AN INQUIRY INTO COMPUTER SYSTEM PATENTS: BREAKING DOWN THE "SOFTWARE ENGINEER"*

*SCOTT ELENGOLD***

INTRODUCTION

In 2002, IBM received 3,288 patents, yielding a total of more than 22,000 patents acquired over the previous ten years, the highest number received by any corporation.[1] IBM is one of the many companies taking advantage of the rapid proliferation of computer system patents that will have a large impact on the future of the technology industry and play a major role in protecting important assets in the United States economy. In fact, computer systems have grown to be such a large component of the United States economy that they accounted for approximately one quarter of the overall increase in Gross Domestic Product during the 1990s.[2] The success of these computer systems often depends upon their innovative step;[3] software companies are always struggling to offer newer or better software than their competitors. Protecting these innovations in order to spur innovative growth in the art has only recently

* This Note received the 2004–2005 Seymour A. Levy Memorial Award, given each year to the graduating student who has written the most outstanding Note for the *NYU Annual Survey of American Law*.

** Note Editor, *NYU Annual Survey of American Law* 2004–05; J.D., New York University School of Law, 2005. I would like to thank the editors and staff of the *NYU Annual Survey of American Law* for all of their hard work and helpful comments. It was a privilege to work with Jessie Beller, who provided invaluable editorial assistance in preparing this Note for publication. I would also like to thank Professors Rochelle Dreyfuss and Diane Zimmerman for providing many useful suggestions on an earlier version of this Note and Professor Katrina Wyman for sparking my desire to contribute to legal scholarship. This Note is dedicated to Kate Sablosky, Linda and Mark Elengold, Harold Anfang, and Steven Pokotilow for their guidance, advice, and support throughout my law school career.

1. Lisa M. Bowman, *.Net patent could stifle standards effort*, CNET News.com, *at* http://news.com.com/2100-1001-984052.html (Feb. 10, 2003).

2. Wikipedia, *Software Engineering - Wikipedia, the free encyclopedia*, *at* http://en.wikipedia.org/wiki/Software_engineering (last visited Mar. 24, 2005).

3. I will use the term "innovative step" to refer to that aspect, contained within the larger system, which is novel and non-obvious.

become a reality in our patent system, and already an estimated eighty thousand computer system patents have been issued.[4]

The goal of this Note is to call for a re-examination of how the patent system treats computer system patents.[5] The courts have often used the fiction of an all-knowing "software engineer" as one of ordinary skill in the art which leads to an improper assessment of the contributions made within the field of computer science. A dissection of the "software engineer" is necessary in order to ensure that the patent system does not reward patent protection where it is unwarranted while denying patents to innovative advancements within the more specific areas of the art.

I will assume for purposes of this Note that the patent system will remain the forum of choice for software protection in the coming years. Other academics have suggested a *sui generis* intellectual property scheme for computer software[6] and there are definite advantages to such a proposal, but at this point such a *sui generis* system has not been instituted nor has such a program been formally proposed. With the Federal Circuit endorsing patentability without qualification, the question of whether software is or should be patentable subject matter is "for the history books."[7] As this subject has been given extensive treatment in the academic literature, I will limit discussion of the merits of expanding the use of the patent system and instead focus on how computer systems should be analyzed within the existing doctrines and patent requirements.

Part I briefly explores the history of software protection. The gradual approval of computer systems as patentable subject matter has played a large role in how the art is perceived within the patent system and helps explain the current state of the patent law as applied to software patents. Computer systems are only viewed "as a whole" when being assessed for patentability, a result of this early precedent. Part II examines an alternative, and I feel more accurate, view of the practical requirements and steps for modern software development. I briefly explain the current standard process for the art of software development—the Rational Unified Pro-

4. Julie E. Cohen & Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 CAL. L. REV. 3, 4 (2001).

5. Computer system patents contain processes that may be implemented in either hardware or software solutions. The patent system has often treated the implementation method as irrelevant to inquiries of patent validity, so I will use the terms "computer system patents" and "software patents" interchangeably.

6. *See, e.g.*, Pamela Samuelson et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, 94 COLUM. L. REV. 2308 (1994).

7. Cohen & Lemley, *supra* note 4, at 4. R

cess;[8] and how the widespread adoption of this production model should alter the current view within the patent system of the all-knowing "software engineer." I propose breaking down the term "software engineer" into four roles: graphic designer, software designer, software developer, and application architect.

Part III reviews how the patent system currently treats software development. I focus on Federal Circuit opinions that deal with the prevalent standard of "one of ordinary skill in the art" and particularly how this standard has affected determinations of non-obviousness and disclosure requirements for computer system patents. In addition, I briefly examine the patent examiner guidelines for reviewing computer system patents. Part IV re-examines the patent requirements and the use of "one of ordinary skill in the art" under the practices of modern software development to identify issues and propose the adoption of this modern view within the patent system.

By the conclusion, I aim to show that by grouping all of the roles involved in software development under the term "software engineer," the patent system has not adjusted to the actual functioning of the art of computer system development. This raises issues for determining the patentable aspects of computer systems and how the patent requirements should be applied to these inventions. Going forward, the industry would be better served by having the courts take a meaningful look at the scope of the art for each innovation and the true skills possessed by those involved in the innovative step when dealing with software patents.

I.
THE HISTORY OF COMPUTER SYSTEMS AS PATENTABLE SUBJECT MATTER

The complicated history of software patentability has resulted from the application of basic patent requirements to a constantly changing art. The patent system has completely reversed course in the last two decades after initially denying the availability of patent

8. For examples of how the Unified Process has come to dominate software engineering, see http://www.enterpriseunifiedprocess.info/ (identifying the Unified Process as the de facto standard development process) (last updated Oct. 26, 2004); http://www.ronin-intl.com/training/rup.html (specifying the Rational Unified Process as the standard development process for mission critical systems) (last visited Mar. 24, 2005); Scott Ambler, *Strategic Reuse Management and the Rational Unified Process (RUP)*, *available at* http://www.flashline.com/content/Ambler/reuseRUP.jsp (last visited Mar. 24, 2005) (stating that "the RUP is a very good, rigorous software process, arguably the de facto standard within the IT industry").

protection to the field. As can often be expected from the legal system, the shift towards patentability has occurred mostly by drawing distinctions from earlier cases rather than proclaiming an overwhelming acceptance of protection for the new art. Thus, the current acceptance of computer system patents is tempered by a variety of precedents that have shaped the perception of the art within the courts. The Supreme Court and the Court of Appeals for the Federal Circuit have led other courts in development of key precedent.

The first case to directly address the patentability of software went before the Supreme Court in 1972. In *Gottschalk v. Benson*,[9] the patent at issue described an invention "related to the processing of data by program and more particularly to the programmed conversion of numerical information in general-purpose digital computers."[10] The claims at issue in the case were rejected by the Patent Office for failing to be a "process" as defined and used in sections 100 and 101 of the Patent Act.[11] The patent broadly claimed a method of converting binary-coded decimal ("BCD") number representations into binary form, and the court conceded that "one may not patent an idea. But in practical effect that would be the result if the formula for converting BCD numerals to pure binary numerals were patented in this case."[12] Leaving the issue of whether to extend patent protection to algorithms for use in a digital computer to Congress, the Supreme Court created a "mathematical algorithm" exception to the term "process" as defined in section 100. This early decision appeared to preclude patents for software, even when embodied within a functioning computer system. Thus, the initial focus for the patentability of software was on the algorithm as a whole.

The Patent Office and the Court of Patent Appeals struggled with applying this exception to the variety of computer patents being filed, and in 1978 the case of *Parker v. Flook*[13] was heard by the Supreme Court. The patent at issue was for a method of utilizing a computer to continuously recalculate an alarm limit during a chemical conversion process. In *Flook*, the Supreme Court found that the

9. 409 U.S. 63 (1972).

10. *Id.* at 64 (internal quotations omitted).

11. 35 U.S.C. § 100(b) ("The term 'process' means process, art or method, and includes a new use of a known process, machine, manufacture, composition of matter, or material."); 35 U.S.C. § 101 ("Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor. . . .").

12. *Benson*, 409 U.S. at 71.

13. 437 U.S. 584 (1978).

only novel part of the invention was the use of the computer software to implement a pre-existing process and reaffirmed the holding in *Benson* that the "discovery of a novel and useful mathematical formula may not be patented."[14] In *Flook*, however, there appeared to have been an inventive step; secondary considerations such as the success of the invention lent strong credibility to the claims that the invention was an advancement in the art. Again, however, the Supreme Court invalidated the patent as not falling within the statutory subject matter of section 101 by focusing on the use of a known algorithm rather than its novel, computerized implementation.[15]

The reversal in favor of software patentability began three years later in *Diamond v. Diehr,*[16] where a process for molding uncured synthetic rubber into shaped, cured products was held to be patentable subject matter. Diehr claimed that while the Arrhenius equation[17] for time, temperature, and cure relationships could yield accurate curing results, the industry had not been able to accurately measure the temperature inside the mold throughout the process to achieve ideal results.[18] Diehr's contribution to the art was to constantly feed the temperature measurements from inside the press into a computer which was programmed to continuously calculate the cure time with the Arrhenius equation and signal a device to open the press when the curing process was complete.[19]

The hard question before the Court was whether this case was truly different from *Flook*. The Court distinguished the case from *Flook* by stating that:

> In contrast, the respondents here do not seek to patent a mathematical formula. Instead, they seek patent protection for a process of curing synthetic rubber. Their process admittedly employs a well-known mathematical equation, but they do not seek to pre-empt the use of that equation. Rather, they seek only to foreclose from others the use of that equation in conjunction with all of the other steps in their claimed process.[20]

The Court focused on the claim as a whole to find an innovative step beyond the use of the Arrhenius equation, but a tenuous

14. *Id.* at 594.

15. *Id.*

16. 450 U.S. 175 (1981).

17. The equation is named after its discoverer Svante Arrhenius and had long been used to calculate the cure time in rubber-molding presses. *Id.* at 177 n.2.

18. *Id.* at 177–78.

19. *Id.* at 178–79.

20. *Id.* at 187.

distinction between simply a patent for an encoded algorithm and a patent for an encoded algorithm used with additional steps in the process remained. Regardless, the decision in *Diehr* appears to have opened the door for a flood of patents where an entirely new process, or a new process utilizing previously known algorithms, could be implemented through the use of computer processing power. As long as a physical process was included in the claims, even if well known in the art, a patent could be issued to a novel computer algorithm; the patentability of computer systems was tied to the process in which they were used rather than the innovations contained within.

This trend continued until 1994 when the Federal Circuit decided *In re Alappat*,[21] a decision which opened the door for an additional class of software patents. Alappat's invention was a means for smoothing out the wave form display in a digital oscilloscope; a claim construed by the Federal Circuit to cover a machine composed of the logical circuits listed by Alappat to perform his smoothing technique.[22] The court acknowledged the mathematical algorithm exception to section 101 and directly stated that the claimed subject matter in the case did not fall within the exception.[23]

The court limited the *Diehr*, *Flook*, and *Benson* line of cases as applying the exception "that certain types of mathematical subject matter, standing alone, represent nothing more than *abstract ideas* until reduced to some type of practical application, and thus that subject matter is not, in and of itself, entitled to patent protection."[24] But the court looked further to how the application of software to a general purpose computer creates a patentable machine.[25] The court said that "such programming creates a new machine, because a general purpose computer in effect becomes a special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software."[26]

The *Alappat* opinion made it clear that software claims must be tied to a machine in order to pass the section 101 subject matter test. The search for patentable subject matter within computer systems was shifted to focusing on the software itself as a novel and useful process, which could be executed by a computer. After *Alap-*

21. 33 F.3d 1526 (Fed. Cir. 1994).
22. *Id.* at 1541.
23. *Id.* at 1542.
24. *Id.* at 1543 (footnote omitted).
25. *Id.* at 1544–45.
26. *Id.* at 1545.

pat, the only remaining major obstacle to patenting computer programs was the requirement that software claims implement the program within an apparatus or machine. This obstacle fell in 1995 when IBM appealed a United States Patent and Trademark Office rejection of a claim to the Federal Circuit.[27] On appeal, the Commissioner of Patents and Trademarks conceded that "computer programs embodied in a tangible medium, such as floppy diskettes, are patentable subject matter," and the case was dismissed for lack of case or controversy.[28]

In *State Street Bank & Trust Co. v. Signature Financial Group, Inc.*,[29] the Federal Circuit addressed business systems or so-called "business method" patents and eliminated an additional hindrance to computer system patents. Signature was the assignee of a patent for implementing an investment structure for pooling assets of mutual funds in a data processing system.[30] State Street brought a declaratory judgment action against Signature alleging invalidity, and the appeal concerned whether the patent was invalid for failing to claim statutory subject matter under section 101. The court reiterated that a mathematical algorithm is only unpatentable when not useful; when transformed by a machine to a "useful, concrete and tangible result," the abstract idea becomes a patentable machine or process.[31] The court also eliminated any conception of a business method exception by describing the exception as "an unwarranted encumbrance to the definition of statutory subject matter in section 101, that [should] be discarded as error-prone, redundant, and obsolete."[32]

Computer systems are now well established as containing patentable subject matter, and the number of patents issued for these innovations continues to rise. As a consequence of this rapid shift to acceptance within the legal community, the focus of the patent system has remained on the statutory subject matter requirements for patentability of the computer systems as a whole, and little attention has been paid to the patentability of innovations that can occur within the larger framework of overall software development. These innovations are analogous to a more powerful engine, a more responsive steering system, or a more sensitive radio antenna

27. *In re* Beauregard, 53 F.3d 1583 (Fed. Cir. 1995).

28. *Id.* at 1584.

29. 149 F.3d 1368 (Fed. Cir. 1998).

30. *Id.* at 1370.

31. *Id.* at 1373.

32. *Id.* at 1375 n.10 (citing *In re* Schrader, 22 F.3d 290, 298 (Fed. Cir. 1994) (Newman, J., dissenting)).

that can all be embodied as components within a larger automobile; similarly important advancements can occur within the overall computer system. These automobile components are evaluated by the patent system as independent contributions, and specialists in the arts of engine design, steering mechanics, and radio technologies are each considered to be one of ordinary skill in their respective art. It is important to likewise analyze the advancements that can occur within the computer system as a whole by breaking down the process of software development into stages and roles and using each of these specialists as one of ordinary skill in his art.

II.
THE MODERN APPROACH TO SOFTWARE DEVELOPMENT

The rapid growth of the software industry has led to such a large number of innovations that individual workers can no longer keep up with the rapid pace of change within the software development process; workers have been forced to specialize within the field in order to maintain the quality of their work product. As a result of increasing competition, software production firms further contributed to worker specialization by placing workers into specific roles to maximize the quality of their software products while taking advantage of the efficiency gain created by worker specialization. Workers and firms alike sought a standard process to yield maximum efficiency in software production and to create a uniform system of roles and responsibilities for employees.

A. The Rational Unified Process

In any industry, there are constant improvements and refinements in the methods of production that can change the whole industry, such as the introduction of assembly lines at Ford Motor Company.[33] The Unified Process has had a similarly sweeping effect on software development over the past five years. The Unified Process is a complete software design and construction strategy that incorporates material in the areas of data engineering, business modeling, project management, and configuration management. The first instance of the Unified Process was the Rational Unified Process ("RUP") and to date it is the most widely accepted version[34]

33. *See* Douglas G. Baird, *In Coase's Footsteps*, 70 U. CHI. L. REV. 23, 31–35 (2003) (chronicling the changes required at GM to reorganize its production to capture the same efficiencies as Ford).

34. For details on the Rational Unified Process, see IBM, *Rational Software*, *at* http://www.ibm.com/rational (last visited Mar. 25, 2005).

and has become the industry standard for object-oriented development.[35]

The Rational Unified Process is a detailed version of a more generic process originally described by Ivar Jacobson, Grady Booch, and James Rumbaugh in the textbook *The Unified Software Development Process*.[36] These three recognized scholars developed the Rational Unified Process in the mid-1990s, and in January of 1999, the publication of *The Unified Software Development Process* released the details of this work to the public.[37] The Rational Unified Process was the first software development tool to use the newly created Unified Modeling Language ("UML"),[38] which allows for an easy explanation of software design and use for both technical and non-technical workers. Centered on the concept of use case[39] and an object-oriented design method,[40] the RUP has rapidly gained recognition in the software industry and has been adopted and integrated by many companies worldwide.[41]

The first phase defined in the RUP is the inception phase.[42] During the inception phase, the business case for the system is established, and the project scope is defined.[43] All external entities with which the system will interact, hereafter referred to as actors, need to be identified, and the nature of the interaction needs to be defined, though only at a high level.[44] Examples of actors include system operators, other computer systems, and end users. In order

35. *See supra* note 8.

36. RATIONAL SOFTWARE CORPORATION, RATIONAL UNIFIED PROCESS: BEST PRACTICES FOR SOFTWARE DEVELOPMENT TEAMS 17 (2001), *available at* http://www.probank.biz/rational.pdf (last visited Mar. 25, 2005).

37. *Id.*

38. UML is the standard language for specifying, visualizing, constructing, and documenting all of the artifacts of a software system. *Id.* at 1.

39. A use case is a sequence of related transactions performed by an actor in an interaction with the system. TERRY QUATRANI, INTRODUCTION TO THE UNIFIED MODELING LANGUAGE 7 (June 11, 2003), *available at* http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/intro_rdn.pdf. A use case differs from a software module in that it purely represents functionality, not any discrete collection of code. Defining the use cases in use case diagrams allows for the developers, supervisors, and clients to be jointly aware of all high-level functionality requirements for the computer system. *Id.* at 7–9.

40. Object-oriented design structures the computer system design around the object that exists within the system instead of only the procedural steps that will need to take place for each interaction.

41. *See supra* note 8.

42. IVAR JACOBSON, GRADY BOOCH, & JAMES RUMBAUGH, THE UNIFIED SOFTWARE DEVELOPMENT PROCESS 8, 341 (1999).

43. *Id.* at 341–42.

44. *Id.* at 345.

to clearly define all of the actors and their roles, all of the system use cases must be identified.

The second phase, elaboration, is essentially what it sounds like; here the problem domains will be worked out, and an architectural foundation will be established.[45] By the conclusion of this phase, the workers will have elaborated upon the use cases in sequence diagrams, collaboration diagrams, and a class diagram. The class diagram is arguably the most important; it lists and defines all of the objects that have been identified within the system. The class diagram then defines the attributes and behaviors for each of the objects, and the objects morph into classes, a group of objects that possess similar qualities.[46]

At the completion of this phase, "the hard 'engineering' is considered complete."[47] Not a single line of code has been written, but the computer system is essentially fully laid out on paper and has been conceptually "created" though there is no working software implementation. The next two phases are the construction phase and the transition phase.[48] The construction phase is the actual implementation or writing of the code to support the computer system, where the bulk of the work in terms of hours will take place, and the transition phase details the release of the software into its intended use.[49]

The RUP illustrates the knowledge required to *design* the "process" of a computer system; this knowledge does not require an ability to *implement* the process within a technical architecture or to write computer code. The RUP also lays out a framework for resource allocation—a collection of roles that workers can be placed into to maximize both the productivity and innovation gains that come from specialization. By having workers gain detailed knowledge over the state of the art in one aspect of software development, the final software product can be a better and more efficiently produced product.

B. Roles in Software Development

The Federal Circuit and the lower courts have continually used the terms "software engineer" and "software programmer" when referring to one of ordinary skill in the art of computer systems.[50]

45. RATIONAL SOFTWARE CORPORATION, *supra* note 36, at 4. R
46. QUATRANI, *supra* note 39, at 12. R
47. RATIONAL SOFTWARE CORPORATION, *supra* note 36, at 4. R
48. *Id.* at 5–7.
49. *Id.*
50. *See infra* Part III.

Instead of grouping all of the software development process participants under this general heading, I have identified four distinct roles in the RUP—the graphic designer, software designer, software developer, and application architect. Breaking down the "software engineer" into these roles allows for a better measure of the contributions made by a patent application within each of these specialized fields.[51]

1. The Graphic Designer

The graphic designer is the worker who designs the user interface for the software. In modern society the interface is often web-enabled, or designed to work inside an Internet browser such as Microsoft's Internet Explorer. For "back-end" systems, systems not accessible to users outside the company building or software installed on the local machine, the interface is often windows-based.[52] While the graphic designer must have knowledge of the capabilities and limitations that the technology being used to build the system possesses, often the worker in this role does not understand the details of the implementation behind the interface. The main job of the graphic designer is to design a user interface that is intuitive and easy to use.

Currently, copyright protection for the aesthetic design is generally viewed as the only intellectual property protection available to the graphic designer. Copyright law, however, does not protect ideas, and when the graphic interface is the only means of expression for the underlying ideas, this "merger" renders the work not copyrightable.[53] "When the 'idea' and its 'expression' are thus inseparable, copying the expression will not be barred, since protecting the expression in such circumstances would confer a monopoly of the idea upon the copyright owner free of the conditions and

51. Some critics may counter that we should impute knowledge from all roles in the RUP to the "person having ordinary skill in the art" because corporations, and not individual inventors, make up the bulk percentage of computer system patent filers. This runs counter to the precedents built into the foundations of our patent system (i.e. the requirement of listing individual inventors on the patent application rather than the employing corporation). For more information on the historical context behind the requirement for joining individual inventors as patent filers rather than the overall corporation see 1 DONALD S. CHISUM, CHISUM ON PATENTS § 2.03 [1]–[2] (2004).

52. An example is Microsoft Word, which runs inside Microsoft Windows.

53. MELVILLE B. NIMMER & DAVID NIMMER, NIMMER ON COPYRIGHT § 2.18 [B][2]–[3] (LexisNexis ed. 2004).

limitations imposed by the patent law."[54] Under the current law, a graphic designer is unable to receive any form of intellectual property protection for the functional benefits achieved by a new software interface design.

Arguably, this is where the patent system should step in. In *Apple Computer, Inc. v. Microsoft Corp.*,[55] the Ninth Circuit addressed the lack of protection for these types of ideas. The court found that the use of windows to display multiple images, the use of menus to store information or functions in a convenient place, and the use of iconic representations of familiar objects were not copyrightable features of an operating system.[56] The court attempted to separate out functional elements from artistic ones, but found that constraints such as the power and speed of the computer could limit the range of possible expressions and hamper copyright protection.[57] The decision left available only the option of protecting the "particular expression" in a graphical user interface and denied intellectual property protection to the truly innovative aspects of Apple's user interface.

Because user interface designs are generally regarded as only aesthetic, it is admittedly difficult to advocate for a widespread use of patent protection. However, when the improvement is a functional, rather than purely expressive, advancement in the art, the patent system is the appropriate place to evaluate whether the improvement merits protection. These improvements are better characterized as ideas than expressions, and patent law is built upon the foundation that patent protection for ideas spurs economic growth and further development.[58] A useful and novel user interface is as much an apparatus or process as a machine executing software instructions; the only major difference is tangibility. With software patents no longer requiring a tangible working machine, user interface patents may be a logical consequence.

54. Herbert Rosenthal Jewelry Corp. v. Kalpakian, 446 F.2d 738, 742 (9th Cir. 1971).

55. 35 F.3d 1435 (9th Cir. 1994).

56. *Id.* at 1443–44.

57. *Id.* at 1444–45.

58. *See, e.g.*, Mazer v. Stein, 347 U.S. 201, 219 (1954) ("The economic philosophy behind the clause empowering Congress to grant patents and copyrights is the conviction that encouragement of individual effort by personal gain is the best way to advance public welfare through the talents of authors and inventors in 'Science and useful Arts.'").

2. The Software Designer

The next role from the Rational Unified Process, that of the software designer, is most closely akin to what the Federal Circuit has described as a "software engineer." This role involves nearly all of the computer system design that is currently protected under patent law. The software designer does the majority of the work accomplished in the elaboration phase of the RUP and details how the system will function and how the various parts will interact. Like the graphic designer, the software designer should also have a basic understanding of the capabilities and limitations of the technology that will be used for the system; however, the majority of the design work should remain independent of these details. The software designer's main responsibility is to detail the business functionality that the computer system will be required to implement.

The patent system has traditionally focused on this role as the main innovator in computer systems; the software designer is responsible for building a computer system that accomplishes fixed requirements of utility in a novel way. This design work often lends itself well to patent applications as the innovation can be easily communicated to non-technical workers via UML diagrams. As a result, software designers have not been required to disclose the technical details required to implement the computer system in their patent applications because the UML designs sufficiently communicate to judges and juries the functions of the system.

3. The Application Architect

The worker in the role of the application architect designs the technological part of the system. The worker in this role has a deep knowledge of the available technologies and their respective capabilities and limitations. It is the job of the application architect to review the requirements laid out by the software designer in order to choose a technology and design an implementation strategy. This usually involves decisions such as deciding whether to build the system from scratch or use third-party software, determining the programming language to be used if new software is needed, and designing the overall structure of the computer system.[59] It is questionable whether this type of invention, a new use of existing technologies for a software implementation, can be protected under the current patent system.

59. This overall structure will include details such as the types and number of servers to be used, the method of data storage to be used, and a security model.

Computer system patents, such as the one at issue in *State Street*, seem to conflate these types of purely technological innovations with the process-driven functional advancements created by the larger computer system. The *State Street* patent detailed a process for pooling mutual fund assets, but also described in detail the server structure and technological implementation of the system.[60] It is worth noting that one lesson learned from the RUP is that the functional design and the technical design are largely independent of each other. The question of whether a purely technological advancement—likely written as a process claim that uses technology in a novel way to solve a problem with an existing business system—is patentable has not been addressed.

4. The Software Developer

A software developer is the worker who builds the nuts and bolts of the computer system. This worker implements the software design in the manner chosen by the application architect, and the role involves the actual programming of the business logic. The majority of the people working on building the computer system will fall within this role, and they require very little knowledge of the overall system design or the system architecture. Each worker is likely to be given a discrete subset of the overall process to build and will implement and test that part of the system. This role often has the lowest knowledge requirement at the system level and the highest level of knowledge of the detailed workings. Only the workers in this role must possess the ability to read the source code that makes the computer system function as designed. The work accomplished in this role is currently protected only under the copyright system, which, as previously noted protects only the expression and not the functional aspect of the source and object code. Thus, a work of developed software is infringed upon only if the exact code is copied and reused by an infringer.

5. Consequences of Recognizing the Roles of the Software Development Process

There are concerns to breaking down the patentable subject matter from the systems "as a whole" to each innovative step accomplished by each of the aforementioned workers. One such concern is the creation of a "patent thicket,"[61] or a tangle of patents that

60. State St. Bank & Trust Co. v. Signature Fin. Group, Inc., 149 F.3d 1368, 1371–72 (Fed. Cir. 1998).

61. Carl Shapiro, Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard Setting, *in* 1 INNOVATION POLICY AND THE ECONOMY 119–20 (Adam

would hinder further innovation. This concern should be alleviated somewhat by the combination of the unique aspects of software development along with the current patent requirements. Unlike the physical world, the virtual world, wherein software innovations exist, is not bound by the laws of physics, and there are far more possibilities for improvement. It is true that computing speed and power create some limitations, but Moore's law of ever-expanding computing power has proven that these limitations are only temporary and constantly in flux.[62] As a result, the only fixed limit on innovation in software design is human ingenuity. This creates the perfect breeding ground for "design around" engineering.[63]

The patent requirements also address this concern by enforcing validity standards of novelty, utility, and non-obviousness. By breaking down the roles in software development into the categories in which workers actually participate, the level of innovation that satisfies these requirements is arguably raised instead of lowered. An ordinary worker in the art of interface design will have more detailed knowledge about advancements in the art than an ordinary worker who participates only in software development more generally. This raises the bar for non-obviousness determinations by creating a higher standard for innovation; "one of ordinary skill in the art" will have a much greater level of skill when the worker has specialized in the art.

In both academic literature and the courts, the inquiry into the patentability of computer system innovations has shifted from subject matter concerns to questions of patent requirements.[64] The decision of how to view software development and what aspects merit patentability is inexorably tied to the case-by-case analysis of the requirements for a patent. Without breaking down the concept of the "software engineer" into more specific roles, the patent system will continue to reward patent protection in situations which

Jaffe et al. eds. 2001) (describing a patent thicket as "a dense web of overlapping intellectual property rights" that a company must navigate in order to "commercialize new technology").

62. In 1965, Gordon Moore observed an "exponential growth in the number of transistors per integrated circuit," and predicted that the number of transistors would continue to double every couple of years. *See* Intel Corporation, *Intel Research – Silicon – Moore's Law*, *available at* http://www.intel.com/research/silicon/mooreslaw.htm (last visited Mar. 25, 2005).

63. "Design around" engineering is the process of copying an invention with small changes sufficient to avoid patent infringement.

64. *See, e.g.*, Jared Earl Grusd, *Internet Business Methods: What Role Does and Should Patent Law Play?*, 4 VA. J.L. & TECH. 9 (1999).

may be unwarranted and deny patents to innovative advancements in more specific arts. To date, only a small number of the issued computer system patents have been litigated in the courts, and the application of the patent requirements to computer system patents is still in the early stages of development. The existing cases from the Federal Circuit illustrate the dangers of continuing down the path of the all-knowing "software engineer."

III.
HOW THE PATENT SYSTEM VIEWS SOFTWARE DEVELOPMENT: "ONE OF ORDINARY SKILL IN THE ART"

There is great importance in patent litigation of accurately determining the scope of the relevant art and the ordinary skill in the art—an inquiry in which the aforementioned roles should play a part. The usage of "one of ordinary skill in the art" comes into play in determinations of non-obviousness and whether a patent specification satisfies the disclosure requirements of enablement, written description, and best mode.[65] An analysis of these requirements demonstrates how they are tied up in determining the patentable aspects of computer systems; only the innovative step should be required to be non-obvious, and steps known within the art should not require disclosure. This section will review and establish the importance of these requirements and then attempt to discern how the Federal Circuit has applied these tests and factors to shape the current patentability of the computer system as a whole.

A. Graham v. John Deere Co.

Much of the discussion in the courts about determining the level of ordinary skill in the art has come when applying the framework from *Graham v. John Deere Co.*[66] This case laid out a test for determining non-obviousness, a requirement rooted in section 103 which provides that:

> [a] patent may not be obtained . . . if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains.[67]

65. *See infra* Part IV.
66. 383 U.S. 1 (1966).
67. 35 U.S.C. § 103(a) (2000).

The creation of the "one of ordinary skill in the art" test for non-obviousness was intended to set a lower standard for patentability than the previous "flash of creative genius" test[68] because a concern existed that judges were too often finding patents obvious.[69] The Federal Circuit has expressly made clear that "[i]nventors . . . possess something . . . which sets them apart from the workers of *ordinary* skill, and one should not go about determining obviousness under section 103 by inquiring into what *patentees* . . . would have known or would likely have done, faced with the revelations of references."[70] The "ordinary skill in the art" standard prevents judges and juries from facing what seems to be an inescapable paradox: for the invention to have been created it must have been obvious to the inventor.

In *Graham*, the Court listed three inquiries that must be made: 1) the scope and content of the prior art are to be determined; 2) differences between the prior art and the claims at issue are to be ascertained; 3) and the level of ordinary skill in the pertinent art must be resolved.[71] The Court also made clear that non-obviousness is a question of law based on factual determinations;[72] the determination of non-obviousness is reviewed de novo in appellate courts, while the underlying factual inquiries, such as the level of ordinary skill in the art, are only reviewed for clear error.[73] As a result, there is little guidance from published appellate opinions on how to best conduct this factual inquiry at the district court level.

The Federal Circuit has, however, articulated some factors to consider in determining the level of ordinary skill in the art, including the educational level of the inventor, the types of problems encountered in the art, prior art solutions to those problems, the rapidity with which innovations are made, the sophistication of the technology, and the educational level of the workers in the field.[74] The court has indicated that while in individual cases one or more may predominate, when these factors are present they should all be

68. *See* Cuno Eng'g Corp. v. Automatic Devices Corp., 314 U.S. 84, 91 (1941) (applying the flash of creative genius test).

69. DONALD S. CHISUM ET AL., PRINCIPLES OF PATENT LAW 598 (2d ed. 2001).

70. Standard Oil Co. v. Am. Cyanamid Co., 774 F.2d 448, 454 (Fed. Cir. 1985).

71. *Graham*, 383 U.S. at 17.

72. *Id.*

73. Ruiz v. A. B. Chance Co., 234 F.3d. 654, 663 (Fed. Cir. 2000).

74. *See* Custom Accessories v. Jeffrey-Allan Indus., 807 F.2d 955, 962 (Fed. Cir. 1986) ("Not all such factors may be present in every case, and one or more of them may predominate.") (citing Envtl. Designs, Ltd. v. Union Oil Co., 713 F.2d 693, 696 (Fed. Cir. 1983)).

considered.[75] The standard of "one of ordinary skill in the art" has also been applied to other areas of patent law such as the requirements of what must be disclosed in a patent application.[76]

B. Northern Telecom v. Datapoint Corp.

In the initial cases that laid the framework for determining the validity of software patents, the Federal Circuit has made clear that software code need not be disclosed in computer system patents. The patent litigated in *Northern Telecom, Inc. v. Datapoint Corp.* contained only a description of the software, equivalent to UML diagrams, used in the computer system instead of any examples of computer code.[77] The district court found that "the patent specification's lack of any information concerning the invention's programs would require a person reasonably skilled in the art of computer programming to experiment unduly in attempting to write programs."[78] The Federal Circuit overturned this finding and held that the description of what the software accomplished was sufficient and adequate for a skilled programmer.[79] In what is now oft quoted language the court reiterated that "the conversion of a complete thought (as expressed in English and mathematics, i.e. the known input, the desired output, the mathematical expressions needed and the methods of using those expressions) into a language a machine understands is necessarily a mere clerical function to a skilled programmer."[80] This was tempered, however, by the statement that "[t]he amount of disclosure . . . may vary according to the nature of the invention, the role of the program in carrying it out, and the complexity of the contemplated programming, all from the viewpoint of the skilled programmer."[81]

C. Fonar Corp. v. General Elec. Co.

The opportunity for accused infringers to argue for patent invalidity due to a lack of code disclosure was foreclosed in subsequent cases. In *Fonar Corp. v. General Electric Co.*,[82] General Electric

75. *Custom*, 807 F.2d at 962–63.

76. *See infra* Part IV. For more detail on the addition of the "person having ordinary skill in the art" ("PHOSITA"), see Cyril A. Soans, *Some Absurd Presumptions in Patent Cases*, 10 IDEA 433, 438–39 (1966).

77. 908 F.2d 931 (Fed. Cir. 1990).

78. *Id.* at 941.

79. *Id.* at 943.

80. *Id.* at 942 (quoting *In re* Sherwood, 613 F.2d 809, 817 n.6 (C.C.P.A. 1980)).

81. *Id.* at 941 (citing *In re* Sherwood, 613 F.2d at 817).

82. 107 F.3d 1543 (Fed. Cir. 1997).

argued that Fonar failed to disclose two software routines that the inventors knew were the best means for using the invention.[83] The Federal Circuit instead agreed with Fonar that the jury's finding that the patent satisfied the best mode requirement was supported by substantial evidence. The court relied heavily on testimony of one of the inventors who agreed that there was a "sufficient description to a software engineer, such as [himself], of what software need to be written."[84] The court stated that "[a]s a general rule, where software constitutes part of a best mode . . . , description of such a best mode is satisfied by a disclosure of the functions of the software."[85]

The strong language from the court established a bright line rule for those drafting patent applications that code disclosure is not required; merely a description of the functions of the software is sufficient. The Federal Circuit reiterated the point later that same year, holding "that when disclosure of software is required, it is generally sufficient if the functions of the software are disclosed, it usually being the case that creation of the specific source code is within the skill of the art."[86]

D. The United States Patent and Trademark Office

The United States Patent and Trademark Office guidelines for evaluating computer system patents also illustrate the view of the art of software development that existed at the time the guidelines were issued. The guidelines state that:

> In certain circumstances, as where self-documenting programming code is employed, use of programming language in a claim would be permissible because such program source code presents "sufficiently high-level language and descriptive identifiers" to make it universally understood to others in the art without the programmer having to insert any comments. Applicants should be encouraged to functionally define the steps the computer will perform rather than simply reciting source or object code instructions.[87]

This paragraph exactly illustrates the view of software development that has remained pervasive within the patent system. While it is permissible to use programming languages in a claim, many of

83. *Id.* at 1548.

84. *Id.*

85. *Id.* at 1549.

86. Robotic Vision Sys. v. View Eng'g, 112 F.3d 1163, 1166 (Fed. Cir. 1997).

87. Examination Guidelines for Computer-Related Inventions, 61 Fed. Reg. 7478, 7486 (Feb. 28, 1996).

the workers that provide the high-level business method design for computer systems may not know this language. Additionally, encouragement to provide a functional block diagram rather than any code at all in the specification assumes that the implementation of nearly all software designs is within the ordinary skill of any programmer in the art.

The Federal Circuit has recently made interesting pronouncements about software development in areas of patent law outside of non-obviousness and disclosure requirements that also influence the determination of "ordinary skill in the art." While the court has not directly returned to the issues of patentable aspects of computer systems or disclosure requirements, recent cases demonstrate a shift in the court's thinking. For instance, in *Creo Products, Inc. v. Presstek, Inc.*,[88] the Federal Circuit issued a decision that indirectly modified the previous precedents on what aspects of a computer system should be reviewed in the search for patentable subject matter.

E. Creo Products v. Presstek

Creo and Presstek were competing manufacturers of electronic imaging systems designed to be installed in printing presses.[89] The patent at issue disclosed a printing press capable of electronically correcting for mechanical imperfections that might otherwise result in errors. Experts from both companies testified on equivalence[90] at trial, and the district court found that the calculation method used was not equivalent because "it [did] not perform substantially the same function in substantially the same way to achieve substantially the same result."[91] Presstek argued that the use of a large look-up table, as opposed to performing calculations on the fly, was a design choice and that a software developer would consider the two means of calculation to be equivalents.

In making the determination of non-equivalence, the district court focused on the differences between the methods in terms of

88. 305 F.3d 1337 (Fed. Cir. 2002).

89. *Id.* at 1341.

90. The doctrine of equivalents is a rule of interpretation under which an accused infringer, although not a literal infringer, is still infringing the patent if the invention "performs substantially the same function in substantially the same way to obtain the same result" as the claimed invention. Warner-Jenkinson Co. v. Hilton Davis Chem. Co., 520 U.S. 17, 38 (1997) (quoting Machine Co. v. Murphy, 97 U.S. 120, 125 (1877)).

91. 305 F.3d at 1351.

complexity, computing speed, and time requirements.[92] These factors were acknowledged in the Federal Circuit's review of the case and allowed as part of the determination that the district court's decision was not clearly erroneous.[93] It is interesting to note that factors such as computing speed and time requirements do not look like parts of the software design "as a whole" but rather parts of the implementation at the architecture and programming levels.[94] This is a strange group of factors to take into consideration since the preceding cases appear to hold that the business method, or process, is the only patentable aspect of a computer system. The case marks a shift from allowing computer system patents only as a business method or general process patent to an analysis of the innovations that can occur within the implementation of the process to increase speed and usability.

F. Medical Instrumentation v. Elekta

The requirements for a software patent and the relevant skill in the art for legal determinations have also recently been muddled in *Medical Instrumentation and Diagnostics Corporation. v. Elekta AB*.[95] The claim at issue was whether Elekta was infringing patents held by Medical Instrumentation and Diagnostics Corp. ("MIDCO"). The patents in the case related to a system for planning surgical treatment using scanned images to locate the site on which a surgeon is supposed to operate and described a "method and apparatus for generating a video presentation of images from a variety of separate scanner imaging sources."[96] The key dispute on appeal was whether the district court was correct in including software as a structure for the conversion means.[97]

The district court decided that a person of ordinary skill in the art would understand software to be a corresponding structure. The majority opinion on appeal seemed quite deferential to expert testimony and stated that "MIDCO presented some evidence before the district court that a skilled programmer at the time of the appli-

92. *Id.*

93. *Id.*

94. For a detailed examination of the roles and levels of software development, see *supra* Part III.

95. 344 F.3d 1205 (Fed. Cir. 2003).

96. *Id.* at 1208.

97. *Id.* at 1209. The allegedly infringed claims were "means-plus-function" claims. "The duty of a patentee to clearly link or associate structure with the claimed function is the quid pro quo for allowing the patentee to express the claim in terms of function." *Id.* at 1211 (citing Budde v. Harley-Davidson, Inc., 250 F.3d 1369, 1370 (Fed. Cir. 2001)).

cation's filing could have written a program for digital-to-digital conversion of image size, and we have no reason to doubt that assertion."[98] Nonetheless, the majority held that even though the software was thus readily available, the failure to adequately describe it in the specification invalidated the patent.[99]

In dissent, Judge Newman clearly illustrated the majority opinion's break from precedent. The dissent agreed with the majority that the software procedures were well known to persons skilled in the art, but argued that this finding was relevant to the requirement of code disclosure. The dissent stated that "the patent specification need not 'teach software' and the writing of routine programs in order to teach how to practice the described method . . . . If one of skill in the programming art would have been able to write such a program without undue experimentation, the statutory requirements are met."[100] Judge Newman continued:

> For decades the rule and practice has been that such software need not be included in the specification. Over thirty years ago this court's predecessor endorsed this format, stating in *In re Ghiron*, that "if such a selection would be well within the skill of persons of ordinary skill in the art, such functional-type block diagrams may be acceptable and, in fact, preferable if they serve in conjunction with the rest of the specification to enable a person skilled in the art to make such a selection and practice the claimed invention with only a reasonable degree of routine experimentation."[101]

Judge Newman correctly summed up the majority's move by stating that it was now unclear what was required for proper disclosure of a computerized software routine,[102] asking "Is this court now requiring a five-foot-shelf of zeros and ones?"[103] It is, however, clear that the disclosure requirements for software patents, such as non-obviousness, now directly turn on a determination of the scope and skill in the art. The best way to solve the confusion is to ascertain where the innovation is occurring within each computer system patent and to utilize the level of ordinary skill employed by workers in that role to determine what should be the proper requirements for patent protection.

98. *Id.* at 1211–12.
99. *Id.* at 1212.
100. *Id.* at 1223.
101. *Id.* at 1224.
102. *Id.*
103. *Id.*

IV.
COMPUTER SYSTEM PATENT REQUIREMENTS VIEWED WITHIN THE RUP

As illustrated in Part III, prior to the recent shift in Federal Circuit analysis, determining the scope of the art for evaluating software patents had been a non-issue for the courts. Even today, very few courts have gone beyond the term "software engineer" or "software programmer" when analyzing software patents and tend to use these terms interchangeably.[104] Both of these general terms have been applied in the same context—as a worker in the art of computer systems who is capable of graphic design, software design, software programming, and architecture design.

When analyzing these terms in the context of the RUP, as detailed above, it appears that these terms are *not* interchangeable. Rather, the software engineer should be recognized as the person spearheading the elaboration phase effort to precisely lay out the actions and usage of the computer system with little knowledge required for software programming. The software programmer, on the other hand, should be viewed as the worker with little to no knowledge of the overall system and a deep knowledge of the programming art. Interestingly enough, however, the Federal Circuit has always assumed that the person in either of these roles can implement the system as well as design it.[105] This section will identify problems that continue to exist with the current patent requirements unless the all-knowing "software engineer" myth is replaced with a modern view of software development.

A. Non-obviousness

The *Graham* findings, detailed in Part III A., are a necessary part of any obviousness inquiry. The Federal Circuit reiterated this point in *Ruiz v. A.B. Chance Corporation*, saying that "[o]ur precedent clearly establishes that the district court must make *Graham* findings before invalidating a patent for obviousness."[106] The court gave a clear signal to district court judges by adding:

> In patent cases, the need for express *Graham* findings takes on an especially significant role because of an occasional tendency

104. *See, e.g.*, Med. Instrumentation and Diagnostics Corp. v. Elektra AB, 344 F.3d 1205, 1212, 1219 (Fed. Cir. 2003) (using a software programmer as one of ordinary skill in the art); Fonar Corp. v. Gen. Elec. Co., 107 F.3d 1543, 1548–49 (Fed. Cir. 1997) (using a software engineer as one of ordinary skill in the art).

105. *See generally* Northern Telecom, Inc. v. Datapoint Corp., 908 F.2d 931 (Fed. Cir. 1990).

106. Ruiz v. A. B. Chance Co., 234 F.3d. 654, 663 (Fed. Cir. 2000).

> of district courts to depart from the *Graham* test, and from the statutory standard of unobviousness that it helps determine, to the tempting but forbidden zone of hindsight. Thus, we must be convinced from the opinion that the district court actually applied *Graham* and must be presented with enough express and necessarily implied findings to know the basis of the trial court's opinion.[107]

The non-obviousness requirement for software patents has been characterized as a hard-to-meet standard under the *Graham* test.[108] I believe this is a result of the widespread usage of the "software engineer" concept in the legal system. The underlying factual inquiries require that the scope and content of the prior art be determined, differences between the prior art and the claims at issue are ascertained, and the level of ordinary skill in the pertinent art be resolved.[109] The fact-finders at trial are often faced with an expert who claims full knowledge of all the identified roles from the RUP, and after all aspects of the computer system are analyzed by the court, little seems not to be obvious to "one of ordinary skill" in all facets of software development.

If the jury were instead instructed to view the innovative step through the eyes of a worker possessing truly "ordinary skill in the art," either as a graphic designer, software designer, application architect, or a software developer, a better assessment of whether patent protection was merited could be obtained. Thus, an innovative user interface should not be reviewed for non-obviousness under the incorrect application of whether a "software engineer" could combine the prior art in all areas of software development to render the innovation unable to attain that "privileged position of a patent," requiring "more ingenuity . . . than the work of a mechanic skilled in the art."[110] Instead, the proper analysis should focus on whether an ordinary worker in the area of interface design, an art in its own right,[111] would have found the innovation obvious.

---

107. *Id.* at 663–64 (quoting Loctite Corp. v. Ultraseal Ltd., 781 F.2d 861, 873 (Fed. Cir. 1985)).

108. Dan L. Burk & Mark A. Lemley, *Is Patent Law Technology-Specific?*, 17 BERKELEY TECH. L.J. 1155, 1156–57 (2002).

109. Graham v. John Deere Co., 383 U.S. 1, 17 (1966).

110. Cuno Engineering Corp. v. Automatic Devices Corp., 314 U.S. 84, 90 (1941).

111. *See, e.g.*, Usability Professionals' Association, *UPA: Home Page*, *at* http://www.upassoc.org/ (last visited Mar. 25, 2005); Association for Computing Machinery, *ACM/SIGCHI*, *at* http://www.acm.org/sigchi/ (last visited Mar. 25, 2005) (detailing the special interest group for computer-human interaction within ACM).

This potential weakening of the non-obviousness requirement could lead to more overlapping software patents and thus stifle innovation, but this outcome is not a foregone conclusion. The determination of whether the level of "ordinary skill in the art" of a graphic designer, software designer, application architect, or software developer is higher or lower than that of the level of a "software engineer" will only be determined once factual inquiries begin to take place in the courts. I predict that while the scope of the art will narrow, since ordinary workers in the art will no longer be presumed to know everything about computer systems, the level of skill in the specific art at issue in most cases will rise as "ordinary" *specialists* will probably be more familiar with the prior art in the field than "ordinary" *generalists*.

B. Disclosure Requirements

The disclosure requirements of written description, enablement, and best mode also use the fictional character of "one of ordinary skill in the art." The use of the "software engineer" in the patent system has shaped these requirements to generally require little to no code disclosure. This creates a risk of granting a patent in situations where the invention may not be fully implemented, such as the example of the Linux operating system described in this section. The premature granting of a patent to an inventor who cannot adequately demonstrate to "one of ordinary skill in the art" that he is in possession of the invention raises concerns that can only be addressed by breaking down the "software engineer" into the aforementioned roles.

1. Written Description

The written description requirement has been given a broad policy rationale by the Federal Circuit, necessitating that the patent specification convey with reasonable clarity to those skilled in the art that the inventor was in possession of the invention.[112] The scope of the art is central to the requirement because "an inventor is not required to describe every detail of his invention"[113] and the "disclosure obligation varies according to the art to which the invention pertains."[114] For complex computer systems, a description of the functions of the software is often not enough to prove posses-

112. Vas-Cath Inc. v. Mahurkar, 935 F.2d 1555, 1563-64 (Fed. Cir. 1991).

113. *In re* Hayes Microcomputer Prods., Inc., 982 F.2d 1527, 1534 (Fed. Cir. 1992).

114. *Id.*

sion of the invention; instead, in certain circumstances, a working implementation of the software should also be required.

A good example of a complex computer system where the implementation, not just the overall software design, was the innovative hurdle is the Linux kernel[115] developed by Linus Torvalds. Prior to the kernel being added, there was an open source version of the UNIX operating system under development, known as GNU.[116] Much of the system was built through contributions from programmers across the globe, but the one problem that tormented the completion of the operation system was the lack of a working kernel. When Linus Torvalds added his kernel to the system, Linux became a popular operating system that is still widely used today.

So what should be made of this example? If the contributors to the GNU project could have accurately described the functions of the kernel with UML diagrams, should they have been entitled to patent protection even though they had no success in implementing the operating system? This example demonstrates that mere disclosure of the design for the software "as a whole" should not always be enough to grant patent protection. Granting computer system patents in these situations runs counter to a goal of disclosing working inventions to the public and should be avoided. By breaking down the "software engineer" into roles, it is easier to identify when implementation of a computer system is truly within the "ordinary skill in the art" of a software programmer, and, when it is not, the disclosure of a working implementation should be required in order to obtain patent protection.

2. Enablement

Another disclosure requirement in section 112 is the "enablement" requirement.[117] The enablement requirement ensures that the inventor has set forth sufficient information to enable a person skilled in the relevant art to make and use the claimed invention without undue experimentation.[118] In the Linux example, the ordinary skill of a software programmer was not sufficient to make

115. A kernel is the central piece of an operating system that handles the management of the computer resources for the programs being executed.

116. The GNU (GNU is a recursive acronym for "GNU's Not UNIX") Project was launched in 1984 to develop a complete UNIX style operating system available as free software. For more information, see http://www.gnu.org (last visited Mar. 25, 2005).

117. 35 U.S.C. § 112 (2000).

118. *In re* Wands, 858 F.2d 731, 736–37 (Fed. Cir. 1988).

implementation of the computer system from the written description a "mere clerical function."[119] The *quid pro quo*[120] of the patent system requires that the public be able to practice the invention. When the innovative step is not merely in the conception of the software, but rather the implementation, the patent specification should contain some actual code or pseudo-code.[121]

Another reason to require some sort of code disclosure is that the implementation of a computer system generally takes significant effort,[122] and may often constitute undue experimentation. The Federal Circuit has stated that:

> Whether undue experimentation is needed is not a single, simple factual determination, but rather is a conclusion reached by weighing many factual considerations . . . . [Factors] include (1) the quantity of experimentation necessary, (2) the amount of direction or guidance presented, (3) presence or absence of working examples, (4) the nature of the invention, (5) the state of the prior art, (6) the relative skill of those in the art, (7) the predictability or unpredictability of the art, and (8) the breadth of the claims.[123]

In these situations, determining whether the system could be built by any programmer of ordinary skill without undue experimentation may be a question of fact that varies by case. Thus, blanket statements from the Federal Circuit that the disclosure of source code is not required could actually run counter to the overall goals of the patent system. I propose that the patent system should no longer follow the assumption that any computer system can be built by a "software engineer" without undue experimentation when no code is disclosed in the patent specification. Instead, analyzing whether the innovation comes from the role of the graphic designer, the software designer, the application architect, or the software developer will lead to a better determination of enablement.

---

119. *See In re* Sherwood, 613 F.2d 809, 817 n.6 (C.C.P.A. 1980) (describing the writing of code as a "mere clerical function").

120. The "*quid pro quo*" of the patent system is that it "seeks to foster and reward innovation, [promote] disclosure of inventions, to stimulate further innovation and to permit the public to practice the invention once the patent expires, [and] assure that ideas in the public domain remain there for the free use of the public." Aronson v. Quick Point Pencil Co., 440 U.S. 257, 262 (1979).

121. Psuedo-code is commonly used by programmers to outline the algorithm for how a specific function will be coded before the effort is undertaken to actually implement the function in a programming language.

122. *See* JACOBSON, *supra* note 42, at 11.

123. *Wands*, 858 F.2d at 737.

3. Best Mode

The best mode requirement involves factual inquiries into whether the patent applicant had a best mode, and if there was a best mode whether it was disclosed in sufficient detail to allow one skilled in the art to practice it.[124] The Federal Circuit has consistently stated that the failure to disclose source code does not violate the best mode requirement because "writing code for such software is within the skill of the art, not requiring undue experimentation, once its functions have been disclosed."[125] This view is obviously consistent with the current practice of not requiring code disclosure for the written description or enablement requirements, but the best mode requirement raises additional issues.

The purpose of the best mode requirement "is to restrain inventors from applying for patents while at the same time concealing from the public preferred embodiments of their inventions which they have in fact conceived."[126] This requirement is intended to ensure that the patent applicant fully discloses the details of using the patented invention. I believe that the broad scope of "ordinary skill in the art" in software patents, evidenced by the "software engineer," allows computer system patentees to conceal the best mode for their invention. The innovative steps can remain undisclosed when the patent system assumes that "one of ordinary skill in the art" can design and build the user interface, implement the system design in computer code, and design the technical architecture. Instead, the proper test for best mode should be whether one who specializes in the innovative aspect of the invention would be able to practice the best mode with the detail described in the patent specification. Thus, I advocate this revised view of the best mode requirement in light of the modern software development process.

V.
CONCLUSION

As the proliferation of computer system patents continues and the relative importance of their impact on the economy increases, the patent system will need to re-examine its view of the software development process. If the patent system is to remain the preferred means for software protection, the courts will have to re-ex-

124. Fonar Corp. v. Gen. Elec. Co., 107 F.3d 1543, 1548 (Fed. Cir. 1997) (citing U.S. Gypsum Co. v. Nat'l Gypsum Co., 74 F.3d 1209, 1212 (Fed. Cir. 1996)).

125. *Id.* at 1548, *reiterated in* Bayer v. Schein Pharmaceuticals, Inc., 301 F.3d 1306, 1323 n.5 (Fed. Cir. 2002).

126. DeGeorge v. Bernier, 768 F.2d 1318, 1324 (Fed. Cir. 1985).

amine the scope of the art and the level of ordinary skill in the art of each innovative aspect of software development to dissect the concept of the all-knowing "software engineer." Breaking down the "software engineer" into specific roles will ensure that we are rewarding innovators in the implementation roles with patent protection instead of only rewarding the first innovator to describe or design a future innovation. It is important that we not deny all forms of intellectual property protection to important innovations that would benefit society if publicly disclosed.

Innovations can come from each of the roles identified, and the patent system will need to do a better job of identifying the innovative step in each computer system claim in order to properly utilize the requirements and doctrines that exist within the patent system. Requirements such as non-obviousness, written description, enablement, and best mode all serve a function in limiting the grant of a patent monopoly to innovations that are not obvious to "one of ordinary skill in the art" and are adequately described to the public in the *quid pro quo* at the heart of the patent system. By applying knowledge of the modern software development process, including its separate and distinct roles, to these determinations of patent validity, the courts will more accurately measure innovation and the required disclosure for computer system patents.