University Software Ownership: Trends, Determinants, Issues

Arti Rai,[*] John Allison,[**] Bhaven Sampat,[***] and Colin Crossman[****]

1. Introduction

Software patents and university-owned patents represent two of the more controversial intellectual property developments of the past 25 years. For the last few years, the European Union has been embroiled in a fierce debate over the merits of explicitly allowing software patents.[1] In the U.S., where such patents clearly exist, various legal scholars have quarreled with the breadth of software patent claims.[2] Some have also suggested that, given the poor quality of prior art documentation and patent examiner training in the area of software, many issued software patents are likely to be obvious.[3] More generally, there is significant debate over the extent to which software patents are likely to foster innovation.[4] Because software products are often "complex" and may infringe many patents, some producers of end-product software are also

---

[*] Professor, Duke Law School; Faculty, Duke Institute for Genome Sciences and Policy. The authors gratefully acknowledge the support of the National Human Genome Research Institute and the Department of Energy (P50 HG003391-02).
[**] Professor, University of Texas
[***] Assistant Professor, Columbia
[****] Faculty Fellow, Duke Law School, 2005-2006

[1] As a formal matter, article 52(2)(c) of the European Patent Convention forbids patents on "programs for computers." However, this provision has been interpreted to cover only computer programs "as such." The European Patent Office allows software patents that have a "technical" character/effect. Moreover, in at least some cases, this technical character/effect has been found in the transformation that the program effects in the internal functioning of the computer on which it is runs. *See, e.g.*, Computer Program Product/IBM, T1173/97-3.5.1 (EPO Board of Appeals July 1, 1998)

[2] Dan Burk & Mark Lemley, *Policy Levers in Patent Law*, 89 VA. L. REV. __ (2003) (arguing that the Court of Appeals for the Federal Circuit has an exaggerated sense of the skill of the ordinary computer scientist and is therefore likely to allow broad patents); *see* Arti K. Rai, *Engaging Facts and Policy: A Multi-Institutional Approach to Patent System Reform*, 103 COLUM. L. REV. 1035, __ (2003) (arguing that because pure software patents are not limited to a particular physical machine or process, they may be problematic in terms of breadth). A recent case involving a university software patent suggests that the Federal Circuit may now be less inclined to allow broad scope in software patents. *See* LizardTech, Inc. v. Earth Resource Mapping, 424 F.3d 1336 (Fed. Cir. 2005) (holding patent claim that generically covered methods for eliminating edge artifacts created by the use of a particular type of digital compression technology invalid under Section 112). Even cases like *Lizard Tech* do not, however, require software patentees to disclose source code.

[3] *See, e.g.*, Glynn Lunney, *E-Obviousness, 7 Michigan Telecommunications & Technology L.R.* 363 (2001).

[4] *Compare* JAMES BESSEN & ROBERT M. HUNT, AN EMPIRICAL LOOK AT SOFTWARE PATENTS 7-9 (Fed. Res. Bank of Philadelphia Working Paper No. 03-17, available at < http://www.researchoninnovation.org/swpat.pdf > (arguing, based on analysis of software patents held in all industries, that such patents are likely to be substitutes for innovation) with Ronald Mann, *Do Patents Facilitate Financing in the Software Industry*, 83 TEX.L.REV. 961 (2005) (arguing that patents may promote entry of small, innovative software firms) and Robert Merges, *Patents, Entry, and Growth in the Software Industry* (working paper 2006) (patents facilitate the entry of new software firms and that incumbent software firms with "good" patents enjoy market success). These papers can be reconciled to the extent that Merges and Mann focus on the software industry while Bessen and Hunt look at software patents more generally.

unhappy.[5] Large incumbent software firms like IBM and Microsoft have been the prime movers behind recent legislative proposals that would make patents easier to challenge and injunctive relief, particularly by non-practicing patentees, more difficult to secure.[6] The controversy over software patents has also been fueled by the rise, and arguable success, of the "open source" movement in software. Open source software developers eschew patents. Although they do rely on intellectual property in the form of copyright, they do so for purposes of enunciating licenses under which source code is made freely available.

In contrast with software patents, university-owned patents existed before 1980. However, the scale of university patenting has increased substantially over the past 25 years, since the 1980 passage of the Bayh-Dole Act. While the legal question was sometimes murky prior to 1980, Bayh-Dole makes it unequivocally clear that universities can patent federally funded research.[7]

Increasingly assertive university patenting has attracted a fair amount of attention in both the scholarly and popular literature.[8] Additionally, because universities, and sometimes even their exclusive licensees, are non-practicing patentees the debate over whether such patentees act as "trolls" when they assert patents against successful commercializers has implications for universities.[9] Most studies have focused, however, on patenting within the life sciences.[10] This focus is hardly surprising, as the majority of university-owned patents (and certainly the majority of large revenue generators) appear

---

[5] In contrast, while products in the biopharmaceutical industry may require many patented inputs for their creation, the products *themselves* are likely to be covered by only a few patents. Thus one common strategy for avoiding patent thickets is secret infringement. *See* Walsh et al. (2003)

[6] *See* Patent Reform Act, H.R. 2795, 109th Cong. (2005). When it was originally introduced, H.R. 2795 contained language that might have made injunctive relief the exception rather than the rule. Opposition from the biotechnology and pharmaceutical industries led to elimination of this language. In a recent Supreme Court case, e-Bay v. MercExchange, the software industry supported e-Bay's argument that the Court of Appeals for the Federal Circuit granted injunctive relief too readily. The Supreme Court decision accepted this argument in part. ___ U.S. ___ (2006)

[7] Although not all university research is federally funded, the federal share represented 64% of university R&D in FY 2004. *See* www.nsf.gov/statistics/nsf06323/pdf/tab1.pdf. In computer science and electrical engineering, the federal percentage of university R&D in FY 2004 was even higher, 73% and 68% respectively. In contrast, industry accounts for only about 5% of university R&D (including CS/EE R&D). These percentages have been fairly consistent over the last 20 years.

We report these statistics to buttress our supposition that most software-related research done by universities is federally (or at least publicly) funded and thus subject to the types of economic analysis usually employed in thinking about publicly funded research. Unfortunately, although Bayh-Dole requires universities to report federal funding when they file for patents, many do not. Thus we can not rely on such reports to determine the funding source of research that led to specific software patents.

[8] For a popular account that is quite critical, *see* JENNIFER WASHBURN, UNIVERSITY, INC.: THE CORPORATE CORRUPTION OF HIGHER EDUCATION (2005). For a comprehensive analysis of available data on the impact of Bayh-Dole, *see* MOWERY ET AL., IVORY TOWER AND INDUSTRIAL INNOVATION (2004).

[9] Mark Lemley, *Are Universities Patent Trolls* (working paper 2006)

[10] *See, e.g.*, Bhaven Sampat, *Genome Patents: Bad for Science*? (working paper 2005); Rai & Eisenberg, *Bayh-Dole Reform and the Progress of Biomedicine* (2003); Pierre Azoulay et al., *The Determinants of Faculty Patenting Behavior: Demographics or Opportunities*; Fiona Murray and Scott Stern, *Do Formal Intellectual Property Rights Hinder the Free Flow of Scientific Knowledge?: An Empirical Test of the Anti-Commons Hypothesis*, NBER Working Paper 11465. A few researchers have compared the patent practices of life scientists with those of physical scientists. *See* Jason Owen-Smith and Walter W. Powell, *To Patent or Not: Faculty Decisions and Institutional Success at Technology Transfer* (discussing patenting practices of life scientists and physical scientists at two universities, one private and one public).

to emerge from the life sciences.[11] Moreover, the major economic argument put forward in the legislative history of the Bayh-Dole Act – that patents on publicly funded invention would promote commercialization of such invention[12] – would appear to apply most clearly to life science areas like drug development. The conventional wisdom is that, without the quasi-monopoly protection of a patent on the small molecule chemical, few firms would be interested in taking a potentially promising drug candidate through the expensive clinical trial and approval process.[13]

In the case of publicly funded software, by contrast, the need for a patent and exclusive license to promote further development is less immediately apparent.[14] Although development costs are not uniformly low, they are certainly low relative to those in the biopharmaceutical industry. Indeed, in certain cases of open source software development, firms derive revenue not from property rights over the software product itself but from a strategy that monetizes the value of support services and complementary hardware.[15]

Some recent events have suggested, however, that universities might in fact be interested in strong assertions of proprietary rights over software. In 2004, a district court upheld a $520.6 million jury verdict in a patent infringement suit against Microsoft brought by the University of California and its exclusive licensee, Eolas Technologies.[16] The software patent in question essentially covers interactive web browsing. Eolas, a one-person startup run by one of the University of California professors listed as an inventor on the patent, filed suit on the patent on February 2, 1999, three months after it issued. And the UC/Eolas lawsuit is not unique. On the contrary, as discussed further below, we have identified a significant number of cases in which a university, together with its start-up exclusive licensee, has sued firms (often large incumbent firms) that appear to have commercialized the technology in question without a need for exclusivity.

---

[11] Rai & Eisenberg, *American Scientist* 2003 (based on PTO and IPC classifications, about 50% of recent university patents are in the area of biotechnology and pharmaceuticals); *see also* MOWERY ET AL. at 99-111 (collecting data from the University of California, Stanford, and Columbia indicating predominance of biomedical invention in terms of patents and revenue regeneration).

[12] Interestingly, the text of Bayh-Dole does not prescribe, or even encourage, a particular licensing model. However, the legislative history of the bill that eventually became the Bayh-Dole Act, as well as that of similar bills that were being discussed at the time, indicates a focus on exclusive licenses. *See, e.g.*, H.R. REP. NO. 96-1307, pt. 1, at 3, 5 (1980) (noting importance of exclusive licensing for attracting capital necessary for development); S. REP. NO. 96-480, at 28 (1979) (noting that because nonexclusive licenses were generally viewed dismissively in the business community, "as no patent protection at all," only 4% of the 28,000 patents owned by the government had been licensed to private industry for development).

[13] Various empirical studies have underscored the critical role played by patents on end stage pharmaceutical products. *See, e.g.*, Wesley Cohen et al.., *Protecting Their Intellectual Assess: Appropriability Conditions and Why U.S. Manufacturing Firms Patent (or Not)*, NBER Working Paper No. 7552 (2000) (discussing the importance of patents relative to other mechanisms of appropriation in various industries and concluding that patents are, by far, most important in the pharmaceutical industry).

[14] Even in the life sciences, the availability of patents on improvements as well as the presence of absorptive capacity in commercial firms may diminish the need for exclusive licensing. *See* MOWERY ET AL. 158 (discussing case of Columbia's co-transformation technology).

[15] *See, e.g.*, A. Bonaccorsi and C. Rossi, *Why Open Source Software Can Succeed*, 32 RESEARCH POLICY 1243, 1249 (2003) (discussing software publishers like Red Hat as well as hardware manufactures like IBM).

[16] On appeal, the Federal Circuit ordered a new trial concerning the validity of the patent. Eolas Technologies Incorporated v. Microsoft Corp., 399 F.3d 1325 (Fed. Cir. 2005).

Universities have, of occasion, had disputes not only with large software firms but also with open source software developers. In 2001, a group of university bioinformaticians felt compelled to petition the National Institutes of Health ("NIH"), asking the agency to mandate that publicly funded bioinformatics software be distributed under an open source model. At about the same time, reports emerged of individual professors facing resistance to their attempts to operate their labs under an open source model.

This paper represents the first systematic study of which we are aware of university software ownership.[17] We rely in part on a unique, hand-curated database of university software patents. Our quantitative analysis focuses on patents primarily because there is no comprehensive data on the extent to which copyright is asserted by universities.[18] This quantitative analysis is supplemented by interviews conducted with technology transfer officers (particularly at universities that own large number of software patents) as well as academic scientists prominent in the open source movement. Through these interviews, we draw out the policies that major university software patentees have not only with respect to patents but also with respect to software ownership more generally.

The combination of our quantitative and qualitative inquiry yields a number of interesting results. First, software patents appear to represent a significant percentage of university patent holdings. Second, the factor that has had the biggest effect on software patenting is not R&D generally, or even computer science R&D in particular, but the overall "patent propensity" of the university – that is, the tendency of the university to seek patents.[19] These findings are reinforced by our qualitative results. Our interviews show that some universities view software as similar to other, more physical inventions. The difficulty with this view is that software may follow a different commercialization path than other inventions. Thus, it is perhaps not surprising that we see a fair number of litigated cases involving software patents, and that almost all of these appear to represent situations where the university and/or its exclusive licensee is asserting the patent against an entity that has commercialized successfully independent of the patent. The main rationale for supporting patenting would, therefore, appear to be the promotion of start-up businesses, presumably on the theory that start-ups, and market-based activities more generally, are likely to be move innovative than activities in large incumbent firms. However, whether patents are necessary for start-up promotion is not as clear in software as it is, for example, in the biotechnology industry. Moreover, in contrast with biotechnology, where copyright is not available, universities can use software copyright to achieve revenue generation goals.

---

[17] One study that touches on some related issues is Agrawal and Henderson's examination of the patenting practices of MIT electrical engineering/computer science faculty. Based on their research, they conclude that patenting is a "minority activity" for most faculty members in the EE/CS department (and the mechanical engineering department). Agrawal and Henderson, *Putting Patents in Context: Exploring Knowledge Transfer From MIT*, 48 MANAGEMENT SCIENCE 44 (2002).

[18] Copyright attaches as soon as the software is created. Because there is no need to register copyrights, it is difficult to know the total volume of university software protected by copyright.

[19] As we discuss further below, our data require us to treat the "university" as something of a black box. Thus we can not determine the extent to which patent propensity operates at the level of the researcher, who is induced by the pro-patent culture to disclose more inventions and push for patents on those inventions, or at the level of the technology transfer office, which files for a higher percentage of patents on invention disclosures.

2. University Software Patents: History and Methods of Identifying

We began by using the USPTO's *Cassis* database to identify all patents issued in 1982, 1987, 1992, 1997, and 2002 that were assigned to institutions classified as Research or Doctoral Universities in the Carnegie Commission of Higher Education's 1972 or 1994 reports. We chose these particular years because they span a series of critical shifts in the legal regime surrounding software produced at universities. Over these two decades, university patenting increased dramatically, from 385 patents in 1982 to 2946 patents in 2002. Patent jurisprudence in the area of software also evolved considerably during this period. Because this evolution is closely related to the manner in which we define the term "software patent," we lay it out in some detail below.

2.1 History

In the 1970s, the dominant intellectual property regime for software was copyright, not patent. A 1972 Supreme Court case, *Gottschalk v. Benson*, had appeared to reject software (in that case a computerized method for converting decimal numbers to binary numbers) as patentable subject matter on the grounds that patent law did not encompass abstract scientific or mathematical principles.[20] Several years later, in the 1976 Copyright Act, Congress expressly endorsed copyright as an appropriate protection regime for software.

The IP terrain shifted in the 1980s. In the 1981 decision *Diamond v. Diehr*, the Supreme Court gave its first clear indication that certain types of software-implemented inventions were patentable. *Diehr* narrowed *Gottschalk* by upholding as patentable subject matter a rubber-curing process that used software (specifically, software that implemented the Arrhenius equation) to calculate cure time. According to the *Diehr* Court, the physical transformation of the rubber "into a different state or thing" took the invention being claimed out of the realm of abstraction. Through the 1980s, the Court of Appeals for the Federal Circuit followed a test similar to that enunciated in *Diehr*. Under this test, if an invention's claims involved nothing more than an algorithm, then the invention could not be patented. However, if the claims involved a mathematical algorithm that was "applied to, or limited by, physical elements or process steps," such claims would constitute patentable subject matter. The overall message to patent attorneys was that software could be patented, but it had to be claimed as something else.

As the patent option was becoming more attractive, copyright was becoming much less so. In the early 1990s, a series of decisions from regional appellate courts[21] made it clear that copyright covered primarily the literal source code of the program.

[20] Although *Gottschalk* is generally considered a subject matter case, the Court may have been equally concerned with breadth – the patent in question was not restricted to any particular implementation of the algorithm. In general, as noted above, "pure" software patents of the type at issue in *Gottschalk* may be broader than patents covering software embedded in a particular machine.

[21] While all patent appeals go to the Federal Circuit, appeals from copyright cases go through the ordinary process of appeal to the regional courts of appeal.

Courts saw broader coverage as running contrary to the principle that copyright is supposed to cover only expression, not ideas or utilitarian functions.

Greater changes lay in store. In the 1994 case of *In re Alappat*, the Federal Circuit effectively eliminated the physicality limitation by arguing that subject matter criteria could be met by showing that the software created a new machine – a "special purpose" computer – when it was executed. Presumably all software would produce such a special purpose computer and hence be patentable. Four years later the Federal Circuit's 1998 decision in *State Street v. Signature Financial Group* explicitly rejected any special subject matter test for software, arguing that software (like all invention) is patentable so long as it produced a "useful" result. After *State Street*, there was no need for even the fig leaf of a physical machine or process.

2.2 Identifying University Software Patents

Given this history, it is perhaps not surprising that identifying "software" patents is very difficult. Even now, there is no universally accepted definition of what a software patent is. Moreover, neither the U.S. Patent and Trademark Office (PTO) classification system nor the International Patent Classification (IPC) system was designed for the purpose of grouping together patents that might be seen as software. Both systems focus on specific functions at a very low level of abstraction and are unsuitable for defining any technology area at a conceptual level. Additionally, even if these systems were suitable for identifying for defining a technology area, software is a critical element of inventions in so many disparate fields that it would be difficult to capture software inventions adequately through a classification system.

To our knowledge, there have been only a few significant efforts to identify a large data set of software patents.[22] An initial paper by Graham and Mowery[23] used the IPC system in an effort to develop a data set of software patents. Graham and Mowery did not attempt to define what a software patent was. Rather, they identified packaged software firms and studied the IPCs of patents issued to those firms. Specifically, they examined patents in the following IPC subclasses: G06F (subclasses 3,4,7,9,11,12,13, and 15), G06K (subclasses 9 and 15) and H04L (subclass 9). To further limit this set, Graham and Mowery focused only on patents owned by one of the top 100 U.S. software firms, as listed in the trade publication *Softletter*. In a more recent paper,[24] Graham and Mowery used U.S. Patent classes (343, 358, 382, 704, 707, 709, 710, 711, 713, 714, 715, and 717), again limiting patents in these classes to those owned by large software firms. The Graham and Mowery approach is not likely to be significantly overinclusive, particularly when limited to patents owned by packaged software firms. However, their

[22] We exclude from our discussion a recent paper by Iain Cockburn and Megan MacGarvie, which focuses on patents held by firms in 27 specific software markets. See Iain Cockburn and Megan MacGarvie, Entry, Exit and Patenting in the Software Industry, NBER Working Paper No. 12563

[23] Stuart J.H. Graham & David C. Mowery, *Intellectual Property Protection in the U.S. Software Industry*, in PATENTS IN THE KNOWLEDGE-BASED ECONOMY 219, 231-33 (Wesley M. Cohen & Stephen A Merrill, eds., The National Academies Press 2003).

[24] Stuart J.H. Graham & David C. Mowery, *Software Patents: Good News or Bad News*, in INTELLECTUAL PROPERTY IN FRONTIER INDUSTRIES: SOFTWARE AND BIOTECHNOLOGY (Robert Hahn, ed., AEI-Brookings 2005).

approach may be quite underinclusive, missing software patents assigned to other firms or patents assigned to these firms in other patent classes.[25]

Another significant effort to identify a large set of software patents, by Bessen and Hunt,[26] defines "software patent" to include patents on inventions in which the data processing algorithms are carried out by code either stored on a magnetic storage medium or embedded in chips ("firmware").[27] Rejecting the use of patent classifications,[28] Bessen and Hunt studied a random sample of patents and classified them according to

---

[25] To get a sense of possible Type I and Type errors, we assessed how the two Graham and Mowery (GM) approaches classified the 2,942 university patents issued in 2002. As shown in Tables 2 and 3, very few of the patents classified by us as non-software were classified as software by either the GM-IPC approach or the GM-PTO approach. However, the GM-IPC approach did not classify as software 86% of the patents we classified as software. Similarly, the GM-PTO approach did not classify as software 82% of the patents we classified as software. *See also* Bronwyn Hall and Meghan MacGarvie, *The Private Value of Software Patents*, NBER Working Paper 12195 (April 2006) (noting that a comparison of Graham-Mowery with earlier datasets manually compiled by Allison indicates that an approach that uses patent classifications misses about 50% of software patents).

[26] JAMES BESSEN & ROBERT M. HUNT, AN EMPIRICAL LOOK AT SOFTWARE PATENTS 7-9. (Fed. Res. Bank of Philadelphia Working Paper No. 03-17, available at < http://www.researchoninnovation.org/swpat.pdf >.

[27] As Bessen and Hunt note, *id*. at 9, one of the current authors, John Allison, earlier employed a definition of software patent that excluded firmware, including only inventions in which the code implementing the data processing algorithms are stored on a magnetic storage medium. *See* John R. Allison & Mark A. Lemley, *Who's Patenting What? An Empirical Exploration of Patent Prosecution*, 53 VAND. L. REV. 2099, 2110-11 (2000); John R. Allison & Mark A. Lemley, *The Growing Complexity of the U.S. Patent System*, 82 B.U. L. REV. 77, 89 (2002); John R. Allison & Emerson H. Tiller, *The Business Method Patent Myth*, 18 BERKELEY TECH. L.J. 987, 1029 (2003). The reasons for using this definition were a combination of initial doubt and compromise with a coauthor, followed by a need for consistency. Each of those articles made use of the same data set of 1,000 randomly selected patents issued between mid-1996 and mid-1998. After a great deal more experience gained from closely reading thousands of computer-related patents, Allison became firmly convinced that the definition should include firmware. When he used the same set of 1,000 randomly selected patents in a subsequent article, he studied each patent again and reclassified them using a definition that included firmware. *See* John R. Allison, Mark A. Lemley, Kimberly A. Moore, & R. Derek Trunkey, *Valuable Patents*, 92 GEO. L.J. 435 (2004) (definition not explicitly provided in article). Allison has used this more inclusive definition in studying almost 20,000 patents issued during 1998-2002 to almost 1,000 firms appearing in the Software 500 list in those years. *See infra* __.

[28] Bessen & Hunt, *supra* note --, at 10-11. The Bessen & Hunt definition of a software patent appears to include patents on inventions that "use" software as part of the invention, but excludes those that "use" off-the-shelf software:

> Our concept of software patent involves a logic algorithm for processing data that is implemented via stored instructions; that is, the logic is not "hard-wired." These instructions could reside on a disk or other storage medium or they could be stored in "firmware," that is, a read-only memory, as is typical of embedded software. But we want to exclude inventions that involve only off-the-shelf software—that is, the software must be at least novel in the sense of needing to be custom-coded, if not actually meeting the patent office standard for novelty.

Id. at 8.

their definition. Using characteristics of patents they found to fit their definition, they then developed a simple keyword search algorithm to identify software patents.[29]

There are, however, pitfalls associated with using automated keyword searches. From Allison's studies of thousands of computer-related patents, it is clear that the use of language in the titles, abstracts, written descriptions, and claims of patents, even in those dealing with the same area of technology, can be highly idiosyncratic among different patent owners. Moreover, software is a critical part of inventions in so many far-flung fields that reliance on particular search terms could produce a data set that has both Type I and Type II errors.[30]

Hall and MacGarvie's recent study on software patent value combines the methods of Bessen/Hunt and the first Graham/Mowery approach with another method of their own creation. The additional method they create encompasses patents that fall within the PTO subclasses to which patents owned by fifteen large software firms are assigned.[31] To assemble their data set, Hall and MacGarvie take the union of the Graham/Mowery and Hall/MacGarvie approach and intersect it with the Bessen/Hunt approach. Notably, they then check their results against two sets of software patents manually identified by Allison in connection with earlier studies.[32]

Our definition of a software patent is *a patent in which at least one claim element consists of data processing, regardless of whether the code carrying out that data processing is on a magnetic storage medium or embedded in a chip*.[33] Not only is it possible to apply the definition consistently, but it also captures the realities of claim drafting. It is common for all or most of the elements in a patent claim to cover the prior art, with only one or perhaps two elements covering the purportedly novel and nonobvious advance. One finds large numbers of patents owned by computer hardware makers, for example, the claims of which initially read as though they cover something

---

[29] The keyword search algorithm initially identifies a set of patents that use the words "software," "computer," or "program" in the claims or specification. Patents within the set that contain the words "semiconductor," "chip," "circuit," "circuitry," or "bus" are then excluded as are patents that contain the words "antigen," "antigenic" or "chromatography."

[30] Bessen and Hunt (BH) also identify substantial degrees of over- and underinclusiveness in the data set generated by their keyword search. *Id*. at 9. Table 1 uses university patents issued in 2002 to compare the BH approach with our own approach. The two approaches yield an approximately similar number of total patents (396 patents using our approach vs. 415 using the BH approach). However, 51% of the patents our approach identifies as software are not identified as such by the BH algorithm. Moreover, we classify as non-software 53% of the patents that BH classify as software. Similarly, one recent study that used software experts to read a sample of the BH patents asserts that more than 50% represented Type II errors. Layne-Farrar (2005).

[31] Bronwyn H. Hall & Megan MacGarvie, *The Private Value of Software Patents*, NBER Working Paper 12195 (April 2006), at 13-18.

[32] Id. at 15-16.

[33] Allison also uses this definition in a collaboration with Ronald Mann that has involved reading every patent issued from Jan. 1, 1998 to Dec. 31, 2002 to the almost 1,000 firms that appeared in Software Magazine's annual "Software 500" list at least once during that five-year period. This list ranks firms according to their gross revenues in software and services, and includes many firms that are primarily manufacturers in addition to firms that produce only software.

like a generic router, printer, magnetic resonance imaging machine, or other hardware, when in fact the only purported novelty is in one element consisting of a function carried out by an algorithm. Some of this may be a consequence of the fact that, prior to *Alappat*, software had to be claimed as part of a physical invention other than a general purpose computer.

An obvious problem with our approach is that it involves the slow and laborious process of reading patents. Although the decision with many patents is clear, there will always be a substantial percentage that must be studied with great care.[34] Claims are often quite obtuse, and in the computer field they are frequently broad, necessitating a close reading of not only independent but also dependent claims and, not uncommonly, resort to the specification to help interpret claim language. Moreover, a degree of subjective judgment is occasionally required. However, at least in the case of a relatively small data set, we believe that increased accuracy more than compensates for time-intensity and absence of algorithmic criteria readily replicable by automated methods. We do not claim that our data set of university-owned software patents is perfect, but we do contend that our error rate is small, certainly far smaller than in any data set acquired by means of patent classifications or keyword searches.[35]

Two final points are in order. First, in a large set of patents it is impossible as a practical matter to include only those patents in which the only software element (as contrasted with other elements of the invention) is novel and nonobvious. In order to restrict ourselves to patent in which the software element was novel and nonobvious, we would have to have a person having ordinary skill in the art conduct a very thorough study of the relevant prior art. Even then, the question would still be plagued by a degree of doubt. But the fact that, under our definition, the data processing must be identified in a claim element does suggest that software is sufficiently important to novelty and nonbviousness for the patent claim drafter to include it as a limitation of the claim (thereby narrowing the claim to some extent). Second, in addition to identifying which patents out of the more than 7,600 university-owned patents in our sample are software patents, we also identified a subset of those that may be called "pure software patents." These are *patents in which the claims consist only of data processing*. That is, the entire invention consists of algorithms. (These sorts of patents could presumably issue with any frequency only after the Federal Circuit's 1994 decision in *In re Alappat*.) This task required thorough study of each of the patents that had already been identified as a software patent. Although the process of identifying "pure" software patents was accomplished with a high degree of accuracy, there was a small number about which reasonable minds could differ. Thus, for this second stage, we also do not profess to have achieved perfection, but we do maintain that our error rate is low.

---

[34] However, if one is studying a large population of patents from the computer-related industries, the percentage that must be carefully scrutinized is far higher than if one is studying a population of patents across a broad array of fields (as in this paper).

[35] Layne-Farrar (2005) reports that when software experts read a random sample of patents from an earlier set identified by Allison, she found that only 5% represented Type II errors. Although the Layne-Farrar analysis could not identify Type I errors, we believe that any errors of underinclusiveness in our data set are similarly small.

3 University Software Patenting: Quantitative Trends and Determinants

3.1 Overview

Figure 1 shows that university software patenting increased more than ten-fold over the 1982-2002 period, from 37 patents in 1982 to 396 patents in 2002. Over this period, the "pure software" proportion of university software patents also increased dramatically, from 13 percent to 32 percent of all university software patents. (Figure 2) The latter change is hardly surprising. While the patentability of pure software was unclear in the 1980s, its status became much more secure in the 1990s. Over these two decades, university software patenting also grew at a faster rate than university patenting overall. As a consequence, the software share of university patents rose from 9 percent in 1982 to 13 percent in 2002, as seen in Figure 3.

Table Four lists the 15 universities that received the most software patents in 2002.[36] Together, these 15 institutions accounted for 60 percent of all university software patents issued in that year. The top five institutions alone (MIT, the University of California, Stanford, Caltech, and the University of Texas) account for over a third (34.2 percent) of all university software patents. The top five patentees also represent the top five university patentees overall in 2002. However, moving further down the list of the top fifteen, we see that a number of the top software patentees are not among the top university patentees overall. The University of Washington (6^{th} in software patenting/15^{th} in overall patenting), Georgia Tech (8^{th}/20^{th}), Carnegie Mellon (9^{th}/51^{st}), the University of Rochester (12^{th}/50^{th}) and the University of Illinois (14^{th}/28^{th}) particularly stand out as institutions substantially more prominent in software patenting than overall patenting.

With respect to the patenting of "pure" software, Table Four shows that the top three patenting institutions overall also rank among the top recipients of pure software patents (1st, 2nd, and 4th). In contrast, although Caltech and the University of Texas rank high in overall software patenting (4^{th} and 5^{th} respectively), they rank relatively low in the patenting of pure software (23^{rd} and 41^{st} respectively). University of Washington, Georgia Tech, Carnegie Mellon, the University of Rochester, and the University of Illinois – mentioned earlier as standing out in software patenting relative to overall patenting – also stand out with respect to numbers of pure software patents (6th, 3rd, 13^{th}, 14^{th}, and 11th respectively).

As these examples suggest, several factors may affect university software patenting. First, the amount of software related research and development, and thus the output of software, may matter. Second, the size of the overall research enterprise may matter, since software can be developed in many parts of the university. Third, an individual university's overall propensity to patent, i.e. the share of research outputs it patents (either because its researchers are prone to file invention disclosure statements and push for patents on those disclosures, or because technology transfer officers are

[36] To be sure, the 2002 data may be somewhat unusual in that it reflects patent filings that occurred during the "dot-com" bubble of the late 1990s. However, with a few exceptions, these universities also received the largest number of software patents over the course of the sampling.

prone to seek patents), may also affect software patenting.[37] Although there are undoubtedly other factors that affect software patenting – for example, the attentiveness to the specific question of patenting software in the technology licensing office or among faculty[38] – these are difficult to measure. However, we examine some of these factors in in our qualitative analyses.

3.2 Simple Regression Analyses

To generate the relative importance of 1) software-related R&D, 2) overall R&D, and 3) university propensity to patent, we estimated simple "patent production functions" relating software patent counts in 2002, 1997, 1992, 1987, and 1982 to characteristics of each of the 202 Carnegie universities in our sample. Specifically, we collected data on total research expenditures and computer science research expenditures from the NSF's *Survey of R&D Expenditures at Universities and Colleges*[39], and aggregated these data to create R&D stocks for the 5 year period prior to patent issue.[40] We estimated university patent propensity by measuring the number of non-software patents each university was issued in each issue year.[41] To facilitate interpretation, we took natural logarithms of the independent variables.[42] Since the dependent variables are integer valued, we estimated negative binomial regressions relating software patents and pure software patents to research expenditures.[43]

---

[37] In our quantitative discussion, we can not distinguish between motivations of university technology transfer officers and motivations of university scientists. We discuss some of these issues in our qualitative portion of the paper. For a discussion of scale economies in academic patenting and licensing activities, see D. Mowery and Bhaven N. Sampat. "Patenting and Licensing University Inventions: Lessons from the History of Research Corporation" *Industrial and Corporate Change* 2001. See also Azoulay 2005 (finding, in study of 3884 life science researchers, that the overall "patent stock" of the university where the researcher is employed has an effect on number of patents held by the life sciences researcher).

[38] In the life sciences context, for example, one study that looked at patenting activity for 3884 researchers found that having co-authors who patent has a positive effect on patenting behavior. Azoulay 2005.

[39] This survey is conducted annually by the NSF Division of Science Resources Statistics, and includes R&D expenditures from all sources of funding. Also helpful for our purposes is the fact that this survey breaks down funding by science and engineering field and by source of funding (federal and non-federal).

[40] Similar results obtain if we aggregate computer science and engineering R&D. Because we sampled on issue years, and there is a lag between patent application and issue, as well as a lag between research and patent application, choosing the appropriate lag period was difficult. Accordingly, we experimented with various windows. All specifications (available upon request) yielded qualitatively similar results.

[41] Using non-software patents as a measure of university patent propensity could skew results to the extent there were universities for which the majority of patents were software patents. In those cases, patent propensity might appear artificially low. However, one can not use the same variable on both sides of the regression. Fortunately, software patents did not represent a majority of patenting activity for any Carnegie university. We did attempt to find data on an additional proxy for patent propensity: the size of the technology transfer office. However, we were able to find data on size for fewer than half of the universities in our sample.

[42] In some cases, one or more of the right-hand side variables was zero, and the natural log of zero is undefined. Accordingly, we took natural logs of $1 plus R&D.

[43] We could not reject the hypothesis of overdispersion, and thus chose negative binomial models over Poisson models. However, we obtained qualitatively similar results from Poisson models with standard errors adjusted to account for overdispersion, and from log-log models estimated via ordinary least squares. These results are available from the authors on request.

Tables Five through Nine show the main results from these simple cross-sectional regressions. In negative binomial models, coefficients on log-transformed variables can be interpreted as elasticities. Model 1 of Table 5 shows that both computer science R&D and overall R&D have a positive and statistically significant impact on software patenting, but that the elasticity of software patenting with respect to computer science[44] R&D is much smaller. In particular, a 1 percent increase in computer science R&D implies a .11 percent increase in software patenting, but a 1 percent increase in non-computer science R&D implies a .84 percent increase in software patenting.

One explanation for this is that a substantial number of software patents result from non-computer science R&D, consistent with the argument that software is produced across the university. Graham and Mowery, for example, have noted that software is likely to be produced throughout the university.[45] Similarly, outside the university context, commentators have argued that many software patents are held by manufacturing firms.[46]

However, it is also likely that universities with more research simply have larger or more sophisticated technology transfer efforts, and higher propensities to patent for a given amount of software related R&D. To assess this, Model 2 controls for the total number of non-software patents issued to the university in 2002. After controlling for total non-software patenting, the impact of the amount of non-computer science R&D on patenting drops dramatically, and becomes statistically insignificant. However, computer science R&D remains positive and statistically significant. Perhaps most notably, overall patenting has a large and statistically significant impact on software patenting, even after controlling for overall and computer science R&D. While a 1% increase in computer science R&D yields a .097% increase in software patents, a 1% increase in non-software patenting yields a .91% increase in software patenting.

Models 3 and 4 in Table Five show similar regressions with the number of "pure" software patents as the dependent variable. Perhaps not surprisingly, the amount of computer science R&D has a much larger impact on "pure" software patenting than on software patenting overall. In the first specification (Model 3), a 1 percent increase in computer science R&D implies a .59 percent increase in pure software patents, more than five times larger than the effect on all software patents. Controlling for the amount of non-software patenting (in Model 4) reduces this effect somewhat, to .52%, but it is still much larger than the corresponding estimate for all software patents. On the other hand, Model 4 also shows that overall non-software patenting (after controlling for non-software R&D) still has a greater effect on "pure" software patenting. A 1% increase in number of non-software patents yields a .78% increase in pure software patents.

Table Six shows qualitatively similar results for the 1997 cohort. Perhaps not surprisingly (given the uncertain status of software patents), results on the importance of computer science R&D for pure software patents in the 1982-1992 cohorts are more mixed. However, with the exception of the 1982 cohort, overall patent propensity remains a statistically and qualitatively significant determination of patenting for both software and pure software.

---

[45] Graham and Mowery (2003).
[46] Bessen and Hunt

We also estimated a panel version of these regressions for the entire period, with university and year fixed effects. In this model, the estimated changes in R&D and other patenting are identified using within university variation over time. Table 10 shows the results. Notably, the year dummies and positive and their magnitude increases over time, reflecting the overall growth in university patenting over the 1982-2002 period. Model 1 shows that the main factor affecting overall software patenting is changes in non-software patenting , with a 1 percent increase in non-software patenting implying a .46 percent increase in software patenting. The corresponding for pure software patenting is neither qualitatively nor statistically significant. Moreover, none of the R&D measures in statistically significant for either software or pure software. This last set of results must be interpreted with caution, however. Because the panels are short and there is limited within university variation, it is difficult to draw strong inferences.

3.3 Departmental Origin

To explore the sources of software patenting in greater detail, we attempted to identify the departmental origin of the inventors on the top fifteen academic software patentees in 2002. As discussed above, these institutions accounted for 60 percent of overall academic software patenting in that year.

These 241 patents had 544 distinct inventors. Based on web searches, we were able to locate the primary departmental affiliation for 74 percent (400) of these inventors. Seen another way, these 241 patents include 661 unique inventor-patent dyad. (For example, if an inventor is included on 3 patents, she generates 3 inventor-patent dyads.) For 27 percent of these dyads, we could not identify the inventor's department.

Table Six shows the top 10 departments for dyads where the department is known. Slightly less than 30 percent of the inventors are from computer science, electrical engineering, and joint EE/CS departments. Moreover, consistent with arguments that software is produced across the university, a number of biomedical departments are also represented, including Neuroscience, Radiology, and Medical Physics. Most of the patents from Neuroscience are speech recognition software and software implemented techniques for training individuals to read and write, and the bulk of the patents from Radiology and Medical Physics relate to image processing and radiation therapy techniques. The Robotics Institute (at Carnegie Mellon) and Lincoln Labs (at MIT) also appear on the top ten list.

Table Seven shows the analogous table for pure software patents. Perhaps not surprisingly, more than half of the patent-inventor dyads on pure software patents emanate from computer science, electrical engineering, or joint CS/EE departments. Other departments are less prominent in production of pure software patents than they are in the production of other software patents.

Overall, these results suggests that university software patents, and particularly pure software patents, are quite likely to come for computer science and electrical engineering departments. While this result may be surprising to those who believe that computer scientists have different attitudes towards patenting than other researchers, we can not draw any definitive conclusions from this data. To examine the issue properly, we would, at a minimum, need to examine patenting propensity (patents per R&D dollar) across a variety of different disciplines. In any event, making generalizations about

patenting by computer science researchers may be difficult. As the regression results suggest, and as we discuss further in the next section, there is considerable heterogeneity across universities in terms of their policies and practices towards software. These heterogeneous policies may sometimes reflect, and sometimes trump, individual faculty members' own preferences.

4 University Ownership Policies

We interviewed technology transfer managers responsible for software at thirteen of the fifteen universities that received the most software patents in 2002. We also conducted interviews with university professors and graduate students prominent in the open source software movement. Finally, we discussed the general phenomenon of university software ownership with several technology transfer officers who are widely seen as having developed pioneering models for university ownership of digital information.

According to Gerald Barnett, one of these pioneering individuals (formerly at University of Washington, now at UC Santa Cruz), university technology licensing offices that have a long history of patenting tend to see software, including pure software, through the lens they use for other inventions, particularly in the life sciences.[47] In the life sciences, established technology transfer offices have long generated revenue not only through exclusive licensing (the model contemplated in the legislative history of Bayh-Dole) but also through nonexclusive licensing (a model not necessarily contemplated by Bayh-Dole but one that can generate substantial revenue).

Barnett's perspective is in accord with results from our interviews with officials at some major technology transfer offices. Lita Nelsen, director of MIT's technology transfer office notes (speaking of pure software),

[I]f there are no strong feelings on the part of the authors to open source their work, we will look at it *like any other invention*: is it worth investing time and, when appropriate, patent money to try to license the software out (either as simple end use license or to a distributor or startup company to improve and distribute it.)[48]

Barnett's view is also in accord with our regression results, which indicate that overall university patent propensity strongly influences both overall and pure software patenting. As noted earlier, the top 5 software patentees (MIT, UC, Stanford, Caltech, and University of Texas) are also the top 5 overall patentees. Notably, none of these five was in the top five in computer science R&D spending for the years 1996-1998.

However, of these five universities, two – Caltech and the University of Texas – do *not* have significant numbers of pure software patents. (Caltech and the University of Texas ranked 23rd and 41st respectively.) At the University of Texas, the reason for this dearth of pure software patents may be attributable to the view, explicitly adopted by UT technology transfer officials in response to complaints from computer science faculty, that faculty should be allowed freedom to share their work as they wanted. Such freedom

[47] Interview with Gerald Barnett, October 2, 2003.

[48] E-mail communication from Lita Nelsen, July 27, 2005 (emphasis added). According to Nelsen, software patents tend to be worth patenting when the primary value is in the algorithm.

includes express permission to use the GNU General Public License, a "viral" open source license under which source code is open to all but those who redistribute the software must also make any modifications they have made to the source code available. This policy was adopted in the 1990s and would therefore have affected the number of pure software patents that issued in 2002.[49] The explanation for Caltech's small number of pure software patents is less transparent. Because we were unable to speak with the technology licensing office,[50] our knowledge of the Caltech situation is based on reports from scientists who work there. According to one prominent open source software developer at Caltech, most software is for internal, in-house use and is probably not even reported to the technology transfer officer. For software that is reported, the faculty researcher decides how the software should be exploited. The technology transfer office is not only familiar with open source but is apparently quite eager to use the viral GPL license.[51] This is apparently because using the GPL license allows for the future possibility of forking, with one fork continuing to be available without charge via the GPL and the other fork converted into a non-viral license for which corporate client might be willing to pay.[52]

In contrast, the positions of MIT and the University of California are less explicitly favorable to open source.[53] According to Lita Nelsen, MIT allows researchers to use the open source approach, and even manages their licenses, but this position is not part of official policy.[54] The University of California system has a highly complex set of positions on open source licensing. This complexity emerges in part from the quasi-federated structure of the UC system, within which copyright licensing (and hence open source licensing) is determined by the individual campus. While some campuses, such as Berkeley and San Diego, are familiar with the open source model, other campuses are less familiar with the model.[55] Complexity is compounded by the fact that various important campuses, including UC Berkeley, appear to be in the midst of fine-tuning their policies. As of early 2002, officials from the Berkeley office were quoted as criticizing the decision, made by Berkeley a decade earlier, in 1992, to release as open source code the Unix operating system and TCP/IP networking protocol.[56] At about the same time, UC Berkeley computational biologist Steven Brenner encountered some difficulty in negotiating to release his lab's software under an open source license.[57] In late 2003, Veronica Lanier of the UC Berkeley licensing office stated that if the software embodied a patentable algorithm, the default approach would be patent rather than open

[49] Interview with Georgia Harper, Office of General Counsel, University of Texas
[50] The Caltech staff refused requests for an interview.
[51] Interview with C. Titus Brown.
[52] Id. Example of MySQL
[53] Where Stanford University fits in the picture is not clear. Stanford did not adopt an explicit policy in favor of open source until 2004. It adopted this policy in response to a number of requests received from professors. Interview with Kathy Ku. However, Stanford has long had a policy allowing professors to put their inventions into the public domain if they so desire. Cite to Stanford IP policy.
[54] Interview with Lita Nelsen, August 8, 2005.
[55] Interview with William Decker, UCSD (noting that Berkeley and San Diego often use open source licenses but that other campuses are not). Cf. Interview with Joel Kirschbaum, UCSF (noting that although his office does not use open source, they often make executable code available to academics free of charge)
[56] Jeffrey Benner, Public Money, Private Code, Salon, January 4, 2002 (quoting Bill Hoskins)
[57] Interview with Steven Brenner, March 2004.

source.[58] By 2004, however, UC Berkeley had announced a policy that appeared to give a much stronger endorsement to the open source model. In this new policy, the Berkeley technology transfer office states that it will work with researchers who want to release their code under an open source model.[59] Although it is not clear whether researchers have the final say in cases of conflict (e.g. if the office believes the software will be commercially valuable and wants to patent or copyright using a commercial license) the new policy appears to be encouraging towards open source. Berkeley has also developed, and encourages researchers to use, an "academic" license, under which source code is made available free of charge to academic and non-profits and made available for a fee for commercial use. In general, the new Berkeley policy appears to encourage the possibility of releasing software (and/or underlying source code) under different types of licenses for different purposes.

As noted earlier, five universities – University of Washington, Georgia Institute of Technology, Carnegie Mellon, the University of Rochester, and the University of Illinois – rank substantially higher in software/pure software patenting than in overall patenting. The first three of these schools have unique characteristics that may explain their high levels of pure software patenting. At the University of Washington, a separate technology licensing office, now called the Digital Ventures office, is responsible for managing pure software (and digital products more generally). University of Washington appears to be the only university in the country with an office specifically devoted to software. This office now has 7 full time professionals as well as 2-4 part-time students. As for Georgia Tech, its level of computer science R&D funding relative to other funding is quite high – it ranks 5th in computer science funding and only 32nd in other funding. Finally, Carnegie Mellon ranks 2nd in computer science funding and 87th in other funding. Carnegie Mellon is also home to the Software Engineering Institute (SEI), a research consortium founded by software firms. Patents from SEI are assigned to Carnegie Mellon, and the software firms in the consortium receive an automatic, royalty-free license.[60]

Notably, although University of Washington, Georgia Tech, and Carnegie Mellon have a significant patent presence, they do promote other models of software ownership. At the University of Washington, 90% of the revenue brought in by its Digital Ventures office in FY 2005 came from copyright licensing. Perhaps not surprisingly, the Digital Ventures web site features a large variety of unpatented software that can be secured for commercial use through an on-line license with a standard rate. Like Berkeley, the University of Washington also encourages "forking" in its licenses i.e. licenses in which software (and/or source code) is made available free of charge for certain uses and available for a fee for other uses. Georgia Tech's Kevin Wozniak estimates that he oversees a few dozen open source-type software licenses each year. He also notes that, given the short technology life cycle in the software industry, algorithm patents are often not very valuable. Finally, at Carnegie Mellon, the current technology transfer team

[58] Interview with Veronica Lanier, October 14, 2003

[59] Office of Technology Licensing, UC Berkeley, Frequently Asked Questions, http://otl.berkeley.edu/about/faqs.php (discussing open source options available on Berkeley's software disclosure form).

[60] Interview with Carl Mahler

almost never patents unless another entity (in their case, often SEI) is willing to pay for such patenting.

5 University Software Ownership: A Policy Analysis

The results above suggest that universities have become more active patentees of software, including pure software. They have availed themselves of the opportunities afforded by Federal Circuit decisions in the 1990s. Moreover, academic software patenting behavior with respect to overall software and pure software is strongly affected by overall patent propensity. From a private point of view, this correlation may make sense, especially if the reason for the effect is scale economies at the technology transfer office. To the extent such scale economies exist, they are likely to lower the private marginal cost of patent acquisition. From a social point of view, however, patenting based on scale economies is problematic. Ideally, we would want decisions about whether to patent publicly-funded academic research to be based not only on the private marginal costs of patent acquisition but on whether a patent is needed to facilitate commercialization in a specific case, which is likely to vary across inventions and fields. To put the point another way, a lack of differentiation between software and other research could be problematic, as the optimal mode of university-industry technology transfer is likely to vary by industry and invention.[61] Indeed, as we have suggested above, the theory espoused in the legislative history of – that patents and exclusive licenses are necessary to create incentives for firms to develop and commercialize "embryonic" university inventions—does not apply neatly in software, where development costs are often low.[62]

Another major argument often advanced in favor of patents is that the prospect of licensing royalties induces university researchers to work with industry licensees and thereby transfer tacit knowledge necessary for commercialization.[63] Although this argument could in theory be compatible with exclusive or non-exclusive licenses, the assumption tends to be that an academic researcher would have sufficient time for an exclusive licensee only. However, in comparison to the life sciences, software (particularly pure software) is an area of invention where knowledge is likely to be

---

[61] See, e.g, Colyvas et al. 2001. In addition to differential incentive effects, university patents and licenses have different informational effects across different industries. Of particular relevance to our study, the comprehensive Carnegie-Mellon survey, conducted on a broad range of large and small firms in the early 1990s, indicates that outside of the pharmaceutical and biotechnology industries, industrial R&D managers rate patents and licenses very low relative to other sources of information on public research (e.g. publications, conferences, informal interaction with university research, and consulting). Cohen, Nelson, and Walsh (2002). Even within the pharmaceutical industry, patents and licenses were less important than research publications and conferences.

[62] To be sure, such costs may be higher in situations where the software is not "pure software." However, even in those cases, to the extent that the novel or nonobvious element is probably software, development costs are probably still low relative to the biopharmaceutical industry.

[63] The evidence generally cited for this argument is survey data presented in Jensen and Thursby (2001). The Jensen and Thursby survey of 62 technology transfer offices found that TTO managers thought that inventor involvement was often important in the commercialization of inventions. Presumably patents and licensing royalties represent the most efficient contractual mechanism for inducing transfer of tacit knowledge. For discussion of this issue, *see* Arti K. Rai, Collaboration, Innovation, and the Firm (working paper 2006).

relatively codified. Object-oriented programming is based on principles of modular design, and one of the reasons that open source methods of software production have been successful is that the development task can be broken up into modular pieces that are then reassembled.[64] So the need for transfer of tacit knowledge may not be as pervasive as it is in the life sciences.

Indeed, in some well known cases, such as the Eolas case noted earlier, it appears that the university patent allowed the university and/or its exclusive licensee to extract rents from other firms without aiding in technology transfer. In the Eolas case, it does not appear that Microsoft's commercialization was aided by the activities of UC/Eolas. Rather, Microsoft and other firms began to use the browser technology at issue in the case well before the patent issued.[65] In order to study the litigation question more systematically, we conducted two, somewhat related empirical inquiries. First, we used a variety of different search methods to collect case studies involving university software litigation.[66] Second, we determined whether the software patents in our sample were litigated at rates higher (or lower) than the non-software patents in our sample.[67]

There was no statistically significant difference in rates of litigation between the software and non-software patents in our sample. However, we did find a nontrivial number of cases in which university software patents appeared to have been used in a manner that arguably hindered rather than promoted commercialization. Research Corporation Technologies ("RCT"), a firm that is the assignee of patents from the University of Rochester, has actively pursued litigation on a group of six patents covering the so-called Blue Noise Mask printing technology. This technology, which was developed at the University of Rochester and assigned to RCT under a technology evaluation and commercialization agreement dating back to 1953, allows for high-quality "half-tone" printing, is used by firms ranging from Hewlett Packard to Microsoft. The firm's web site emphasizes the patent suits it has brought against HP, Epson, and Microsoft and "invites current and potential users of this landmark technology to contact us about licenses under RCT's patent rights."[68] The lawsuits against HP and Epson were settled in 1999 and 2002 respectively. However, in a suit against Microsoft, a district court in Arizona founds that three of the six patents asserted by RCT were invalid and that another three were "unenforceable due to the patent applicants' inequitable conduct in withholding material information from the Patent Office with an intent to deceive the Patent Office."

---

[64] von Hippel; other open source theorists

[65] According to the UC Berkeley web site, Microsoft and other firms were selling the technology by the time the patent issued.

[66] Specifically, we gleaned our cases from four sources: 1) searches of the USPQ database for the university was a party to the litigation; 2) searches of the USPTO Litalert database for cases where a university was a litigation party; 3) searches of general news databases for discussions of university-related patent litigation; 4) running the patent numbers for all university software patents against the LitAlert database. When we found cases of interest, we used PACER docket records to supplement our inquiry. In general, our discussion of cases is unlikely to be underinclusive. The USPQ database is underinclusive to the extent that it contains information only about cases from which a written order has emerged. Although the LitAlert database is supposed to include all cases involving a patent, it actually includes a little over half of such cases. So our collection of cases is probably underinclusive.

[67] To make this determination, we ran the patent numbers for all university patents in the years we sampled (1982, 1987, 1992, 1997, 2002) against the LitAlert database.

[68] http://www.rctech.com/licensing/lic-blue-noise-mask.php (visited June 4, 2006)

Additionally, MIT and its exclusive licensee Akamai were recently involved in lawsuits against two firms, Speedera Networks Inc. and C&W Wireless Internet Services, that allegedly infringed the MIT/Akamai patent on software for decreasing congestion and delay in accessing web pages on the Internet.[69] The Akamai technology, which was launched in 1999, at the height of the "dot-com" boom was similar to technologies developed by other firms before Akamai. Indeed, the defendant in one of the cases, C&W Wireless Internet Services, Inc., used software that differed from Akamai's only in its location.[70] The patent was found valid and infringed by the district court, leading to the grant of a permanent injunction. On appeal, however, the Court of Appeals for the Federal Circuit determined that the broadest claims of the patent were anticipated by the C&W software.[71] The lawsuit against Speedera also involved claims, denied by Speedera, that its chief technical officer was stealing proprietary marketing information. According to industry observers, the litigation left "outsiders shaking their heads at legal battles that seem to be accomplishing little for anybody, but are draining the resources of the companies involved."[72]

In other cases, the university patent has been asserted for large damage awards (usually for the statutory maximum of six years of past infringement) soon before it is about to expire. In one much-discussed lawsuit, MIT and an exclusive licensee, Electronics for Imaging, sued 92 firms, including Microsoft and IBM, alleging infringement of a patent covering image editing software. This lawsuit was filed in December 2001, six months before the patent was set to expire. The technology in question is a color imaging method that can be applied to any system that produces color pictures.[73] Similarly, in March and May 2005, a few months before the relevant patent was due to expire, the University of Texas filed three lawsuits against a total of 42 electronics manufacturers alleging infringement on a patent on software that allows text messaging through a standard telephone keypad. In some cases, the patent in question may not be due to expire immediately, but it is nonetheless relatively old. In 2002, Cornell sued Hewlett-Packard over a 1989 patent obtained by a Cornell professor for a technique that accelerates a computer's processing speed.[74] A 2001 suit by MIT against Lockheed-Martin involved a 1989 patent on systems for analyzing acoustic waveforms.[75]

In all of these cases, commercialization by firms other than the university licensee was going forward, i.e., it does not appear that patent rights or exclusive licenses were

---

[69] See Akamai Technologies, Inc. and MIT v. C&W Internet Services, 344 F.3d 1186 (Fed. Cir. 2003) (discussing patent and reversing in part district court's permanent injunction, based on finding that patent was valid and infringed).

[70] Id. at 1193 (quoting Akamai brief noting this point).

[71] Even the narrower claims were found valid only because the Federal Circuit required C&W to establish that there be a "suggestion or motivation to combine" prior art references. Many scholars have taken issue with the Federal Circuit's "suggestion" test. See briefs in KSR v. Teleflex.

[72] John Borland, *Once-Hot Net Business Crippled by Feuds*, CNET News.com, August 26, 2002

[73] In April 2002, MIT and EFI expanded their complaint to include 214 defendants. In the course of litigation, plaintiffs settled with some defendants and dismissed their claims against others, so that only four remained: Corel, Microsoft, Roxio, and MGI software. MIT v. Abacus et al., __ F.3d __ (Fed. Cir. 2006). As discussed further below, the Federal Circuit recently overturned a district court claim construction that favored the defendants in this case. *Id.*

[74] *See* Cornell Univ. v. Hewlett-Packard Co., 313 F.Supp.2d 114 (2004) (claim construction)

[75] Massachusetts Institute of Technology v. Lockheed Martin et al., 251 F.Supp.2d 1006 (D.Ma. 2003) (granting defendant's motion for summary judgment on noninfringement).

necessary to facilitate "technology transfer." Moreover, there is no evidence in these cases that the other firms' development efforts were "free-riding" on licensees' investments. Contrary to the spirit of Bayh-Dole, in these cases patents allowed universities to extract rents, and perhaps even to "hold up" development efforts.[76]

Even if patenting and exclusive licensing of software does not facilitate commercialization *per se*, it could be argued that it provides a spur to small firm start-ups that market technology inputs.[77] Strong property rights arguably benefit small firm start-ups by serving as a defense against misappropriation when start-ups market their technology.[78] With these strong rights, start-ups may also be able to customize technology inputs for the firms that absorb these inputs. More generally, to the extent that market-based arrangements are likely to be more innovative than large, vertically integrated firms (or at least disseminate information more widely than do such firms), promoting these start-ups could be a valuable goal.

The difficulty with vertical "dis-integration" is that creates the potential for large transaction costs, including hold-up. Moreover, even assuming that the general argument for vertical dis-integration has merit, the force with which it applies to software is not clear. While most small biotechnology firms have patents, a recent study by Mann indicates that most software start-ups that receive venture financing do not in fact have patents.[79] Moreover, although a subsequent study by Mann and Sager suggests that start-up software firms with patents tend to do better on various metrics of financing and investment than start-up software firms without such patents, the disparity is much less significant than the disparity between biotechnology firms with and without patents.[80] At a minimum, the "generating start-ups" argument for software is less compelling than it is for biotechnology.

Rent extraction from firms that have commercialized successfully may be a particular concern where the case is ultimately a weak one. Indeed, where the patent in question is weak, even *ex ante* licensing arrangements are arguably socially inefficient.[81] In other words, in the case of weak patents, even if the university and its exclusive licensee had attempted to license *ex ante*, such licensing would arguably be socially inefficient. Thus it is notable that in a number of litigated cases, the university's argument has been unequivocally rejected. The University of Texas recently lost a case

---

[76] The issue of hold-up of a downstream innovator with sunk costs has been much discussed in the innovation literature. *See*, *e.g.*, __.

[77] Arora and Merges, Specialized Supply Firms, Property Rights, and Firm Boundaries, 13(2) Industrial an Corporate Change 451 (2004); *see also* ASHISH ARORA ET AL., MARKETS FOR TECHNOLOGY. Although Arora and Merges do not discuss university-generated research, the logic of their argument – that strong patents may have the beneficial effect of encouraging the formation of specialized supply firms – applies to such research. Additionally, various scholars have argued that patents can serve as signals that enable small firms to attract venture capital. On this view, at least certain patents are a signal of managerial competence. Lemley (2001) Long (2002); Mann (2005). In any event, it is clear that start-ups often take exclusive licenses to patents. According to AUTM, about 95% of licenses to start-ups are exclusive.

[78] Id.

[79] According to Mann (2005), 80% of software start-ups that received venture financing in 1998-99 did not have patents as of 2003.

[80] Mann and Sager, at 21-22.

[81] Lemley and Shapiro (2006) provide a detailed argument for this proposition. They show that *ex ante* licensing may yield inefficient royalty rates where the patent is weak but injunctive relief is the likely remedy for infringement and "inventing around" is difficult.

involving patents on positron emission technology (PET), a medical imaging technology that uses gamma rays to detect cancer, heart disease, brain disorders, and other health conditions. UT claimed that CTI Molecular Imaging Inc., a leading provider of positron emission tomography (PET) equipment infringed two of its patents on PET technology. In that case, both the district court and the Federal Circuit held that the defendant did not infringe.[82] Similarly, in a recent suit brought by the University of California and its exclusive licensee over patented software for eliminating edge artifacts when compressing digital images, both the district court and the Federal Circuit found that the patent in question was not only invalid but also not infringed.[83] In 2003, a lawsuit by the University of Illinois against Fujitsu on software patents relating to plasma display panels resulted in a summary judgment determination that the patent was invalid.[84] In contrast, our search found only one case in which a university's claims regarded its patented software were largely vindicated by the court system, either in a final district court decision that was not appealed or in an appellate court decision.[85]

Of course, each of these cases was the subject of litigation, and unlikely to be representative of all university software patents. There are certainly cases of successful commercialization where the firm in question did have an exclusive license to a university patent. The firm Google, which was the exclusive licensee of a patent assigned to Stanford, is a prominent example, as is RSA Security, the exclusive licensee of various data encryption patents assigned to MIT.[86] Moreover, university software patents do not appear to be litigated significantly different from those of university patents in other fields. Thus we can not state unequivocally that the incidence of hold-up in the software patent context is higher than that for university patents in other fields.[87] Nonetheless, there is reason to believe that hold-ups are more likely to be common in software than other fields, mainly because, as discussed above, patents/exclusive licenses are less likely to be important for commercialization in this field than others. To put the point another way, the ratio of false positives (patenting and giving an exclusive license when it is not necessary for commercialization) to false negatives (failing to patent and give an exclusive license when it is necessary for commercialization) is likely to be higher in software than in the life sciences. In this regard, it bears mention that various important cases of software commercialization have not involved patents and exclusive licenses. For example, a number of unpatented Stanford programs have been widely adopted by industry. Both MINOS, a linear and nonlinear optimization program, and Genscan, a gene structure prediction program, have been licensed for many years to

[82] 164 Fed. App. 982 (2005). There was no need to rule on the defendant's invalidity challenges.
[83] LizardTech. Inc. v. Earth Resource Mapping Inc., 424 F.3d 1336 (Fed. Cir. 2000).
[84] Competitive Technologies v. Fujitsi, 286 F.Supp.2d 1161 (N.D.Cal. 2003).
[85] MIT v. Abacus (Fed. Cir. 2006). There were a number of cases in which the parties appear to have settled prior to a final, appealable order by the district court.
[86] Whether a patent and exclusive license was particularly important to commercial success in these cases is unclear, however.
[87] A recent example of hold-up in the life sciences arena might be the case brought by Ariad Pharamaceuticals, the exclusive licensee of a broad Harvard/MIT patent, against Eli Lilly. Ariad Pharmaceuticals recently secured a large jury verdict in that case. Cite

dozens of different commercial firms.[88] Stanford has licensed other unpatented software to firms that have independently developed the software for sale to their own customers.[89]

Finally, to the extent the university interest in revenue generation is seen as legitimate---presumably because some of the revenue goes back into research---the alternative of *ex ante* non-exclusive licensing at low rates might be seen as a mechanism for satisfying university interests while minimizing potential harms. Although non-exclusive licenses are something of a tax on commercialization, a small royalty rate is unlikely to deter most commercialization. In some cases, universities may be reluctant to license non-exclusively because they must incur the immediate cost of filing for a patent. (In contrast, exclusive licensees typically pay for patent applications.) Notably, however, software is unlike virtually every other university invention because patents do not have to serve as the foundation of the licensing scheme. Copyright, which attaches without cost upon creation of the software, will do the job. For this reason, at least some technology transfer offices say that they are beginning to shy away from seeking patents for the purpose of non-exclusive licensing. At the University of Washington, for example, the Digital Ventures office says that it takes a "hard look" at patenting and will use it only if there is a real need to improve the technology (presumably via an exclusive license). Digital Ventures has also convinced startups to "go without a patent."[90] Non-exclusive copyright licensing can be quite lucrative for universities. MINOS, which is available via a non-exclusive copyright license, is one of Stanford's largest money generators.[91] Moreover, as noted earlier, in FY 2005, the UW Digital Ventures office made 90% of its revenue from copyright licensing.

Because copyright has been interpreted by the courts to cover little more than literal source code, it generally confers little in the way of monopoly power. Moreover, in copyright law, unlike patent law, independent creation is a defense to a charge of infringement. For this reason, commentators concerned about the negative effects of proprietary rights have properly focused on software patents. At the same time, firms are willing to license software because they do not want to bear the costs of independent creation. However, even nonexclusive copyright licensing with relatively low fees can be problematic for non-profit researchers. Thus, universities that want to balance the goal of academic access with that of revenue generation, such as the University of Washington, are assessing which licenses and royalty structures are appropriate for which situations. The TTOs at UW, the University of California, Berkeley, and Stanford have all embraced the idea of forked licenses that give relatively inexpensive access to the non-profit sector but allow for revenue generation from the commercial sector. Indeed, one of Stanford's largest revenue generators in software, the MINOS business software program is available via a forked copyright license: the commercial fork costs more than ten times as much as the academic fork.

The related question of using copyright to promote "open source" within the university is an interesting one. As we have noted, some universities have embraced open source, and these universities tend to have smaller number of pure software patents. But even among technology transfer officials sympathetic to the goals of open source, a

---

[88] Software Licensing in the University Environment, Computing Research News (January 2002).
[89] Id.
[90] E-mail communication from Chuck Williams, September 2, 2005
[91] Interview with Kathy Ku

number mention difficulties that open source may create in the university setting. For example, faculty may prefer open source as a method of distribution not because of ideological commitment but because open source-related consulting revenues, unlike licensing royalties, don't have to be shared with the university.[92] Indeed, at least one prominent technology transfer officer (who preferred to remain anonymous) believes that some faculty make software open source for the purpose of attracting widespread interest but have every intention of asserting proprietary rights over the source code at some later point.[93] Additionally, software is often developed by groups, and TTOs sometimes find themselves in the middle of disputes among group members about the best open source mechanism to use (or, indeed, whether open source should be used at all).[94] TTOs are also wary that particular types of open source licenses will conflict with obligations to sponsors, including the federal government under Bayh-Dole.[95] Thus, to the extent funding agencies are interested in an open source approach because they think such an approach is likely to produce better software, they need to be aware of possible institutional impediments.

Conclusion

Our research indicates that software patents represent a significant percentage of university patent holdings. Moreover, at least historically (that is, through the late 1990s), software patenting appears to have been linked most closely not with R&D spending, or even R&D spending in computer science, but with the university's patent propensity. From a policy standpoint, this finding suggests a potential problem: because software – and particularly pure software – is likely to follow a different commercialization path than invention in the life sciences, patenting and exclusive licensing of software may yield a higher proportion of situations where the exclusive licensee attempts to hold up an entity that has successfully commercialized without the need for an exclusive license. Moreover, even if the goal is not promoting commercialization *per se*, but promoting start-ups, exclusive patent licenses are not necessarily critical to that goal.

To be sure, university practices with respect to software are not uniform. For example, universities that have policies friendly to open source are less likely to patent pure software. More generally, a fair number of universities – including universities with large number of software patents – may now be moving away from patenting and

---

[92] Pat Jones

[93] This technology transfer officer did not specify precisely how a faculty member would assert proprietary rights. In the context of a viral license, one mechanism for doing so would be to "fork" the license. One prong of the license would remain viral while the other, which was made available to paying customers, would not be viral in character. MySQL has adopted this strategy. See http://www.mysql.com/company/legal/licensing/commercial-license.html

[94] Chuck Williams; Dana Bostrom; Lita Nelsen

[95] In theory, under Bayh-Dole, if the university and researcher choose not to patent, the government has the option of patenting. Whether a decision to release software under an open source license represents an unwarranted interference with the government's option remains an open question, at least in theory. In practice, however, we are unaware of any situation where a decision to release software under an open source license has interfered with an agency's desire to patent.

exclusive licensing of software.[96] However, the lingering impact of a substantial university patent portfolio, some of which has been licensed exclusively, is being felt in socially unproductive litigation.

[96] To investigate this proposition further, we plan to do further work investigating more recent patent applications that have resulted in issued patents.

Figure 1: University Software Patents, By Type and Year

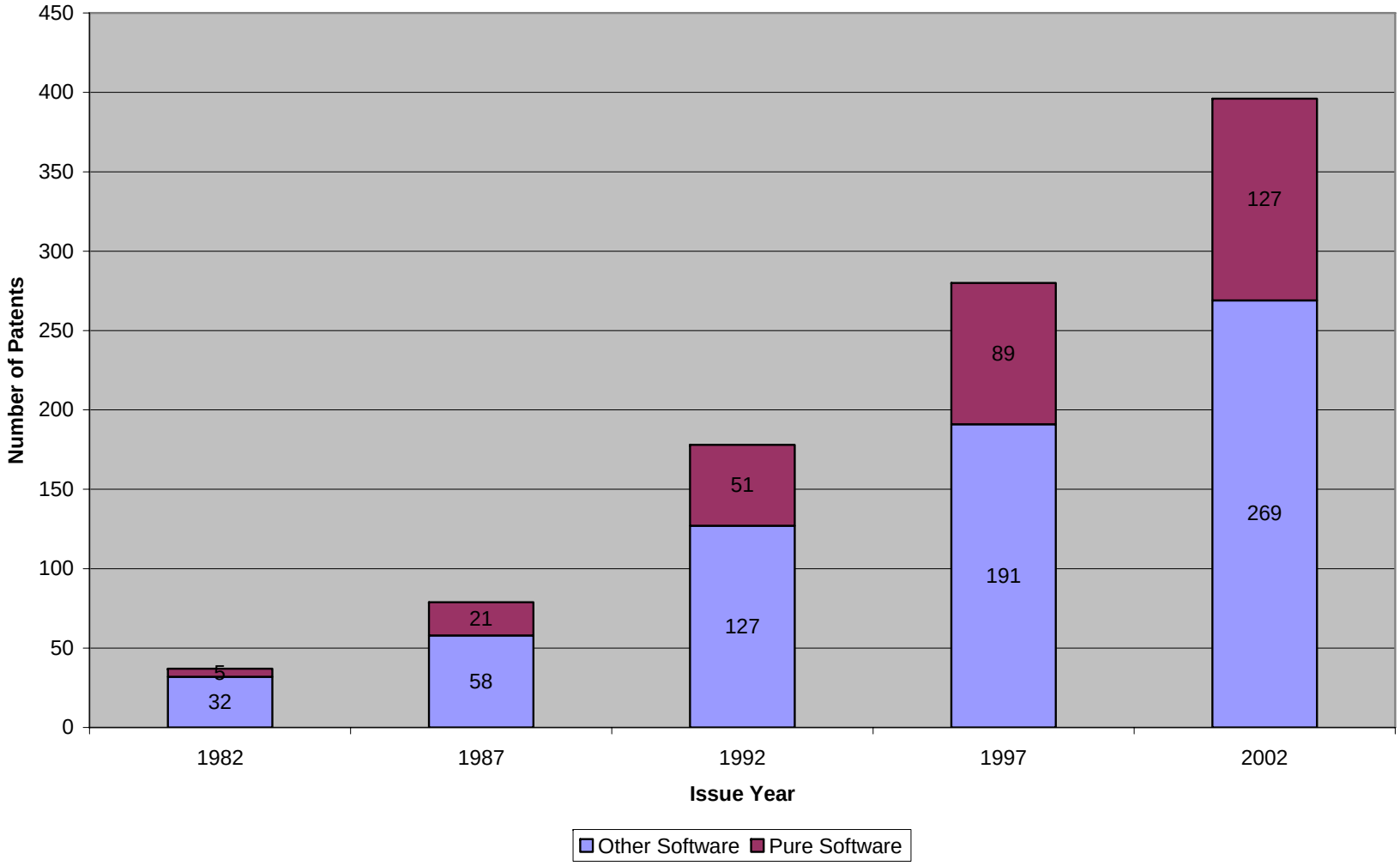

Figure 2: University Software Patents as a Share of All University Patents

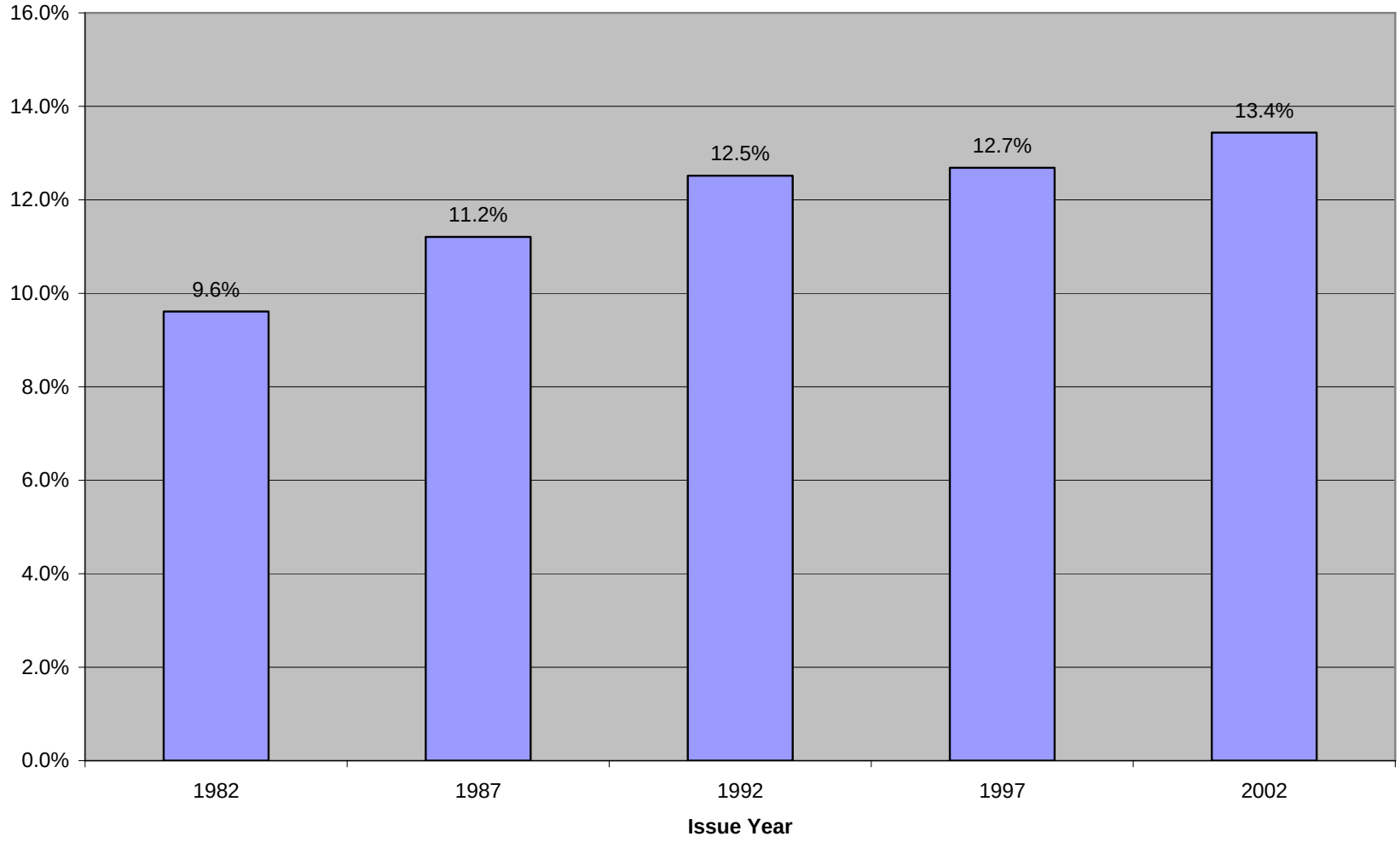

Table 1: Comparison of Bessen-Hunt Keyword-Based Classification of Software Patents With Our Classification

	Our Classification		
Bessen-Hunt Classification	Non SW	SW	Total
Non SW	2,325	202	2,527
	92.01	7.99	100
	91.32	51.01	85.89
SW	221	194	415
	53.25	46.75	100
	8.68	48.99	14.11
Total	2,546	396	2,942
	86.54	13.46	100
	100	100	100

Table 2: Comparison of Graham-Mowery IPC-Based Classification of Software Patents With Our Classification

	Our Classification		
Graham-Mowery IPC Classification	Non SW	SW	Total
Non SW	2,544	341	2,885
	88.18	11.82	100
	99.92	86.11	98.06
SW	2	55	57
	3.51	96.49	100
	0.08	13.89	1.94
Total	2,546	396	2,942
	86.54	13.46	100
	100	100	100

Table 3: Comparison of Graham-Mowery USPC-Based Classification of Software Patents With Our Classification

	Our Classification		
Graham-Mowery USPC Classification	Non SW	SW	Total
Non SW	2,541	328	2,869
	88.57	11.43	100
	99.8	82.83	97.52
SW	5	68	73
	6.85	93.15	100
	0.2	17.17	2.48
Total	2,546	396	2,942
	86.54	13.46	100
	100	100	100

Table 4: University Software Patenting, Overall Patenting, And Pure Software Patenting In 2002

	Rank in Patenting (Issue Year 2002)		
University	**Software**	**Overall**	**Pure Software**
Massachusetts Institute of Technology	1	2	1
University of California	2	1	2
Stanford University	3	4	4
California Institute of Technology	4	3	23
University of Texas	5	5	41
University of Washington	6	15	6
University of Wisconsin	7	7	8
Georgia Institute of Technology	8	20	3
Carnegie Mellon University	9	51	13
Johns Hopkins University	10	6	11
State University of New York	11	8	20
University of Rochester	12	50	14
University of Pennsylvania	13	13	42
University of Illinois	14	28	5
Columbia University in the City of New York	15	14	10

Table 5: Negative Binomial Models of Determinants of Software Patents (Issue Year 2002)

	SW	SW	Pure SW	Pure SW
	(1)	(2)	(3)	(4)
ln(lagged CS R&D)	.110**	.097**	.588***	.523***
	(.049)	(.045)	(.121)	(.110)
ln(lagged Other R&D)	.839***	.073	.223	-.395**
	(.118)	(.141)	(.161)	(.163)
ln(Non Software Patents)		.910***		.781***
		(.141)		(.167)
Const.	-11.577***	-3.613**	-9.017***	-2.182
	(1.406)	(1.510)	(1.615)	(1.576)
Obs.	202	202	202	202

Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 6: Negative Binomial Models of Determinants of Software Patents (Issue Year 1997)

	SW	SW	Pure SW	Pure SW
	(1)	(2)	(3)	(4)
ln(lagged CS R&D)	.245***	.172**	.348***	.275**
	(.076)	(.072)	(.134)	(.131)
ln(lagged Other R&D)	.986***	.308	1.005***	.271
	(.160)	(.196)	(.252)	(.320)
ln(Non Software Patents)		.832***		.813***
		(.185)		(.282)
Const.	-14.941***	-7.394***	-17.358***	-9.023**
	(1.878)	(2.179)	(2.975)	(3.587)
Obs.	202	202	202	202

Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 7: Negative Binomial Models of Determinants of Software Patents (Issue Year 1992)

	SW	SW	Pure SW	Pure SW
	(1)	(2)	(3)	(4)
ln(lagged CS R&D)	.143*	.082	.122	.071
	(.077)	(.073)	(.110)	(.107)
ln(lagged Other R&D)	1.083***	.348*	.968***	.264
	(.192)	(.206)	(.287)	(.323)
ln(Non Software Patents)		1.027***		.955***
		(.199)		(.318)
Const.	-15.295***	-7.610***	-14.735***	-7.392**
	(2.213)	(2.221)	(3.357)	(3.446)
Obs.	202	202	202	202

Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 8: Negative Binomial Models of Determinants of Software Patents (Issue Year 1987)

	SW	SW	Pure SW	Pure SW
	(1)	(2)	(3)	(4)
ln(lagged CS R&D)	.343***	.314***	.289	.210
	(.122)	(.092)	(.194)	(.162)
ln(lagged Other R&D)	1.013***	.291	.864**	.094
	(.236)	(.198)	(.351)	(.319)
ln(Non Software Patents)		.797***		1.035***
		(.163)		(.303)
Const.	-16.411***	-8.430***	-15.243***	-6.796**
	(2.592)	(2.117)	(3.792)	(3.261)
Obs.	202	202	202	202

Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 9: Negative Binomial Models of Determinants of Software Patents (Issue Year 1982)

	SW	SW	Pure SW	Pure SW
	(1)	(2)	(3)	(4)
ln(lagged CS R&D)	.218*	.167	.770**	.748**
	(.128)	(.124)	(.336)	(.358)
ln(lagged Other R&D)	.612**	-.043	1.435**	.709
	(.270)	(.193)	(.635)	(.774)
ln(Non Software Patents)		1.326***		.742
		(.272)		(.501)
Const.	-10.317***	-4.045**	-27.910***	-19.811**
	(2.824)	(1.798)	(8.697)	(10.044)
Obs.	202	202	202	202

Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 10: Negative Binomial Panel Models of Determinants of Software Patents, Issue Years 1982-2002

	SW	Pure SW
	(1)	(2)
ln(lagged CS R&D)	.037	.107
	(.046)	(.096)
ln(lagged Other R&D)	-.344	-.119
	(.277)	(.628)
ln(Non Software Patents)	.464***	.054
	(.090)	(.167)
1987 Dummy	.665***	1.363**
	(.237)	(.583)
1992 Dummy	1.354***	2.193***
	(.311)	(.754)
1997 Dummy	1.714***	2.708***
	(.379)	(.897)
2002 Dummy	2.008***	3.045***
	(.442)	(1.040)
Const.	-9.405	-15.118
	(87.124)	(418.802)
Obs.	1010	1010

All models include university fixed effects. The left–out year category is 1982. Robust standard errors in parentheses. *** denotes p<.001, ** denotes p<.01, and * denotes p<.05.

Table 10: Departmental Origin of Inventors on Software Patents Issued to the Top 15 Recipient of Software Patents in Issue Year 2002

Table 11: Departmental Origin of Inventors on Pure Software Patents Issued to the Top 15 Recipient of Software Patents in Issue Year 2002