User benutzt werden kann; zweitens schützt die Lizenz den hinterlegenden Quellcode. Es wird nämlich als Copyrightverletzung bezeichnet, den Maschinencode in Assembler oder eine höhere Sprache zu übersetzen (reverse-engineering), um die Funktionsweise des Codes zu entdecken. Obwohl diese Lizenz für den User sehr beschränkend ist, erzeugt sie für den Entwickler zahlreiche Vorteile. Diese Beschränkungen können beide rein ökonomisch begründet werden; wenn man also erstens garantiert, dass jeder User der Software eine Lizenz gekauft hat, verkauft der Entwickler so viele Lizenzen wie möglich. Perfekte Kontrolle des Quellcodes ist auch eine gute Idee in diesem Sinne, denn man kann vermeiden, dass jemand anders eine reverse-enginnered Version der Software veröffentlicht oder verkauft.

Die Microsoft-EULA ist ein Beispiel einer Lizenz, die von keiner Moral ausgeht. Sie nimmt Rücksicht weder auf die Rechte des Users noch auf die moralischen Obligationen des Entwicklers und ist von daher nicht auf die Deontologie zu begründen. Es mag trotzdem möglich sein, sie in einer utilitaristischen Art und Weise zu erklären. Das „Gute“, das sie produziert, ist die Tatsache, dass man möglichst viel Geld mit Software unter der Lizenz verdient. Theoretisch erzeugt diese Idee ein besseres Produkt, das auch eventuell für den User „gut“ ist. Es gibt aber dazu eine nicht „gute“ Folge, nämlich, dass der User keinen Zugriff auf den Quellcode hat. Wenn man Zugriff hätte, könnte man den Code nach seinem eigenen Bedürfnis erweitern und verbessern. Mann muss denn annehmen, dass das Gute in diesem Falle überwiegt, bzw., dass ein besseres Produkt für den User wichtiger ist als das Erlaubnis, den Code zu modifizieren. Für die meisten Leute ist dies allerdings auch der Fall, weil relativ wenige Personen

3 Ich habe als Beispiel die Microsoft-EULA „Microsoft Software License Terms: Windows Vista Starter“ unter http://www.microsoft.com/About/Legal/EN/US/IntellectualProperty/UseTerms/ verwendet. Diese Version war in den USA damals günstiger als andere Versionen wie Home oder Professional und dafür auch mehr begrenzt.

programmieren können.

4.1.2 GPL

Man könnte aber dagegen argumentieren, dass besserer Code eher aus einer öffentlichen Gesellschaft als einer geschlossenen folgt und der Vorteil von proprietärer Software in einer anderen Art und Weise produziert werden kann. Diese Idee entspricht dem Konzept von open-source Software und der Ideal der GNU Public License (GPL).[4] Der Autor Richard Stallman hat die Lizenz im Gegensatz zu Lizenzen wie dem Microsoft-EULA und anderen proprietären Lizenzen entwickelt mit der Idee, dass eine Softwarelizenz garantieren soll, dass man den Werk frei (aber nicht notwendigerweise gratis) verwenden und weitergeben darf, solange man dasselbe Recht für anderen erhält. Die GPL listet sehr wenige Verwendungsbedingungen auf, jedenfalls viel weniger als die Microsoft EULA; sie erklärt aber genau, wie ein Programm unter der GPL modifiziert und weitergegeben werden soll:

„To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others. For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights" (GPL Vorwort)

Wenn man einen Werk mit der GPL lizenzieren lässt, überlegt man sich nicht nur seine Obligation als ein Entwickler, sondern auch die Rechte von den Personen, die seine Software verwenden werden. Offensichtlich hat die GPL sehr starke deontologischen Grundlagen, denn man ist dazu gezwungen, die Rechte für anderen zu garantieren, die man selbst bekommen hat. Das Recht in diesem Falle ist das grundlegende Prinzip hinter der Lizenz: man soll immer erlaubt sein, den Quellcode zu modifizieren, verbessern, und weitergeben. Wie die Microsoft-EULA ist diese Lizenz trotzdem nicht ohne Nachteile. Es ist erstens schwieriger aber nicht umöglich, Geld von Software unter der GPL zu verdienen. Obwohl der Quellcode gratis erhältlich sein muss, ist es möglich die Software in Maschinen- oder Objektcode zu verkaufen. Es gibt aber keine Art und

4 Text der GPL: http://www.gnu.org/copyleft/gpl.html

Weise, die Wiedergabe des Quellcodes zu vermeiden. Aus einer deontologischen Perspektive ist diese negative Folge aber nicht wichtig. Hauptsache hat man damit das moralische Recht vollzogen.

4.1.3 BSD

Diese zwei Lizenzen haben sehr andere Zeile und sind in eigener Art und Weise und von verschiedenen Perspektiven beschränkend. Für den Entwickler und Verkäufer ist die Microsoft-EULA ideal, denn es garantiert perfekte Kontrolle über das Produkt. Für den User/Entwickler ist die GPL ideal, solange man die gleichen Rechte von anderen respektiert. Ein interessanter und sehr kurzer Kompromiss ist die BSD Lizenz.[5] Die BSD Lizenz ist entworfen, so frei und einfach wie möglich zu sein. Die einzige Bedingungen lauten, dass (1) man dem originalen Entwickler des Programms Glauben schenken soll und (2) das Programm ohne Garantie weitergegeben wird.

4.2 Softwarepatent

Mit Copyright wird Software als geistiges Eigentum geschützt. Der Patent ist eine komplett andere Form von Softwarekontrolle, die mit der Software vollständig anders umgeht. Der Patent entstand zum ersten Mal um Erfinder zu unterstützen; es garantiert dem Erfinder für eine gewisse Zeit Kontrolle seiner Erfindung, damit er davon Geld verdienen kann. In seinem Text „Intellectual Property Is Still Property“ hat Frank Easterbrook das Konzept des Patents so erklärt: „The product itself, not the patent papers, usually discloses things. Inventors want and need patents only when disclosure is inevitable in the absence of protection“ (Easterbrook 1990, 114). Patenten sind notwendig um den Erfinder zu schützen, wenn die Funktionsweise des Produkts feststellbar oder komplett offensichtlich vom Produkt selbst ist. Wie wendet man also diese Idee auf Software an? Ein Patent schützt nicht die Erfindung, bzw. den physischen Gegenstand selbst, sondern das Konzept, bzw. Entwurf der Erfindung. Wir müssen uns im Zusammenhang mit einem Computerprogramm überlegen, was das Konzept eines Programms wäre. Wahrscheinlich wäre dies der hinterlegende Algorithmus. Ein Patent würde deswegen nicht die Implementierung eines Algorithmus (wie Copyright) schützen, sondern den Algorithmus selbst. Das heißt, wenn man einen bestimmten Algorithmus würde patentieren lassen, wäre es dann illegal, den Algorithmus ohne

5 BSD Erklärung: http://www.linfo.org/bsdlicense.html

Erlaubnis des Patenteigentümers zu implementieren, auch nicht wenn der Algorithmus komplett abhängig/isoliert entdeckt würde.

Wenn wir von diesem Verständnis von Softwarepatent ausgehen, treffen wir schon zahlreiche ethische Probleme. Es klingt zuerst problematisch, dass man überhaupt einen Algorithmus dürfte patentieren lassen. Algorithmen sind Folgen der Mathematik und existieren in diesem Sinne in der Natur. Auf diesem Grund wurde es allerdings 1972 in den USA im Prozess *Gottschalk v. Benson* entschieden, dass ein reiner Algorithmus kein Gegenstand des Patents sei.[6] Heute existieren Softwarepatente trotzdem immer noch, aber sie müssen in einer anderen Weise gerechtfertigt werden. Bevor wir mit einer ethischen Analyse des Softwarepatents anfangen, müssen wir zuerst Softwarepatente im historischen Zusammenhang und in ihre modernen Verwendung betrachten.

Wie es vorher darauf hingewiesen wurde, ist der legale und gesellschaftliche Umgang mit Softwarepatenten nicht klar definiert. Die Geschichte des Softwarepatents parallelisiert die Geschichte des modernen Computers. Nach *Gottschalk v. Benson* haben sich die notwendigen Voraussetzungen eines Softwarepatents schrittweise bis in den 90en verändert. 1998 wurde es in *State Street Bank & Trust v. Signature Financial Group, Inc.* letztendlich entschieden, dass Software und andere Verfahren nur dann patentierbar sind, wenn sie eine „useful, concrete and tangible result“ ergeben.[7] Diese ist eine sehr vage Definition, die sehr interpretierbar ist. Infogeldessen wurden Softwarepatente zur Zeit auch extrem vag geschrieben, damit sie unter dieser Definition als einen Gegenstand des Patents qualifizieren:

„Patent lawyers still have to pretend that's what they're doing when they patent algorithms. You must not use the word 'algorithm' in the title of a patent application, just as you must not use the word 'essays' in the title of a book. If you want to patent an algorithm, you have to frame it as a computer system executing that algorithm. Then it's mechanical; phew. The default euphemism for algorithm is 'system and method.' Try a patent search for that phrase and see how many results you get.“ (Graham 2006)

6 Zusammenfassung der Patenteierbarkeit von Algorithmen auf der Website des US Patent and Trademark Office: http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2106_02.htm

7 Text der Federal Court of Appeals Decision unter: http://www.bitlaw.com/source/cases/patent/State_Street_Bank.html

Obwohl die Situation im neuen Jahrtausend langsam angefangen hat, sich in die andere Richtung zu bewegen (siehe Supreme Court case *In re Bilski*), spielt Software Patentieren in den USA eine sehr große Rolle sowohl in Softwareentwicklung als auch in der Ökonomie. In seinem Aufsatz „Are Software Patents Evil?“ beschreibt Paul Graham, dass Patente in der modernen Software-business-world völlig integriert sind: „A closer comparison might be someone seeing a hockey game for the first time, realizing with shock that the players were deliberately bumping into one another, and deciding that one would on no account be so rude when playing hockey oneself. Hockey allows checking. It's part of the game. If your team refuses to do it, you simply lose. So it is in business. Under the present rules, patents are part of the game“ (Graham 2006).

In dem Aufsatz erklärt Graham, wie kleine Softwarefirmen (die sogenannte Start-ups) mit Patenten umgehen sollen. Seiner Meinung nach sind Patente notwendig für ein Unternehmen, um sich selbst zu schützen und in der modernen Geschäftsklima zu überleben (Graham 2006). Seit den 90en Jahren werden Softwarepatente verwendet, um Geld in einer sehr neuen und besonderen Art und Weise zu verdienen. „Patent trolls are companies consisting mainly of lawyers whose whole business is to accumulate patents and threaten to sue companies who actually make things … In the last couple years they've extracted hundreds of millions of dollars from [big companies]“ (Graham 2006). Obwohl nicht jeder Softwarepatenteigentümer wie die Trolls Geld verdient, ist diese Verwendung des Patents ein starkes Zeichen dafür, dass Patente im Zusammenhang mit Software mit der originalen Absicht nicht mehr verwendet werden. Der Patent wurde entworfen, dem Erfinder Kontrolle über seine Erfindung zu garantieren; heutzutage spielen sie die Rolle in Software, die Atomwaffen in einem Krieg spielen, insbesondere für große Unternehmen: „A lot of companies (Microsoft, for example) have been granted large numbers of preposterously over-broad patents, but they keep them mainly for defensive purposes. Like nuclear weapons, the main role of big companies' patent portfolios is to threaten anyone who attacks them with a counter-suit“ (Graham 2006).

Es gibt dennoch zahlreiche andere Aspekte der Softwarepatentdebatte, die alle in diese Arbeit nicht eingefügt werden kann. Es wurde aber zwei Probleme angezeigt, die wegen Softwarepatenten existieren: Software patentieren ist Algorithmus patentieren, und die

moderne Anwendung von diesen Patenten. Viele Leute sind der Meinung, dass Patente für Software überhaupt nicht berechtigt sind. Anderen glauben, dass dieser moderne Gebrauch von solchen Patenten vermeiden werden soll, aber Softwarepatente sind genau so wichtig wie andere Patente. Paul Graham behauptet, die Business-world funktioniert so und diese Funktionsweise ist nicht zu vermeiden; die Firmen, die nur Geld von Patenten verdienen, die Trolls, sollen aber illegal sein.

Die ethische Behandlung dieser Debatte ist allerdings sehr schwierig. Wenn wir Code als geistiges Eigentum behandelten, waren die ethischen Grundlagen der verschiedenen Softwarelizenzen ziemlich klar und verständlich. Die Diskussion über Patente hat aber mehrere Schichte, die alle eine entsprechende Grundfrage haben. Die erste Frage lautet einfach, sind Patente, im originalen Sinne des Wortes, ethisch? In der Opfer-zentrierten Deontologie könnte man die Kontrolle über eine Erfindung als das Recht des Erfinders betrachten; wir müssen Patente in die Gesellschaft einfügen, um die Rechte des Erfinders zu respektieren. Der Patent hat auch utilitaristischen Sinn, weil Patente Erfindung unterstützen. Gilt diese deontologische Idee im Zusammenhang mit Software? Wenn Patente ethisch sind, sind Softwarepatente auch nicht ethisch? Gibt es eine besondere Eigenschaft von Software, die dazu führt, dass Software nicht patentierbar ist?

Das erste Argument, Softwarepatent ist Algorithmuspatent, bedeutet, dass der Softwarepatent wissenschaftliche Erweiterung behindert. Die im vorherigen Abschnitt beschriebene Idee gilt auch in diesem Falle; wenn ein Programmierer ein Programm „erfunden“ hat, hat er die gleichen Rechte wie andere Erfinder. Aus einer alternativen deontologischen Perspektive könnten wir aber sagen, der Täter hätte die Obligation, wissenschaftlichen Fortschritt nicht zu vermeiden. Dann ist das Patentieren von Software nicht berechtigt. Eine utilitaristische Begründung von Softwarepatenten ist wegen dieses Argument auch schwierig. Wenn wir einerseits sie erlauben, behindern wir wissenschaftlichen Fortschritt. Wenn wir auf der anderen Seite solche Patente nicht erlauben, unterstützen wir Erfindung nicht. Die utilitaristische Frage lautet, welche Folge ist wichtiger oder besser? Welche Folge erzeugt die größte Glück?

Das zweite Argument das diskutiert wurde ist die moderne Wirkung von Patenten auf Softwareentwicklung. Dieses Problem führt aber zu einer neuen Frage: Ist es ethisch, in den Wörtern von Paul Graham, das Spiel zu spielen? Ist es ethisch Patente zu benutzen auf einen anderen Grund außer der originalen Absicht? Die Taten der sogenannten Patent-trolls sind offensichtlich nicht ethisch; ist es aber auch nicht ethisch an dem Kampf teilzunehmen, obgleich man nur überleben will? Obwohl Teilnehme im deontologischen Sinne wahrscheinlich nicht berechtigt ist. ist es dennoch vielleicht die beste utilitaristische Entscheidung. Wenn man nicht teilnimmt, ist es sehr wahrscheinlich, dass man sein Unternehmen verliert, ohne irgendwas positives für die Gesellschaft zu tun.

Ein drittes Argument gegen den Softwarepatent ist die Existenz des Copyrights. Wenn man anderen aus der Funktionsweise eines Algorithmus ausschließen will, bietet das Copyright nicht genug Schutz? Mit dem Copyright haben wir allerdings ein ähnliches Problem wie der Patent. Ist es ethisch, einen Algorithmus mittels eines Copyrights zu schützen? Es ist möglich herauszufinden was Software macht, die entweder mit einem Copyright versehen ist oder patentiert ist. Man kann dann versuchen, die Software selbst zu implementieren. Wenn man das Problem identisch aber ohne Kenntnis der Funktionsweise des Algorithmus löst, ist die Implementierung nicht Copyrightsverletzung sondern Patentverletzung. Dieser ist der pragmatische Unterschied zwischen den zwei Schutzmitteln. Um dieses Argument zu lösen, muss diese Verletzung ethisch begründet werden.

5 Schluss und sozialer Umgang

Wir haben somit zwei Arten von Softwareschutz, die von anderen Perspektiven ausgehen. Mit Copyright bezeichnet man Computercode als geistiges Eigentum genau wie ein Musikstück, eine Malerei, oder Literatur. Copyright garantiert dem Entwickler perfekte Kontrolle über sein Eigentum; man hat das Recht, anderen vom Zugriff auf den hinterlegenden Quellcode auszuschließen, die Wiedergabe des gesamten Programms zu vermeiden, den Gebrauch des Programms allgemein zu bestimmen, usw. Die Rechte des Users und Entwicklers anhand des Programms werden durch eine Lizenz erklärt. Proprietäre Lizenzen implementieren tatsächlich diese Kontrolle und obwohl sie den

User begrenzen, ergeben sie sehr oft gute Folgen. Die Microsoft-EULA ist eine solche Lizenz, die man utiltaristisch begründen könnte. Es gibt aber Lizenzen die von einer Moral ausgehen und gewisse Maßnahmen treffen, um dem User bestimmte Rechte zu garantieren. Die GPL ist ein Beispiel einer Lizenz, die eine starke deontologische Grundlage hat.

Der Patent im Zusammenhang mit Software ist ein Schutzmittel, dessen Grundlagen bisher sehr unscharf sind. Wenn wir der Patent auf Software anwenden, tauchen zahlreiche ethische Probleme auf, die bei dem Copyright überhaupt nicht existieren. Obwohl der Patent von einer Moral ausgeht, nämlich dass Erfinder Kontrolle über ihre Erfindungen haben sollen, ergibt er ein neues moralisches Problem: die Behinderung des wissenschaftlichen Fortschritts. Der Softwarepatent ist allerdings im Konflikt mit Copyright. Warum würde man einen Algorithmus patentieren lassen, wenn das Copyright perfekte Kontrolle über die Implementierung des Algorithmus bietet? Wenn man anderen aus einer Idee, bzw. Funktionsweise eines Algorithmus, ausschließen will, lässt man das Material bzw. Quellcode mit einem Copyright versehen. Dann ist es Copyright Verletzung, die Software in einer unbestimmten Weise zu benutzen. Wenn man wiederum den Algorithmus direkt patentieren lässt, ist es Verletzung, den Algorithmus sogar ohne Kenntnis des Patents zu implementieren. Der Patent bietet mehrere Kontrolle; ist diese größerer Kontrolle aber notwendig? Ist es berechtigt, diese Kontrolle zu erlauben?

Dieser Konflikt erfordert eine Lösung. Im Grunde genommen ist die Tatsache, dass das gleiche Material in zwei Weisen kontrolliert werden kann, selbst problematisch. Im Zusammenhang mit Software sollen eine Methode oder die andere ausgeschlossen werden. Ich würde persönlich behaupten, dass der Patent für Software, außer sehr seltenen Fällen, nicht notwendig ist. Die Softwarelizenz erlaubt jede Ebene Kontrolle, die man brauchen könnte. Diese würde aber bedeuten, dass Erfindung im Bereich von Informatik und den anderen Bereichen, die Informatik berührt, durch den Patent nicht unterstützt wird; ich bin aber der Meinung, dass das Erweiterung der Wissenschaft im utilitaristischen Sinne viel wichtiger ist. Thomas Jefferson hätte vielleicht sagen können, Innovation ist ein Teil der Natur, die sich immer weiter verbreitet und nicht zu

kontrollieren ist. Hinter Innovation sind nur Ideen.

6 Literaturverzeichnis

Alexander, Larry and Moore, Michael (2008): „Deontological Ethics" in: The Stanford Encyclopedia of Philosophy (Fall 2008 Edition), ed. Edward N. Zalta. <http://plato.stanford.edu/archives/fall2008/entries/ethics-deontological/>.

Easterbrook, Frank H. (1990): „Intellecutal Property Is Still Property“ in: Information Ethiscs: Privacy, Property and, Power, ed. Adam D. Moore, S 113-121.

Graham, Paul (2006): „Are Software Patents Evil?“. <http://www.paulgraham.com/softwarepatents.html>

Lessig, Lawrence (2000): „Open Code and Open Societies“.

Mill, Jon Stuart (1879). „Utilitarianism“. Reprinted From *Fraser's Magazine, 7th Edition.* Longmans, Green, and Co. London, 1879. 22. Feb 2004. <http://www.gutenberg.org/files/11224/11224-h/11224-h.htm>