Software Patents

Gregory A. Stobbs, Second Edition

Chapter One: History of Software Patents

§1.01 History of Patents

- Why study history?: Software patents have raised numerous issues over the nature of patentable inventions (and the differentiation from unpatentable ideas), since software is closer to a pure "idea" than any other kind of invention – it is helpful to consider the historic basis of the patent system as the justification for allowing or disallowing software patents
- Greek patents: During the third century B.C., the ancient Greeks apparently granted monopolies for novel recipes
- Early English patents: The ancient Greek concept was applied in the Medieval England for economic development, where merchants were rewarded for traveling abroad and bringing back useful goods and technologies – eventually, English merchant guilds held local monopoly rights over particular trades, and in 1326, the government began granting "open letters" ("letters patent") extending guilds' monopoly rights to novel imported goods (the textile industry made particularly good use of this system) – this system was eventually extended to novel imported inventions (1440: monopoly granted to John of Sheidame for a process of manufacturing salt)
- English patents under Queen Elizabeth I: After a hiatus under the Tudors, Queen Elizabeth I expanded the patent system and secured the importation of 20 major technologies in 10 years – famous patent to Giacopo Acontio for a newly invented furnace: "Nothing is more honest than that those who by searching have found out things useful to the public should have some fruits of their rights and labors, as meanwhile they abandon all other modes of gain, are at much expense in experiments, and often sustain much loss" – the Queen referred to her power to grant patents as "the chiefest flower of her garden and the principal and head pearl of her crown and diadem" – however, patent abuse grew out of control (e.g., tenfold increase in price of salt) and the monopoly breadth was based on the Queen's personal attachment to the patentee – these consequences drew public protest and contention in the House of Commons, forcing the Queen to withdraw the broadest monopolies and creating common-law court system for review of patent validity (beginning of patent litigation)
- *Case of Monopolies* (1602): This important lawsuit involved a patent for playing cards – the government contented that playing cards were an object of amusement that the Queen could entirely abolish, so her patent power should be absolute – the defense demonstrated that patent monopolies raise the prices of goods, and thus should be limited to novel inventions (invented by the patentee) and for short durations; the defense also characterized patents as contracts, and patent monopolies as consideration for inventive efforts – court held that patent was

invalid: any activity that prevents idleness isn't just a leisure activity, but is for the good of the commonwealth, and hence can't be unduly restricted – the court drew an analogy to hunting (which didn't require a license), as differentiated from establishing a public park (required a license) – in this regard, the Queen had been incorrect in presuming that the monopoly would promote the public interest, since monopolies necessarily raise prices and reduce quality; hence, patents much be granted only by necessity – finally, the court found that a patent for a deck of playing cards necessarily impacted the act of playing games with cards, and since the Queen couldn't ban the latter, she could also not ban the former

- English patents under King James I and the Statute of Monopolies: Queen Elizabeth's successor suspended all of her monopolies and ordered a validity inquiry for each one; however, these patents were found valid, and the patent system grew exponentially under his leadership (Francis Bacon: "Now among all the benefits that could be conferred upon mankind, I discovered none so great as the discovery of new arts for the bettering of human life") – however, King James I abused the patent system much as Elizabeth had, and this prompted Parliament to pass the Statute of Monopolies – this statute outlawed patent monopolies except for "letters patent and grants of privilege for the term of fourteen years or under, hereafter to be made of the sole working or making of any manner of new manufactures within this realm, to the true and first inventor and inventors of such manufactures, which others at the time of making such letter s patents and grants shall not use, so as also they be not contrary to law, nor mischievous to the state, by raising prices of commodities at home, or hurt of trade, or generally inconvenient, the said fourteen years to be accounted from the date of the first letters patent or grants of such privilege hereafter to be made"
- Written enablement requirement: The Statute of Monopolies did not require any teaching of how to use the invention, but merely that the letters patent must describe it – court decisions over a few hundred years of patents created a written description requirement – by the end of this evolutionary phase, the requirements of patentability resembled the requirements applied by today's patent systems

§1.02 History of Patent Law in the United States

- U.S. Consitution and the Patent Act of 1790: Jefferson and Madison were strong proponents of a federal patent system – Jefferson initially opposed the grant of patent monopolies, but conceded to Madison's arguments, and Madison drafted the constitutional provision for copyrights and patents – the first federal Congress moved quickly to implement the constitutional power to grant patents – Congress initially issued patents on an ad hoc basis by petition, but quickly realized that it couldn't cope with the volume – Washington urged the first Congress to create a patent system (as well as a copyright system), and Congress reacted by passing the first Patent Act, which created an agency under the Department of State – this agency involved a panel staffed by the Secretary of State, Secretary of War, and Attorney General ("Commissioners for the Promotion of Useful Arts"), which issued patents upon majority vote for "any useful art, manufacture, or device, or any improvement therein not before known or used," that was "sufficiently useful and important" – as a member of the first panel of Commissioners, Thomas

Jefferson took a deep interest in patents, examining every one and authorizing only 57 patents in the first three years

- Patent Act of 1793: Before long, the workload of the Commissioners grew unmanageable – Jefferson headed an effort to replace the Commissioners examination system with a patent registration system, which dispensed with the examination and frequent rejection of patent applications – this system was codified in the Patent Act of 1793, which limited the role of the federal government to a record-keeper of registrations, which were issued virtually automatically upon request and payment of $30 filing fee – however, Jefferson soon regretted the shift and argued for the benefits of examination, and this registration system was eventually disbanded – however, the 1793 Act is noteworthy for some new points of law: the concept an inventors of an original invention and an improvement thereof have no inherent right to use the other's inventions; changes in form or proportions are not patentable; the addition of a written description requirement, and the filing of an oath or affirmation; and the creation of interference proceedings resolved by a panel of three arbitrators – under this Patent Act, Samuel Morse obtained several patents for the telegraph, and the Supreme Ct issued a famous ruling that rejected Morse's contention that his patents covered any transmission of communication signals by electromagnetism
- Patent Act of 1836: Senator John Ruggles of Maine headed an effort to create a new examination system, culminating with the filing of a historic report to the Senate on the discouraging results of the registration system ("the country becomes flooded with patent monopolies, embarrassing to *bona fide* patentees, whose rights are thus invaded from all sides… it opens the door to frauds, which have already become extensive and serious") – this prompted Congress to pass the Patent Act of 1836 – noteworthy features: 14-year term, extendable to 21 years by approval of patent board; widely differing fees for U.S., British, and foreign applicants; requirements of novelty, utility, and originality; process of appealing patent decisions to patent board – this act was amended in 1842 to add design patents, and again in 1861 to fix the patent term at 17 years (with no option of extension), and again in 1930s to add two types of plant patents
- Patent Act of 1952: This revision arose from a congressional effort to unify the U.S. Code – the most noteworthy change in this act was the addition of an obviousness requirement (35 USC §103), which finally codified a common-law principle created through 100 years of jurisprudence
- 1966 Presidential Report on the Patent System: President Johnson ordered a comprehensive review of the patent system by several branches of the government – the final report of the Commission concluded that the current patent system was "gridlocked" and not representative of the changing economy – the report recommended many sweeping changes to patent law and practice: first-to-file system of priority; abandoning the plant and design patent systems, and prohibiting broadening reissue practice; permitting filings in the name of an assignee; publication of all patent applications 18 months after filing, and a public comment system following publication; option of deferred examination; modification of patent term to 20 years from the date of filing; grant of deference

to Patent Office decisions (reversal only for clear error), and review of CCPA decisions by federal appellate court; amendment practice; ex parte reexamination procedure; pre-issuance damages for infringement; extension of infringement to the importation of an unpatented composition made abroad by a method patented in the U.S.; and effort to harmonize international practice – notably, this Commission recommended disallowing patents on software, due to inadequate search and examination procedures by USPTO and to the tremendous rate at which new art was being generated, and recommended that software be protectable only by copyright – most of the Commission's recommendations were rejected, including its proscriptive ban on software patents

- National Commission on New Technological Uses of Copyrighted Works (CONTU report): While formulating a revision of federal copyright law (subsequently enacted in 1976), Congress ordered a report on the adequacy of copyright protection for new technologies, including software – the Software Subcommittee considered a hybrid system of protection that would include patent-like protection, but concluded that such a hybrid system "would restrict society's access to information to a greater extent than does copyright because such other forms afford proprietors far greater monopoly power" – the Subcommittee observed that copyrights protect expression, while patents protect ideas; also noted that copyright granted protection of more limited scope but much longer duration to all works, regardless of quality, while patent law granted shorter but stronger protection to sufficiently meritorious works – the Subcommittee concluded that copyright law was sufficiently protective for software; this recommendation might have arisen from the fact that the Commission was only revising copyright law, not patent law, as evidenced by its characterization of software as a "writing," and as "the explanation of a process and not the process itself" – however, the Subcommittee drew a distinction between the idea and its expression (the "description of an activity" and the actual "activity"), and did not reject the idea of patents for the idea underlying the software – but in acknowledging the potential for software patents, the Subcommittee noted that court precedent rejected the patentability of software (citing *Gottschalk v. Benson*), that software should be patentable only if it meets the strict requirements of 35 USC, and that detecting and proving infringement for patent enforcement was difficult
- 1994 Software Patent Hearings: In 1994, the USPTO held hearings to solicit public comment on software patents – many anti-software-patent advocates, primarily individual programmers and companies that had been punished for infringement, presented the same arguments as presented in response to the CONTU Subcommittee report, based on the burgeoning state of the software industry in the absence of patent protection – most large companies, including Microsoft, and many small startups advocated for software patents

§1.03 History of Computers

- Charles Babbage: In 1822, Babbage built a prototype mechanical calculator named the "difference engine" to replace inaccurate mathematics tables; he worked on it for ten years under British funding but failed to produce a working device – in 1834, he conceived an "analytical engine" that included a memory

unit and a CPU-like "mill," and accepted programs via punched cards – unfortunately, the British government refused to fund his work, so the more sophisticated device was never constructed

- IBM: The inventor of the 1890 census machine started a company called the Tabulating Machine Company to develop the idea – this company was later sold to the Computing-Tabulating-Recording Company, which eventually become IBM, focusing on punched-card machines – IBM's first products were simple but large tabulating machines that accepted punched cards (each representing a large integer) and accumulated a running sum – many refinements followed, including the ability to handle negative numbers, multiplication, sorting, mathematical comparisons, and eventually the ability to provide custom programs for the punched-card machines to carry out different mathematical computations
- ENIAC: In 1941, promoted by defense funding in anticipation of World War II, Dr. John Mauchly and Persper Eckert (borrowing ideas from Dr. John Atanasoff) developed the Electronic Numerical Integrator and Calculator (ENIAC), which could perform 5,000 additions per second (compared with the mechanical-relay-based machines with a top speed of 10 additions per second) – ENIAC had 18,000 vacuum tubes and cost over $500,000 to develop in 1941 – this technology was later adapted for the 1951 Census as the UNIVAC; this device was patented, but during ensuing patent litigation (*Honeywell, Inc. v. Sperry Rand Corp.* (1973)), the patent was declared invalid due to late filing
- Precursors to computers: In 1949, Edmund Berkeley attempted to differentiate contemporary computers with the "mechanical brains" that had existed for centuries based on two criteria: (1) the capability of moving and storing data without the need for human intervention, and (2) the capability of providing a program, rather than requiring a human operator for every step – the latter quality provides a flexibility that differentiates the machine from fixed-purpose mechanical devices, and is the "soul of computer self-control" – the idea of a program began with the music box/automated organ, and was adapted to automatic looms for producing complex weaves without human intervention via a series of punched cards – this "punched card" model was further adapted for use in manufacturing and data processing; e.g., 1890 census was computed via a punched-card data-gathering machine

§1.04 History of Software

- Unbundling of software: The first machines were programmed entirely in their own machine language, so there was no market for independent programs – moreover, the owners of the machines did not have the expertise to write custom programs, and just ran the software provided with the machine – during the late 1950's and 1960's, owners began learning how to program computers – manufacturers appreciated this trend because it drove demand; IBM created a user group called SHARE to facilitate the independent creation of programs, and IBM freely distributed the best programs created by this group – when other companies started selling software, IBM had to choose whether to maintain the tie between its software and hardware, or whether to sell them as separate products – its president at the time pinned its decision on whether or not the software could be

protected from being copied – eventually, market pressure and antitrust litigation forced IBM to unbundled its software and hardware sales

- Early cases rejecting software patents: At the time the software market was created in the 1970's, the Supreme Ct was taking an aggressive stance against patents, based on the fear that Congress sought to remove information from the public domain (*Graham v. John Deere Co.* (1966): "Congress may not authorize the issuance of patents whose effects are to remove existent knowledge from the public domain, or to restrict free access to materials already available") – this hostility is apparent in the opinion in *Gottschalk v. Benson* (1972), first modern software patent case, involving a method of converting binary-coded decimal numbers to binary numbers; in holding that this technique was unpatentable, the Supreme Ct held that upholding the patent would "wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself" – although the result of this case was correct (because the algorithm was not novel), its reasoning conflated the concepts of an idea, a formula, and an algorithm as one entity, and failed to differentiate patentable from unpatentable subject matter– even so, the Supreme Ct denied that its ruling constituted a ban on software patents – this rationale was cited in *Parker v. Flook* (1978), involving a method of controlling a catalytic process by updating certain alarm limits; in rejecting patentability, the Supreme Ct held that "a claim for an improved method of calculation, even when tied to a specific end use, is unpatentable subject matter" – the dissenting opinions in this case noted that the majority had imported issues of novelty and obviousness into a §101 patentability determination
- Subsequent cases supporting software patents: *Diamond v. Chakrabarty* (1980) involved a patent for a genetically engineered bacterium for decomposing crude oil – in supporting patentability of this novel "composition," the Supreme Ct consulted the Patent Act of 1793, the motivation of Thomas Jefferson, and the Congressional changes to patent law over the centuries, and concluded that Congress intended statutory subject matter must "include anything under the sun that is made by man" – the Court noted that "laws of nature, physical phenomena, and abstract ideas" remained unpatentable, and cited *Parker v. Flook* and *Gottschalk v. Benson* as examples of unpatentable ideas – this rule was applied in *Diamond v. Diehr* (1981), involving a method for controlling a furnace for preparing synthetically cured rubber based on the Arrhenius equation; the examiner and Commissioner of the USPTO rejected this application as claiming unpatentable subject matter, but the Supreme Ct reversed: "Arrhenius' equation is not patentable in isolation, but when a process for curing rubber is devised which incorporates it in a more efficient solution of the equation, that process is at the very least not barred at the threshold by §101" – thus, the Supreme Ct advocated for the examination of the claimed invention "as a whole," and not just an assessment of which parts are already known and not novel

Chapter Two: Software Technology Primer for the Attorney

§2.01 Software Primer

- Overview: This chapter reviews the technical details of software, with an eye toward counseling software developers on legal issues and writing patent applications

§2.02 Nature of Computers and Software

- The nature of software: Software enables a computer to control information flow – a program is both a written set of instructions, and a series of events occurring in real time – even the earliest programmers foresaw the vast potential of software as automated language translators, large databases of information, optical character recognition and text printing, and transcription of spoken language; all of these tasks are routinely performed by computers today – however, the early characterization of computing machinery as a "mechanical brain" fueled the early view of software as a set of "mental steps," which had been a basis for rejecting patentability since the 1800's – of course, even today's machines are still too primitive to be compared with "brains," thereby invalidating the "mental steps" argument (but this argument is likely to arise again when machines are capable of performing an equivalent of human thought)

§2.03 Computer Science, a Science of Complexity

- Abstraction: The simplest modern computer includes an input device, a memory store, a processing unit, and an output device; the software is the set of data that instructs the processor in managing the other three devices – however, as computers grow increasingly powerful and complex, software has evolved into ever-higher levels of abstraction – modern software is usually organized as a set of layers, each functioning on the layer below it and serving as a platform for a more abstract level of programming – the key concept in this model: each layer hides the underlying details from the levels above it, allowing more complex programs to be written without worrying about the details; as a result, the higher-level program is both more sophisticated and less prone to more routine mistakes – e.g., a database management system is built on top of a standard input/output software layer, so that it doesn't have to deal with the details of reading or writing data; it can just assume that data can be read and stored
- Software interface: Any two layers of abstraction must share an interface for communication – the breadth and coordination of the layers is governed by the interface design – this also allows a handy division of labor: different groups of programmers can simultaneously write different layers of a software package, where each only focuses on the responsibilities of its part and the interface with other units – this also supports software reuse: a module with a well-designed interface can be repurposed for a different project

§2.04 Building Software

- Software engineering: The process of building software involves the same stepwise approach as for building other kinds of complex structures: defining the problem, design a solution, implement and test each part of the solution, integrate the parts, and test and maintain the whole unit – many variations on this process are available, and software design is now recognized as a distinct kind of engineering with its own challenges
- Software engineering models: Many models of software design exist: "waterfall model": this is the traditional model featuring a stepwise progression, but is

criticized as inflexible, since not all requirements and problems can be known at early stages – "rapid prototyping": this cyclic process involves repeated rounds of requirements analysis, design, and quick prototyping; each round of prototypes is evaluated in order to refine the requirements for the next round – however, the end point of this process is not well-defined; any prototype can be considered either the final product or the basis for improvement – moreover, the rapid prototyping step often creates inefficient code and poor design choices that might not be identified or corrected

§2.05 Programming Models

- Sequential programming: Before 1963, programs were generally viewed as a linear set of steps – however, this crude organization did not scale well to more complex tasks – the GOTO statement was routinely used to control program flow, but since the jump had no explicit meaning, even slightly complex programs made of such statement became unreadable "spaghetti code" – a famous publication condemning the GOTO statement persuasively argued that better methods of organizing code were needed
- Procedural programming: This programming model involves housing code in a series of modules of varying complexity – this kind of design can be formulated by approaching a problem at the highest level of abstraction, identifying key steps, and recursively breaking those steps into modules containing lower levels of abstraction – this "structured programming" model focuses on building self-contained units of code that utilize lower-level code units, and that are invoked by higher-level code units – thus, the programmer focuses on the flow of data and code through the computer
- Object-oriented programming: This programming model involves modeling the problem as a series of interacting objects; rather than focusing on the details of the computer, the programmer can focus on the details of the problem – when designing some sales software, a procedural programmer might focus on the structure of the database records as involving data about salespeople, items, costs, etc.; but an object-oriented programmer might focus on entities like Customers, Salespeople, Products, and Orders, and can create a distinct code module representing each entity – thus, the unimportant internal details of how these objects work are present but hidden ("encapsulation")

§2.06 Programming Languages

- Types of programming languages: Programs must be written in a specific set of instructions that the computer can understand – however, programmers rarely care about the specific powers of a particular computer, and can operate more efficiently by focusing on the abstract logic of the algorithm – a programming "language" enables the programmer to model the logic for a particular computer that can apply it to its own hardware capabilities; the programmer writes "source code" at a high level of abstraction, and then uses some software to translate ("compile") it into a low-level set of instructions – hundreds of languages have been written, and computer scientists generally view them as different "generations" – first-generation languages (e.g., BASIC, FORTRAN): subroutines accessing global data – second-generation languages (COBOL, Lisp): procedural design with data passed as parameters between functions – third-

generation languages (C, Pascal): organization of code into libraries that can be separately compiled and linked together at a later time – fourth-generation languages (C++, Ada): object-oriented programming

- Language selection: Of course, a programmer must decide on a programming language early in the design phase of a project – each language can handle certain kinds of tasks particularly well (e.g., Lisp is useful for parsing natural English statements; C is useful for high-performance computing and lower-level systems programming; and FORTRAN is useful for mathematical analysis) – this choice may also be dictated by the availability of compiler software, the needs of the customer (e.g., the federal government requires many projects to be written in ADA), the details of the hardware on which the program will execute, or simply the familiarity of the development team with a particular language
- Creating and compiling source code: Programmers write source code using a software development package that closely resembles word processing software – more complex development packages include syntax checkers and context-sensitive help – when the code has been written, the programmer sends it to the compiler for translation into low-level machine-readable instructions – the resulting set of data ("object code") may be a complete program, or may be a module that must be linked to other programs before being run; this latter concept is commonplace in large projects involving many distinct programmers and programming teams – also, the object code must usually be linked to object code libraries provided by the operating system
- Computer-aided software engineering (CASE) tools: These tools accelerate the development of software by writing typical source code for the programmer – the programmer merely describes the kind of software he wants, and the tool authors the source code based on his preferences; the programmer may then modify the source code for customization – these tools complement the typical programming tools of the trade, including compilers, debuggers, project management tools, and reverse-engineering tools (decompilers, disassemblers, etc.)

§2.07 Large-Scale System Design

- Large-scale systems: The complexity of modern software systems is manageable by the "divide and conquer" approach to software design; this enables programming teams to create tremendously complex software by producing incremental steps – concepts like layering, abstraction, interfaces, and encapsulation help focus the attention of the programmer – some companies choose to protect their products by maintaining their software as proprietary ("closed") and not disclosing the details to competitors; others choose to allow customers (and competitors) to further develop their products by publishing the details ("open" software) and standardizing the interfaces – two common "open" standards: the Open System Interconnection (OSI) model of communications networks comprises seven layers of network communication, each of which adds some features to the layer immediately below it; TCP/IP adapts five of the layers of the OSI model for a specific network communication protocol
- Client-server technology: This model involves a division of labor between machines that provide services ("servers") and machines that consume services ("clients") – e.g., a network may involve some machines that store data, and other

machines that request changes to the stored data – key benefits of this architecture include the ability of the server can provide a host of specialized services to enable clients to focus on more sophisticated tasks, and the ability of the server to provide the same service simultaneously to multiple clients, and even to allow clients to work together in utilizing services – client-server technology is particularly useful for designing robust databases, e.g., a patent docketing system; many different kinds of employees might be working on the same set of patent application data for different reasons (prosecution, docketing, billing, client communication)

§2.08 Microsoft Windows

- Overview of Windows: Microsoft produces a series of operating systems known as Windows that provide a computing environment for running programs – many features distinguish Windows as an improvement over the older command-line interface operating system known as MS-DOS: graphical user interface (a visual interface that visually ties together a program with input devices like a mouse and keyboard), a device-independent graphics layer (graphics can be drawn on a wide variety of display hardware, each of which merely provides a "device driver" that interfaces with the operating system), and multitasking (the ability to run multiple programs in parallel; the user may organize the input and output of many running programs simultaneously, and the programs can share computing resources like memory and processing time)
- Windows internals: Windows also overcame the "640K boundary" present in MS-DOS, which was a hard limit on the maximum size of a program based on the capabilities of early computing hardware (8088/80286 processors); this limitation hugely constricted the maximum complexity of programs and the capacity to store data, and by overcoming this technical limitation, Windows enabled programmers to write much more robust programs – also, while MS-DOS provided some basic services to programmers, Windows operates internally as a client-server architecture, providing a number of complex services useful to programmers – e.g., while MS-DOS can simply deliver keyboard input to the currently running program, Windows can coordinate keyboard input among several programs and can intercept reserved keyboard input for its own use – Windows does this by providing every program with a message queue, so that it can receive messages in isolation from unrelated programs and the details of the Windows environment; under this system, Windows can provide programs with very sophisticated messages, such as the fact that a user has selected a particular menu item (rather than requiring the program to determine this from raw keyboard and mouse input) – also, Windows programs can take advantage of a very robust advanced programming interface (API) for tasks like memory management and drawing device-independent graphics
- Late binding: This concept involves assigning physical memory addresses to variables, objects, and code modules as close to the time of use as possible, instead of far in advance – this allows the programmer to change the details of the bound object before it's used – one common use of this concept is in the connection between a high-level program to a lower-level module; the implementation of the lower-level module can be changed (improved, debugged,

etc.) without needing to change the higher-level program – another use: programs can change their internal operation on the fly (e.g., a user might wish to redirect word processing output from one printer to another; the word processing program can do so simply by remapping its memory references to those of the other printer driver) – late binding is made possible by the use of a linker, which associates virtual memory references with actual addresses on a "just-in-time" basis – also, this allows better resource allocation: several programs can use the same module at once

§2.09 Software Design Patterns

- Design patterns: As demonstrated by the growing size of Windows with each iteration, software products are clearly increasing in complexity – to address such complexity, software engineering borrows a concept from classical architecture: design patterns, the idea that a particular kind of problem can arise in many contexts, and can be solved in the same ways (though the details vary) – this gives rise to the concept of software reuse – recent advances in software engineering have identified 23 common design patterns, falling generally into three categories: creational patterns, structural patterns, and behavioral patterns

§2.10 Creational Patterns

- Creational patterns: Object-oriented design involves packaging data and code as "objects" that represent different components of a solution – the different creational patterns define what kinds of objects are created, who creates them, and when and how they are created – abstract factory pattern: the interface generally defines the broad features of a class, but allows each platform to define the particulars (e.g., a generic "scrollbar" class, implemented differently by each OS) – builder pattern: the constructor joins together many other objects that relate to each other in a particular way (e.g., a human language phrase constructor: perhaps different human languages have different parts of speech, but concepts must still be able to relate to each other in a phrase) – factory method pattern: the interface doesn't define a constructor for an abstract class, but merely contains a stub constructor that can be filled in later for a specific object – prototype pattern: a generic template object is developed as a prototype, which may later be cloned and customized for any specific application – singleton pattern: exactly one instance of the class must be created and must be globally available

§2.11 Structural Patterns

- Structural patterns: These design patterns describe how classes might organize and manipulate their member variables – adapter pattern: one object encapsulates another object and translates calls between an interface and the encapsulated object – bridge pattern: this pattern intermediates between an object and its abstraction, allowing each to vary independently – composite pattern: a collection object binds together one or several primitives, and can be nested within another collection object in a hierarchial fashion – decorator pattern: a wrapper object that adds a feature to the encapsulated object (a way to add features to the encapsulated class on an as-needed basis, without having to modify/expand the class definition) – façade pattern: a unified interface that can present a set of dissimilar objects in a similar manner (e.g., the windowing interface for most GUIs presents all applications with a common set of features, such as scrollbars

all looking alike) – flyweight pattern: in a scenario where many objects will encapsulate one of a few kinds of objects as a descriptor, this pattern involves hosting a complete set of these objects in a specific location, and allowing all encapsulating objects to point to the most applicable one (rather than multiply instantiating each one) – proxy pattern: one object stands in as a stub for a much more resource-intensive object until the latter is needed

§2.12 Behavioral Patterns

- Behavioral patterns: These patterns describe how objects interact with each other – chain of responsibility pattern: a data object is serially offered to a set of handler objects to see if any of them want to process it – command pattern or transaction pattern: an object is responsible for performing some action, but the details of that action won't be known until later, when a set of preconditions is complete – interpreter pattern: objects are defined and arranged as grammar rules for parsing a complex language – iterator pattern: a class for traversing the elements of a specified collection type in a particular manner, so that the collection class itself doesn't need to offer a number of traversal methods and keep track of an internal pointer – mediator pattern: one object serves as the interface between two other objects – memento pattern: one object represents the internal state of another object at a specific point in time; useful for storing an object state in case a backup is needed – observer pattern: one object creates a representation of another object, and changes the presentation as the state of the observed object is updated – state pattern: an object alters its behavior as its internal state changes (e..g, the functionality of a computer might change depending on its power source and power management settings) – strategy pattern: a group of objects that each represent a different way to solve a problem (e.g., an assortment of objects that each encapsulate a different sorting algorithms); clients may choose whichever solution strategy they wish – template method pattern: a higher level of abstraction of the strategy pattern, where the solutions define the rough characteristics of a solution method, and allow the client to specify further details later – visitor pattern: the opposite of encapsulation, this pattern involves storing a set of objects apart from other objects that operate on and maintain them; this requires exposing the internal state of each object set to the visitor objects

§2.13 Software Agent Technology

- Software agent technology: This new model of client/server architecture involves a service that permits the execution of a client "software agent" within its environment – instead of the client software touching the server through the interface, a software agent is actually uploaded to the server, hosted there while it does its work, and allowed to return results to the client – the agent is almost completely preprogrammed before interacting with the server, and then operates with little or no control by the client user – this requires concepts such as autonomy, prioritization, cooperation with the server environment, and some learning capabilities – of course, novel software agents and improvements on known software agents are both patentable; the specification should describe the environment in which such an agent operates, as well as its behavioral attributes (learning, prioritization, etc.)

Chapter Three: A Review of Legal Concepts for the Software Professional

§3.01 Introduction

- Introduction: This chapter presents basic legal concepts for non-attorneys who want to understand software patent

§3.02 Nature of Law

- The history of law: American law most directly stems from the English law, which inherited properties from ancient Green and Roman law – at its most basic, law is a way to settle disputes without resorting to violence or vengeance, and deals with both intentional wrongdoing and negligence (inadvertent but avoidable wrongdoing) – law is necessarily adversarial, and the law usually intervenes only where all involved parties want its participation, based on their interests, means, and the value of the dispute – however, the law is not a system of absolute rules; no such set of rules could govern the complexities and dynamics of modern society without creating vast injustices
- The court system: Courts are organized as a pyramidal hierarchy – the base of the pyramid includes trial courts, where a dispute is resolved by presenting evidence and witnesses to a judge (who decides issues of procedure) and jurors (who decide issues of fact) – the next step on the pyramid involves appellate courts, which only resolve arguments as to whether or not the decisions of the trial court were procedurally and legally correct (i.e., whether the law was properly applied to the factual findings); the parties have a right to have the appellate ct hear all such disputes – at the top is the Supreme Court, which hears only appeals from cases that involve major judicial policy decisions; the Supreme Ct hears appeals from appellate ct decisions only at its discretion (*certiorari*) – every state has a court system to resolve state law issues, and the federal government has a court system to resolve federal law issues (including state law issues that overlap federal law, including the U.S. Constitution)

§3.03 Common Law and Statutory Law

- Common law vs. statutory law: Originally, common law included the "unwritten" law reflecting the enforced customs of the region, while statutory law is the written set of laws – eventually, legal systems began writing down their common law as well – today, the main distinction now is that common law is a written set of laws inductively fashioned to model jurisprudence, while statutory law is the set of laws passed by a legislature – American law involves both common law and statutory law
- American common law: Our common law arises from judicial decisions in past cases: although no two cases are identical, certain common themes may arise from a set of cases that suggest a general principle – these written decisions serve as a record of the holding for review by the appellate ct; these opinions are often strengthened by citing and following the rationale of earlier decisions; this establishes a precedent – appellate cts are willing to overturn decisions based on inapplicable or incorrect precedent, but are less willing to overturn factual findings unless they are clearly unsupportable – these concepts of precedent and deference make the legal system more predictable and fair, and conserve court resources for the highest priorities, but greatly increase the amount of force

necessary to change the legal system (the proponents of change need a factual situation directly on point to arise, and need to assemble a very persuasive argument from an array of dissimilar cases)

- American statutory law: State statutory law is enacted and maintained by a state legislature, while federal statutory law is enacted and maintained by the U.S. Congress – when a bill is passed into law, it is assigned a public law number and published (federal law is published by the U.S. Government Printing Office) – it is also included in the chronological volume of "session laws" passed during a period of the legislature – finally, it is incorporated into the body of law, which is organized into topics called "titles"; however, this topical organization is not perfect (e.g., the Atomic Energy Act, part of America's energy law, contains important changes to the patent system)

§3.04 Property Law and the Foundations of Patent Law

- Patents as property: Property law describes a set of rights owned by a person over a tangible object – the legal system created these rules to resolve disputes over property, and regards as *prima facie* evidence of ownership the possession of a "deed," which is a legal instrument that entitles the bearer to enforce those rights in a court of law against violators – of course, property rights can be jointly held by several owners, each of which may freely control his rights, sell them, etc. – a patent is essentially a deed that provides a monopoly right to a feature of the economy; in the early days of patents, this feature was a particular trade or material, but today it is an invention – the government grants deeds to encourage people to disclose knowledge of new and useful inventions to the public; however, a tension exists between private property (patents, copyrights, trade secrets, etc.) and bordering public property ("public domain"), and the government has the power to exercise a form of eminent domain over all patents: it may utilize patents without the consent of the owner, but must pay a fair royalty

§3.05 Current Patent Law and Procedures

- Sources of patent law: Most of patent law is located in U.S. Code Title 35, which defines what is patentable and what services the government must provide – also relevant is Code of Federal Regulations (CFR) Title 37, which describes the internal workings of the U.S. Patent & Trademark Office as a federal agency; these rules are procedural and do not have the force of law, but knowledge of the USPTO's working policies can be useful for working through the system – also relevant is the Manual of Patent Examining Procedure (MPEP), which describes itself as "a reference work on the practices and procedures relative to the prosecution of patent applications before the Patent and Trademark Office" (i.e., a plan-language description of patent law for patent examiners)
- Patent organizations: The U.S. Patent & Trademark Office is the centralized federal agency for granting U.S. patents – it comprises a set of buildings in Washington, DC that house patent records, the offices of patent examiners, and the libraries of technical documents that serve as primary search resource for examiners – the USPTO is managed by the Commissioner of Patents and Trademarks, a political appointee who reports to the Secretary of Commerce; many cases involving important issues of patent law are filed nominally against

the presiding Commissioner by the patentee (e.g., Commissioners Robert Gottschalk, Lutrelle Parker, and Sidney Diamond)

§3.06 Patent Court System

- Patent courts: Patents are exclusively issues of federal law, so federal courts have exclusive jurisdiction over most patent suits – patent litigation typically involves one of three issues: patentability of an invention, priority where multiple parties claim the same invention, and infringement – patentability and priority disputes are first adjudicated by the Board of Patent Appeals and Interferences, which is an administrative judicial system staffed by former/senior patent examiners – appeals from decisions by the Board, and the first trials in infringement suits, are handled by the federal courts – the primary federal court is the Court of Appeals for the Federal Circuit (CAFC), a centralized appellate court staffed (ostensibly) by patent experts; this was an improvement over the old system where any of the appellate cts could hear such appeals, which created a great deal of "forum shopping" and patent law inconsistency – of course, CAFC decisions can be appealed to the U.S. Supreme Court, which has final authority (but rarely exercises it)

§3.07 State Court and Patent Ownership Issues

- State court patent cases: The Constitution vests exclusive authority in the federal government to issue patents – however, state courts can hear cases where patents are treated as property, e.g., contractual disputes over patent ownership

§3.08 Particulars of a Patent

- Contents of a patent: The cover of a patent identifies the title of the patent, its serial number and filing date, and the names of the inventors – the patent also lists the name of the assignee identified at the time of filing (subsequent assignees can be recorded afterwards, but are not part of the patent) – the patent also lists the serial numbers of the patent application from which it issued and any prior applications on which the patent relies for priority – patents are categorized by the USPTO by subject matter, the patent lists the patent classification numbers; also, the "search fields" referenced by the primary examiner are indicated – the patent also includes an abstract, drawings, a lengthy description, and set of "claims"

Chapter Four: What Is Patentable Subject Matter

§4.01 Origins of §101 Requirement

- Patentable subject matter: "Inventions" like the discovery of gravity cannot be patented – during the early history of software patents, patent examiners often rejected patents for software under the same rationale – bright lines must be drawn because patents always have an upside (promoting innovation) and a downside (granting innovation-stifling monopolies) – in the U.S. patent system, this bright-line rule is formulated at 35 USC §101: "Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or nay new and useful improvement thereof, may obtain a patent therefore, subject to the conditions and requirements of this title" – contrast with EPO convention: "European patents shall be granted for any inventions which are susceptible of industrial application, which are new and which involve an inventive step"; this convention then lists exceptions, in the form of inventions that are not patentable

("discoveries, scientific theories, and mathematical methods; aesthetic creations; rules and methods for performing metal acts, playing games, or doing business; computer programs; and presentations of information")

§4.02 The Supreme Court Shapes Computer Software Patent Law

- *Nielson v. Harford* (1841): In this early English case relevant to the patentability of software, the inventor disclosed a blower-driven furnace and then claimed "a blast or current of air propelled by bellows or other blowing apparatus, passed into an air-vessel or receptacle into the fire" – an accused infringer claimed that a patent invalidly claimed a "principle" instead of an invention – the English ct upheld the patent, but acknowledged that the breadth of this claim might raise invalidating overbreadth problems
- *Le Roy v. Tatham* (1852): This case involved a patent for a method of making pipe by making it out of lead; this was based on a discovery of a melting property of lead – the Supreme Ct held that this was a natural quality that was not amenable to patent protection: "a principle, in the abstract, is a fundamental truth; an original cause; a motive" – a process can rely on this discovery, but must be enablingly described, not merely suggested
- *O'Relly v. Morse* (1854): Samuel Morse filed for broad patent protection on "the use of the motive power of the electric or galvanic current, which I call electromagnetism, however deployed, for marking or printing intelligible characters, signs or letters, at any distances" – in essence, Morse sought to patent any use of electromagnetism for communication – the Supreme Ct held this broad claim invalid, citing *Nielson v. Harford* as grounds for rejecting any patent on a principle – the Court thus invalidated this claim as covering more than Morse had invented: "if this claim can be maintained, it matters not by what process or machinery the result is accomplished; some future inventor, in the onward march of science, may discover by means of the electric or galvanic current, without using any part of the process or combination set forth in the plaintiff's specification"
- *Rubber Tip Pencil Co. v. Howard* (1874): This case involved a patent for an eraser mounted on the end of a pencil – however, the patentee merely suggested the mounting of any rubber (or equivalent) eraser on a pencil, yet claimed "a new article of manufacture, an elastic, erasive pencil head, made substantially as in the manner described" – (of course, the USPTO uniformly disallows omnibus claims like this today) – the Court invalidated his patent, noting that other pencil-mark erasers could be mounted on the ends of pencils: "the idea of this patentee was a good one, but his device to give it effect, though useful, was not new"
- *Cochrane v. Deener* (1877): This case involved a patent for an improved process for making flour, without describing any particular machine that could be used to apply it – the Supreme Ct upheld the patent over the accused infringer's objection: "a process smay be patentable, irrespective of the particular form of the instrumentality's [sic] used"
- *Tilghman v. Proctor* (1881): This case involved a method of preparing margarine fat by using hot, pressurized water, which the inventor patented as both a process and a machine that applied this concept – the accused infringer's machine didn't resemble the claimed apparatus, but did use hot, pressurized water (though cooler

than recommended by the patent) – the infringer characterized the patent as claiming any use of hot, pressurized water to process margarine fat – the Supreme Ct upheld the patent, holding that the claim was properly limited to a patentable process first created by the inventor, and not "every mode of accomplishing this result" – thus, it's good patent drafting practice to demonstrate that a novel step does not claim any method of performing the step, and to note that prior art methods accomplish the same result with a different step based on the same principle

- *Expanded Metal Co. v. Bradford* (1909): This case involved a patent for a method of making expandable sheet metal screens – appellate cts had conflictingly ruled valid and invalid, differing on the question of whether mechanical processes were patentable – in upholding the patent, the Supreme Ct expansively interpreted the term process: "A machine is a thing. A process is an act, or a mode of acting. The one is visible to the eye, an object of perpetual observation. The other is a conception of the mind, seen only by its effects when being executed or performed. Either may be the means of producing a useful result."
- *MacKay Radio & Telegraph Co. v. Radio Corp. of America* (1939): This case involved a V-shaped radio antenna design, with the engineering details resolved according to a well-known mathematical formula – the Supreme Ct upheld the patent over an invalidity challenge: "whether a scientific truth, or the mathematical expression of it, is not patentable invention, a novel and useful structure created with the aid of knowledge of scientic [sic] truth may be" – the Ct affirmed that abstract mathematical principles were not broadly patentable, but held that a specific, functional application of such a principle was potentially patentable
- *Funk Bros. v. Kalo Co.* (1948): This case involved a patent application claiming the process of improving the nitrogen-fixing ability of legumes by introducing strains of root-nodule bacteria; while this principle had been known, the applicant suggested a particular set of strains that could cohabitate – the Supreme Ct held this patent invalid, regarding it as a known scientific principle coupled with "an advance in the packaging of the inoculates" that "produces no bacteria, no change in the six species of bacteria, and no enlargement of the range of their utility… each species has the same effect it always had"
- *Gottschalk v. Benson* (1972): This landmark case is the first Supreme Ct decision directly involving the patentability of an algorithm – the inventor patented a method of transforming binary-coded decimal numbers into binary, claiming it as a series of mathematical steps (separating the binary-coded decimal digits, adding and shifting each part in certain ways, etc.) – the Supreme Ct held the patent invalid – first, the Ct framed the issue as "whether the method described and claimed is a 'process' within the meaning of the Patent Act," referencing the distinction drawn in earlier cases between processes and principles – the heart of the majority's reasoning: It is 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. The mathematical formula involved here has no substantial practical application except in connection with a digital computer, which means that if the judgment below

[upholding patentability] is affirmed, the patent would wholly pre-empt the mathematical formula and in practical effect would be a patent on the algorithm itself" – the Ct concluded by refraining to rule on the broad issue of the patentability of software, but noted that the Report of the Presidential Commission rejected a proposal to extend patentability to software – nevertheless, many interpreted *Benson* as establishing that software was categorically unpatentable, where it simply concluded that Benson's process was abstract, and so rejected a patent on it in any form (software or otherwise)

- *Parker v. Flook* (1978): This case involved a method of controlling the progress of a catalytic process by monitoring certain parameters and updating alarm limits based on mathematical calculations – Flook claimed this invention in this context, but described it in the claim as a set of mathematical steps, which could easily be implemented either by a computer or by an operator with pencil and paper – the Court of Customs and Patent Appeals upheld this patent, noting that, unlike in *Gottschalk v. Benson*, this patent was limited to a specific context and did not preempt all uses of the algorithm – however, the Supreme Ct rejected this patent application on the basis of unpatentability – the Ct seized on the fact that the only novel feature of the invention was the application of this new mathematical process – thus, the Ct reached a verdict of §101 unpatentability, "not because it contains a mathematical algorithm as one component, but because once that algorithm is assumed to be within the prior art, the application, considered as a whole, contains no patentable invention" – Flook argued that the solution involved significant "post-solution activity," but the Ct rejected this: "the notion that post-solution activity, no matter how conventional or obvious in itself, can transform an unpatentable principle into a patentable process exalts form over substance" – while the Ct warned that its holding did not assert that patent protection of software would not promote science and useful arts, it did hold that "an improved method of calculation, even when tied to a specific end use, is unpatentable subject matter" (but "calculation" does not encompass anything that a computer does, but only mathematical calculations embodying principles of nature) – the key problem with Flook's claim was that its end product was an arbitrary number, which could be used by other processes; hence, for algorithms, it is better practice not to focus on the output data, but to conclude with a mention of how it is to be used
- *Diamond v. Chakrabarty* (1980): This case involved a patent for a new species of Pseudomonas capable of digesting industrial oil spills – the examiner allowed claims to a method of making the bacteria and as a solution of inoculum and bacteria, but rejected a claim for the bacterium as a new composition of matter, noting that it is a "product of nature" and an unpatentable "living thing" – the Supreme Ct ruled in favor of patentability, particularly (and famously) citing the legislative history of the 1952 Patent Act, which indicated that "under section 101 a person may have invented a machine or a manufacture, which may include anything under the sun that is made by man" (authored by P.J. Federico, the principal drafter of the Act) – thus, the Ct affirmed the patentability of "a non-naturally-occurring manufacture or composition of matter – a product of human ingenuity," so long as it comprised more than a (still unpatentable) law of nature,

physical phenomenon, or abstract idea – while not directly involving software, this case represents the opposite side of the "laws of nature" rules deriving from *Morse* and *Nielson*; as such, it is misunderstood as overruling *Gottschalk v. Benson* and *Parker v. Flook*, while it really just defines the same bright line by the material on the other side – *Parker v. Flook* urged judicial caution in "extending patent rights into areas wholly unforeseen by Congress," but the Ct here acknowledged this caution but did not interpret it as a ban on such considerations

- *Diamond v. Diehr* (1981): This is the first Supreme Ct case that squarely considered and affirmed patentability of a pure software algorithm – this case involved a method of curing synthetic rubber comprising an algorithm that accepted sensor input, ran them through a well-known physics formula known as the Arrhenius equation, and provided output used to operate the curing molds – the patent specifically claimed the use of a general-purpose computer to apply the Arrhenius equation to this context, and separately claimed the invention as the process of applying the Arrhenius equation in this context (without the use of a computer) – in affirming the patentability of this algorithm, the Supreme Ct reviewed *Chakrabarty* and its limitations (again, "laws of nature, natural phenomena, and abstract ideas") and found that the patentee sought "only to foreclose others from the use of that equation in conjunction with all of the other steps in their claimed process" – the Ct then distinguished *Gottschalk v. Benson* and *Parker v. Flook* as regarding unpatentable subject matter – this was a 5-4 decision with a deep dissenting split; one key difference: the majority opinion interprets the patent by focusing on the claim language (a method of continuously monitoring a mold, and eventually opening it), while the minority interprets the patent according to its inventive concept (using the Arrhenius equation to determine how long the rubber should cure), which might look more like prior art
- *Diehr* distinguished from *Flook*: These cases regarded fairly similar inventions, and only three years apart, but reached fundamentally different decisions (while reciting the same bright-line test) – in defense of the differing position, key differences do exist between *Flook* and *Diehr*: the method claimed here ended with a state change in a physical object, which cannot be construed as insignificant "post-solution activity" – however, a deeper reason for such a dramatic shift may be a shift in IP jurisprudence: both Stevens and Blackmun joined in the majority opinions of both *Flook* and *Diehr*

§4.03 The Federal Court Reshapes Software Patent Law

- *In re Alappat* (1994): This en banc hearing of the CAFC involved a patent for a rasterizer device for improving the output of an oscilloscope – the applicant claimed this invention as "a rasterizer for converting vector list data into anti-aliased pixel illumination intensity data on a display means, comprising" the steps performed by an anti-aliasing algorithm – the examiner had rejected this patent application as attempting to claim the principle of anti-aliasing in any context, based on the claimant's use of means-plus-function language; the examiner thus accused the applicant of attempting to preempt "the whole algorithm" – the en banc CAFC upheld patentability, finding error in the examiner's reading of the claim as "reading on any and every means for performing the functions," when the claim was clearly limited to a specific method for a specific application; thus,

the CAFC rejected the argument that the applicant had claimed "a disembodied mathematical concept as a whole" – the CAFC acknowledged that the method involved the use of a general-purpose computer, but found this irrelevant to patentability – again, the dissenting opinions focused on the "inventive concept" as a whole, and accused the majority of focusing too heavily on the claim language and of being misled by the presence of unimportant structural claim limitations – the dissent argued that "the dispositive issue is whether the invention or discover for which an award of patent is sought is more than just a discovery in abstract mathematics"

- Freeman-Walter-Abele test: In a series of decisions following *Alappat* (*In re Freeman* (1978), *In re Walter* (1980), and *In re Abele* (1982)), the CAFC attempted to fashion a two-step test, instructing the USPTO to determine (1) whether a mathematical algorithm was recited directly or indirectly, and if so, (2) whether the claimed invention, as a whole, was no more than the algorithm – if both questions were answered affirmatively, then the patent should be rejected as reciting unpatentable subject matter – this test was prohibitively difficult to apply, and so has been deprecated in subsequent opinions (covered below)
- *In re Warmerdam* (1994): This case involved a patent claiming a data structure used in a collision detection algorithm – the data structure was useful in modeling the boundaries of an object as a closely-tailored series of bubbles, and in storing this model in a compact but accessible manner – the context for this invention was the movement of a robotic arm in such a manner as to avoid physical objects – the examiner rejected claims to "a method for generating a data structure" and "a data structure generated by the method" under §101 as reciting unpatentable subject matter, and also rejected a claim to "a machine having a memory which contains data representing [the bubble model]" as indefinite under §112 – the CAFC reversed the latter rejection and allowed the machine patent, but upheld the rejection of the method and "data structure" claims – in evaluating the latter claims, the CAFC expressly abandoned the *Freeman-Walter-Abele* test as unworkable, due to the inability to define the critical term "mathematical algorithm" with any precision – instead, the CAFC fell back on the "laws of nature, natural phenomena, or abstract ideas" exception, and held that the method and data structure claims comprised "abstract ideas"
- *In re Lowry* (1994): This case involved a patent for a data structure organized into a hierarchy including objects stored in a database, patented first in the abstract and second as embedded in memory – the examiner rejected all claims as unpatentable under §101 – the Board reversed the §101 rejection for claims to a memory, citing it as an article of manufacture, but rejected the claims under §102 and §103, regarding the data structure as inapplicable to a novelty analysis on the basis of the "printed matter doctrine" – the CAFC reversed the "printed matter" rejection, noting that printed matter can convey a structural limitation that makes a broader invention patentable – the CAFC did not consider the patentability of the data structure in the abstract (and probably would have rejected it as per *In re Warmerdam*), but allowed the data structure to convey a critical limitation as part of a novel memory object – perhaps the critical patentability distinction between the *Lowry* data structure and the *Warmerdam* data structure is the breadth of

applicability of the data structure: the patentable data structure is a component of human design with many preferential options, while the latter is dictated by the nature of the problem, and so is closer to a "principle of nature" than an invention

- *In re Trovato* (1994): This case involved a patent application for a pathfinding algorithm, which the applicant claimed as a method involving the building of a graph of the search space (as a data structure) followed by a recursive search through the graph, and as a machine embodying his process – the examiner rejected the claims under the *Freeman-Walter-abele* test, finding that the application tried to appropriate an algorithm, and noting the absence of data gathering and post-solution activity – the CAFC affirmed the rejection – the applicant argued that the application involved a physical problem resolved by a physical entity; the CAFC rejected this, noting that the specification recommended "no computer architecture, no circuit diagram, and no hardware at all," and in fact discussed the problem as one of mathematics – thus, "as in *Warmerdam*, Trovato's claims operate merely in the domain of abstract ideas" – later, and surprisingly, the CAFC later withdrew its opinion, vacated its judgment, and ordered the Board to reexamine the application in light of *In re Alappat* and its new guidelines on §101 patentability – a harsh dissent criticized this action as affording no guidance to the Board or the patent law community as to how to evaluate this patent application – practice tip from *In re Trovato:* it is worthwhile to describe in the specification the anticipated field of use, and preferably applying the invention to a specific (preferably physical) problem and solution
- *Ex parte Beauregard* (1993): This case tested the patentability of a CD-ROM or other computer-readable media, which had been alternatively regarded as a patentable article of manufacture or unpatentable "printed matter" – the invention at bar was a method of drawing and filling complex polygons quickly and thoroughly – the patent office had already allowed method and system claims for this process, but IBM filed a divisional application claiming this as an article of manufacture comprising a computer-readable medium – the examiner rejected the claims as printed matter, which the Board affirmed – the Board's opinion focused on the physical relationship between the hardware and the computer-readable medium, and found the latter to serve as a mere "substrate" that didn't directly generate the claimed utility – thus, "an article of manufacture in the form of printed matter on a substrate cannot be statutory if it differs from other substrates only by the informational content of printed matter, unless there is some functional interaction between the printed matter and the substrate" – the Board noted that it would be difficult to delineate a "magic dividing line" between computer-readable media containing a computer program, and the same media containing music or video – this decision was not appealed to the CAFC, but the decision is instructive as to how the Board approaches this claim style
- *State Street Bank & Trust Co. v. Signature Financial Group* (1998): This landmark case involved a method of evaluating pooled financial investments according to a "hub and spoke" method – Signature Financial Group patented this as a pure software algorithm, and as a business method; State Street filed for declaratory judgment of invalidity – the district ct summarized the issue as "whether computer software that essentially performs mathematical accounting

functions and is configured to run on a general-purpose (i.e., personal) computer is patentable under 35 USC § 101" – the district ct then applied the *Freeman-Walter-Abele* test and found the invention to recite a mathematical algorithm lacking a physical transformation, and held the invention unpatentable due to the lack of physicality (drawing a strong parallel with *In re Schrader*, which held a real estate bidding system as unpatentable) – the CAFC reversed the rejection: "Today, we hold that the transformation of data, representing discrete dollar amounts, by a machine through a series of mathematical calculations into a final share price, constitutes a practical application of a mathematical algorithm, formula, or calculation, because it produces 'a useful, concrete and tangible result' – a final share price momentarily fixed for recording and reporting purposes and even accepted and relied upon by regulatory authorities and in subsequent trades." – the majority noted that the issue should not depend on which statutory class the invention resembled, but rather "the essential characteristics of the subject matter, in particular, its practical utility" – the patentee had claimed the invention as a series of means, and in demonstrating the patentability of the invention, the CAFC rewrote the claim to recite specific hardware recommended in the specification for each means – the CAFC also affirmed the eradication of the *Freeman-Walter-Abele* test ("has little, if any, applicability to determining the presence of statutory subject matter")

- Analysis of *State Street Bank*: As in *In re Alappat*, the district ct in *State Street* looked behind the claims, concluded that the invention was a method of mathematically transforming numbers, and found it unpatentable; but the CAFC focused on the claim style to find statutorily patentable subject matter – this highlights an important difference in approaches to §101 analyses: is this resolved by focusing on the claim language or on the disclosed invention? – i.e., if someone claims a machine containing some novel code, are they attempting to patent the machine or the novel code? – in *Gottschalk*, and in the dissent in *Diamond v. Diehr*, the Supreme Ct looked to the heart of the invention for §101 determinations; but the CAFC here focused on the claim style, and has done so since – the result of *State Street* is to extend §101 patentability to business models, which may even encompass a general-purpose machine containing a software embodiment of the business method; of course, this holding eradicated the "business method" exception to §101 patent eligibility

§4.04 The Federal Circuit Judges' Software Patent Voting Record

- CAFC judge voting trends: Summary of table on page 188:
  - Strong proponent: Giles Rich authored the majority opinions in *Alappat* and *State Street Bank*, and the decision to withdraw *Trovato*, and concurred in *Lowry*
  - Proponent: Rader wrote the majority opinion in *Lowry* and concurred in *Alappat*
  - Weak proponents: Newman concurred in *Alappat* and the decision to withdraw *Trovato*; and Lourie and Bryson concurred in *State Street*
  - Weak opponents: Plager wrote the majority opinion in *Warmerdam*; and Schall and Michel concurred in *Trovato*

- Strong opponent: Nies wrote the majority opinion in *Trovato* and dissented in *Alappat* and the decision to withdraw *Trovato*

§4.05 Patentable Subject Matter Case Digest

- Case digest: This section features a list of many cases dealing with specific patent cases finding software patentable (Section A) or unpatentable (Section B) – there may be a bright line between these two kinds of inventions: is the invention subject to human control, or is it a mere truism? – mathematical inventions may constitute a new model to track the trajectory of an object, or a new precision to π – both may be useful, but only the former is patentable: the former is an object of human creation; the latter is a discovery

Chapter Five: Searching for Prior Art

§5.01 Prior Art

- Context: Since much of prosecution and infringement analysis focuses on whether the patented invention is novel and nonobvious, prior art is an important consideration – since software patents were presumed unallowable until the 1980's, very few patent application before this period were directed toward software inventions, so very little pre-1985 patent prior art in the form of U.S. patents exists – when changing case law abruptly obligated the USPTO to consider software patents, it found itself with a sparse prior art software database, so it issued a number of patents that much of the industry considered invalid – however, the USPTO database has greatly improved, and of course it is within both the duty and best interests of the patent applicant (including the patent attorney) to disclose known relevant prior art

§5.02 Patent Classification

- USPTO patent classification system: The patent office organizes its prior art records into millions of "shoes" (full-height compartmentalized shelves) – the shoes are organized based on the Patent Classification System – many such systems have been tried (including one involving a "miscellaneous" category, which unfortunately led to a growing proportion of patents being so classified); the current system involves assigning patent and patent application to one or more classes and subclasses to identify the invention (but even with this system, ambiguity creeps in) – thus, prior art references on a particular topic can be found by using the Manual of Classification to identify the subclasses of interest, and then locating the shoe containing the records for those references (this usually requires the help of USPTO employees, because shoes are often moved around to accommodate the expansion of the prior art collection)
- Principles of the current USPTO patent classification system: The current system uses a two-tier hierarchy (classes containing subclasses), based on the central utility of the invention – in fact, the subclasses define a further hierarchy: class 364 (Electronic Computers and Data Processing Systems) contains subclass 274 for artificial intelligence inventions, which further contains subclasses 274.1 (software), which contains 275.1 (software applications), which contains 275.2 (control applications), which contains 275.4 (vehicle control/navigation control applications) – the patent classification guide indicates which subclasses are subordinate to which other subclasses (the list of subclasses contains indentation

to indicate priority) – each class and subclass is assigned a description of the contained subject matter (and these descriptions often indicate inventions excluded from the subclass) – the USPTO creates new classes/subclasses by splitting a new category off from an overly large group of existing inventions, rather than prospectively creating empty classes when it anticipates a need – classes may contain a "miscellaneous" subclass; if this subclass grows too large, inventions will be redistributed into new subclasses within the class

- Principles for classifying inventions in the current USPTO patent classification system: The principal basis for classification is the utility of the invention ("the direct, proximate, or necessary function, effect, or product"), especially when caused by a single unitary act – where utility is not clear or singularly defined, mechanical and composition inventions can also be classified by characteristic structure ("tubular stock" is generally useful for many kinds of construction, so no "clear" utility; hence, physical configuration is the only differentiating feature) – pure chemical inventions are always categorized by chemical composition, regardless of utility; but mixtures are characterized based on utility – in categorizing an invention, the examiners consider the invention as claimed, not as described in the specification – when an invention belongs to two or more classes, the examiner chooses the "most comprehensive claim," i.e., the one that most closely identifies the invention (note: this is often not the broadest claim) – exceptions: (1) when the heart of the invention is a subcombination of elements in a well-known combination object, the subcombination is used as the basis for classification; (2) when a novel article is centrally defined by its material, the material is used as the basis for classification; (3) a novel process for using a composition will be classified by the composition – of course, the invention can be cross-referenced with other classes/subclasses for cross-topic inventions, or in case the patent secondarily claims (or discloses but does not claim) subject matter that should be assigned to other subclasses

§5.03 Classification of Software Patents

- Software patent classification: Initially, all software patents were relegated to class 364 ("Electrical Computers and Data Processing"); subclasses 100 and 400-800 were heavily subdivided, so most software patents were placed in subclasses 200, 300, and 900, which grew unchecked – instead of further subdividing these classes, the patent office assigned keywords to certain subclasses: 223.1 was associated with the "aeronautics/space" keywords; thus, patents were not re-classified (and the shoes for these broad categories were not reorganized), but were merely tagged into "pseudosubclasses" – also, the USPTO published "digests" or "art collections" listing all of the patents in a particular field – in November 1991, the magnitude of the problem compelled the USPTO to transfer all patents from 200, 300, and 900 into subclasses of new class 395 ("Information Processing System Organization"), with major groups of subclasses dedicated to display processing and artificial intelligence – the process of further subdividing class 395 has been incessant, and will probably remain continuous in order to keep up with the rate of issuing software patents

§5.04 Searching CASSIS

- CASSIS: This is the USPTO's CD-ROM-based patent collection, which can be purchased from the USPTO (with subscriptions for further offerings) or accessed via the Internet – note: most of the information in this section pertains to the 1994 edition of CASSIS, and so is outdated and hence omitted

§5.05 Searching for Prior Art

- Prior art sources: CASSIS (or the current USPTO analogue thereof) is a good starting point – LEXIS offers an excellent resource called LEXPAT under a few kinds of billing schemes

§5.06 Basics of Internet Searching

- Internet searching: the Internet provides a number of resources (this text explains the basics of the Internet, e.g., formatting of URLs and addresses of popular search engines)

§5.07 Software Patent Institute

- Software Patent Institute: This nonprofit group maintains a database of software prior art, and is primarily aimed at serving the needs of the USPTO – the database accepts submissions of software prior art for a nominal per-item fee, and will provide a public bulletin of the submitted invention; this is intended to serve as a defensive publication service

§5.08 DIALOG Information Services

- DIALOG: As of the printing of this text, DIALOG was one of the most comprehensive patent and trademark databases – note: again, most of the information in this section pertains to the basics of DIALOG searching, which are now outdated and hence omitted

§5.09 Searching CD-ROM Resources

- CD-ROM resources: Compilations of prior art are available from the USPTO, from the Software Patent Institute, Dr. Dobb's Journal, and the Microsoft Developer Network

Chapter Six: Software Patent Specification

§6.01 Legal Requirements of the Specification

- Requirements of specification under U.S .Constitution: According to patent policy, the purpose of the specification is to teach the invention to the public as the *quid pro quo* for the patent monopoly grant – this requirement implicitly derives from the U.S. Constitution, Article I, Section 8, Clause 8
- Requirements of specification under 35 USC §112: "The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most clearly connected, to make and use the same, and shall set forth the best mode contemplated by the inventor of carrying out his invention" – thus, three distinct requirements: (1) written description, (2) enablement, and (3) best mode – failure to satisfy all three requirements results in an invalid patent, so the specification should be drafted carefully
- Requirements of specification under 37 CFR §1.71: This section repeats the written description, enablement, and best mode requirements, and also requires the specification to "set forth the precise invention for which a patent is solicited,

in such manner as to distinguish it from other inventions and from what is old"; and also to "describe completely a specific embodiment of the process, machine, manufacture, composition of matter, or improvement invented"; and also to "explain the mode of operation or principle whenever applicable"; and for improvement inventions, the specification must "particularly point out the part or parts of the process, machine, manufacture, or composition of matter to which the improvement relates," and should be "confined to the specific improvement and to such parts as necessarily cooperate with it or as may be necessary to a complete understanding or description of it" – thus, new requirements: (1) description of precise invention, (2) distinguishment from the prior art, (3) specific embodiment (usually merges with "best mode" requirement), (4) mode of operation or operative principle, and (5) limited focus for improvement inventions

- Requirements of specification under common law: As applicators of the law, judges must adapt these disparate rules to the sufficiency of the patent or patent application at bar – thus, cases might refine the foregoing rules, but are not very consistent or reliable
- Written description requirement: The purpose of this section is to evidence the applicant's complete possession of the invention at the time of filing – this can arise as an issue if subsequently added claims are not fully supported by the specification as filed – this requirement stems from the former practice of relying on the specification to determine the metes and bounds of the patent (which is now fulfilled by the claims), but today serves as a check against inventor "overreaching" – this requirement is parallel with the enablement requirement, but is not redundant: "it is possible for a specification to *enable* the practice of an invention as broadly as it is claimed, and still not *describe* that invention" (*In re DiLeone* (1971))
- Enablement requirement: This requirement ensures that the patent teaches the method of using the invention to the public, and must be enabling at the time of filing; i.e., the invention cannot be contingent on future advances in the field (though it can describe components that the expert would be technically capable of providing at the time of the invention) – the sufficiency is judged in the context of "those of ordinary skill in the art," and is sufficient if the invention can be used "without undue experimentation" – the level of skill depends on the sophistication of the field to which the invention pertains, and the thoroughness of the teaching is directly related to the scope of the claims, and inversely related to the prior art crowding of the field – relevant factors (from *In re Wands* (1988)): quantity of experimentation needed to practice the invention; amount of direction or guidance presented; presence or absence of working examples; nature of the invention; state of the prior art; relative skill of those in the art; predictability or unpredictability in the art; and breadth of claims – the CAFC has dubbed this a "not merely qualitative" test, i.e., a factual inquiry involving many factors with no bright lines – e.g.: "a considerable amount of experimentation is permissible, if it is merely routine, or if the specification in question provides a reasonable amount of guidance with respect to the direction in which the experimentation should proceed" – a small amount of experimentation can be sufficient (*Hirschfeld v. Banner* (1978)) – trial evaluation may include the process of giving a skilled

programmer the specification and asking him to implement it; interestingly, debugging or flawless execution is not required or relevant to this inquiry (*Hirschfield v. Banner* (1978))

- Best mode requirement: This requirement ensures that the patent precludes inventors from withholding some valuable information from the public – the evaluation of this test involves two steps: a subjective measure into what the inventor actually contemplated was the "best mode," and an objective measure of whether or not he adequately disclosed it – the critical issue is concealment: did the inventor hide what he contemplated to be the best mode? – of course, it's easy to avoid intentional concealment: merely ask the inventor how the invention is best used, and then disclose that method in the specification – more difficult to avoid is accidental concealment – this may arise from a specification so inadequate as to fail this test; in this circumstance, the same factors used to test enablement are relevant to concealment – accidental concealment can also arise where the patent application is filed long after disclosure, and where the inventors have concocted a better mode in the interim (note that the relevant time period is "at the time of filing"); to avoid this, the inventors should be consulted about best mode right before the application is filed

§6.02 Mechanics of Drafting the Specification

- Parts of specification: As dictated by 37 CFR §1.77 and MPEP §608, a patent application should include the following parts in logical order:
  - Title of invention
  - Cross-reference to related applications, if any
  - Cross-reference to microfiche appendix, if any
  - Brief summary of the invention
  - Brief description of the drawings
  - Detailed description
  - Claims
  - Abstract of the disclosure
  - Oath or declaration
  - Drawings
- Specification styles: A patent may be read and used by a variety of audiences (examiners, judges and juries, investors, licensees, business managers, and engineers) and for a variety of purposes; the specification should serve all of their needs – examiners need to understand the invention and its novelty; since examiners are pressed for time, a clear and well-organized specification will facilitate examination and speed the application to issuance – diagrams can be used to help convey a complicated software process to the examiner (and others); moreover, a multi-element claim can be supported with a drawing showing the different claimed parts and their interaction – businesspeople will focus on the title, abstract, drawings, and first few paragraphs of the specification; if the invention is to be licensed or used to attract investors, the beginning of the specification should be written to advertise the invention (don't do so in the abstract, which is intended to be a search guide) – finally, judges and juries will read the patent in order to test validity and infringement, so the specification should explain the invention in layman's terminology

- Defining and describing the software invention: Before drafting anything, the patent attorney should have a full understanding of the invention – it may be difficult to get the inventor to focus on the invention, rather than their embodiment; it may help to explain the purpose and structure of a patent application to the inventor, and to draft the broadest claims in the presence of the inventor – for drafting, it is important to make the specification enabling, but not necessary to feature actual source code (and this may not even help explain the invention well) – it may be clearer to teach the invention in a bottom-up manner (describing basic components and their synthesis) or a top-down manner (describe the overall task and then how each step is accomplished)
- Attorney-client privilege: The majority legal view construes discussions with inventors as covered by the attorney-client privilege, since this closely resembles client counseling in other areas of law – however, some minority views portray the patent attorney as an agent of the USPTO, so when the inventor gives the patent attorney information to be disclosed to the USPTO, that information is not covered by the attorney-client privilege (*W.R. Grace & Co. v. Viskase Corp.* (1991)) – case law is somewhat inconsistent on this point, with many fact-specific inquiries spawning a host of conflicting rules; generally, anything submitted to a patent office (USPTO or foreign) is not covered, nor is technical information conveyed to an attorney for the sole purpose of incorporation into a patent application (i.e., where the technical information does not affect any legal opinions) – thus, it is helpful to clarify all work-product notes, including client letters, as pertaining to the legal task of drafting a patent application
- Functional components of software: Software may be envisioned as a collection of data structures, data processors (manipulation algorithms), and interfaces (transformation algorithms) – this abstraction defines a different view of software, and may not match the source code implementation; moreover, identifying these components may suggest the elements of a claim – common data structures include arrays, linked lists, stacks, queues, and binary trees; see Don Knuth's *The Art of Computer Programming* for more info on these – interfaces may be between data processing algorithms, between a program and the user, between two programs that will share data, between software and a hardware device, or between software and the operating system

Chapter Seven: Software Patent Drawings

§7.01 Drawings in a Patent Application

- Purpose of drawings: For software patents, drawings may explain the invention in plain terms to a judge, jury, examiner, and potential licensee or infringer – thus, the inclusion of drawings may help secure the patent more efficiently, may illustrate its value more clearly, and may facilitate enforcement

§7.02 Legal Requirements for Drawings

- Legal requirement of drawings: 35 USC §113: "The applicant shall furnish a drawing where necessary for the understanding of the subject matter sought to be patented. When the nature of such subject matter admits of illustration by a drawing and the applicant has not furnished such a drawing, the Commissioner may require its submission within a time period of not less than two months from

the sending of a notice thereof. Drawings submitted after the filing date of the application may not be used (i) to overcome any insufficiency of the specification due to lack of an enabling disclosure or otherwise inadequate disclosure therein, or (ii) to supplement the original disclosure thereof for the purposes of interpretation of the scope of the claims." – thus, where necessary, the drawings should be filed with the application – see also 37 CFR §1.81: the application may be submitted with sketch drawings that don't conform to the USPTO rules for figures, but compliant drawings must be filed before the patent can issue – the primary case for defining "when" drawings are "necessary": *In re Hayes Microcomputer Products, Inc.* (1992): even a drawing that depicts a blank square labeled with the device comprising the "heart" of the invention can be sufficient, if the specification and drawing taken together sufficiently teach the invention – this conclusion was bolstered by expert testimony of disclosure sufficiency – sometimes, drawings are not required; most chemical patents fully teach the invention in the specification, and so feature no drawings – the USPTO's ruling on whether or not drawings are "required" is given considerable deference by the CAFC, and in fact the USPTO will not register a filing date for the application if drawings are required but not included with the application as filed – e.g., *Jack Winter, Inc. v. Koratron Co.* (1974): the patentee's failure to file drawings "required" of the patent application caused a delay in receiving a filing date, which triggered an "on sale" bar that invalidated the patent; the patentee argued at trial that the USPTO had erred in "requiring" the drawings, but the CAFC refused to consider this argument, deferring to the USPTO's judgment – fortunately, such a scenario is unlikely to arise today: the application receiving branch of the USPTO makes most such judgments almost immediately after the application is filed, and an opinion by an examiner (at a much later date) is usually considered a request for "supplementary" drawings (second sentence of 35 USC §113), and does not affect the filing date

- Content of drawings: 37 CFR §1.83 indicates that every claim element should be illustrated in at least one drawing of the invention, but non-essential elements can be shown with little detail – improvements of larger devices should show the improvement both within the larger device and in isolation – as for the level of detail: "The question is not whether one skilled in the art can decipher the invention, but whether the drawing is so clearly and artistically executed as to facilitate a ready understanding of the invention both at the time of examination and in searches afterward in which reference to the patent is made" (*Ex parte Good* (1911))

§7.03 Practical Considerations Prior to Litigation

- Litigation considerations: Good drawings may form the most persuasive exhibits in patent litigation – drawings should "tell the story" of how the invention works, and how it was invented – interference proceedings require the parties to submit drawings to the Board in a sealed envelope, each with an indication of the date on which they were made, as part of the preliminary statement; thus, it is critical to preserve drawings and to establish the earliest possible dates of invention, including sketches or first drawings; inventors should be encouraged to date and sign such sketches throughout the development process

- Patent Office *Official Gazette*: The USPTO publishes this guide every Tuesday with all of the patents issuing on that day – the patentee of each such patent is required to designate one view for publication in the Gazette; this should be a drawing that representatively portrays the whole invention, but does not overwhelm the viewer with detail – the drawing may be organized either to outline the specification (top-down approach), or to illustrate a key claim (bottom-up approach); in fact, sketching out a drawing of the invention may be a useful technique for drafting the claims

§7.04 Flowcharts

- Flowchart drawings: These drawings are classic methods of illustrating software, and are even recommended by 37 USC §1.81(b); however, flowcharts can more easily illustrate simple, procedural algorithms than very complex ones or asynchronous, object-oriented algorithms – ISO 5807 defines standards for flowchart illustrations, including a set of rules for program operation; rectangular boxes represent process steps, while diamonds represent decision points – a parallelograms represent data, while cylinders represent data stores (databases, files) – many kinds of flowcharts are available:
  - Program flowcharts (e.g., p. 307) show the steps in a simple, primarily linear algorithm
  - Data flowcharts (e.g., p. 308) show data moving through a set of methods as a digital workpiece
  - System flowcharts (e.g., p. 309) show the functional components of a system, with a focus on how the process is flowing
  - Network charts (e.g., p. 311) also show the functional components of a system, but focus on the interfaces between them (i.e., while a system flowchart may have to show an element several times to illustrate its involvement at several stages of a process, a network chart would only show it once, with many connections)
  - System resource charts (e.g., p. 312) show processing units, what kinds of data they exchange, and control relationships; this is more of a top-level abstraction than the more process-oriented kinds of flowcharts

§7.05 Pseudocode

- Pseudocode defined: Pseudocode is a source-code-like illustration of steps, focusing more on English descriptions of steps than adhering to the syntactic rules of a programming language – however, this is a rather poor drawing technique; perhaps should be limited to illustrating a preferred embodiment or best mode than as a description of "the invention" – pseudocode less than ten pages can be included in the specification as a table (should be at the end of the description, but before the claims), or as a set of drawings (each one must conform with the requirements of 37 CFR §1.52, e.g., must include figure labels) – pseudocode longer than ten pages must be submitted as a sequence listing, either on paper or microfiche (37 CFR §1.58)

§7.06 Entity-Relationship Diagrams

- ER diagrams: These drawings are particularly helpful for illustrating relational database schema – these diagrams show data as a series of tables, and show relationships as arrows connecting the tables, with a "cardinality" to describe each

end of the relationship (one-to-one, one-to-many, or many-to-many) – see example on page 318

§7.07 Booch Notation

- Booch notation: This set of diagram types focuses on programming constructs that reflect the mechanical operations of software – several kinds of Booch notation diagrams are available:
  - Booch class diagrams illustrate the interfaces between classes, reflecting four types of relationships: association (a general cooperative relationship), inheritance (an extension of a class), encapsulation (one class embodying another as a member), and reliance (a server/client type of relationship)
  - Booch object diagrams illustrate the interaction of class instances
  - Booch module diagrams show the locations of class and object definitions within source code
  - Booch process diagrams show the collaboration of threads and processes in a multithreaded/multi-process algorithm

§7.08 Use of Object-Oriented Notation

- Object-oriented notation: The purpose for using any kind of object-oriented diagram style is to illustrate the architectural design of a system – this is much more pragmatic and clearer than attempting to depict an object-oriented or asynchronous algorithm with a poorly fitting flowchart

§7.09 Data Flow Diagram

- Data flow diagrams: Data flow diagrams model operate as flowcharts of data between components – two kinds are in common use: Gane/Sarson notation and Yourdon/DeMarch notation – these are very similar models (compare charts on pages 326 and 327), but a few design philosophy disparities lead to different kinds of diagrams – Gane/Sarson models encourage showing as much detail as possible, while Yourdon/DeMarch models show at most seven elements, with more detail added through other diagrams focusing on a component of a higher diagram – Gane/Sarson modeling suggests starting with an existing object, and then focusing on the steps necessary to improve it; Yourdon/DeMarch modeling suggests starting with the improvement, and then putting it in the context of the prior art (hence, Gane/Sarson models are clearer for patent analysis purposes)

§7.10 Representing Data Structures

- Data structure representation: Gane/Sarson models depict data as a hierarchical list of elements, with levels denoted by indentation – Yourdon/DeMarco lists data structures "in-line" as a sum of elements

§7.11 Checklist

§7.12 Unified Modeling Language

- UML: The Unified Modeling Language was devised as a general-purpose way of describing all software – many kinds of UML diagrams are available:
  - UML use case diagrams (e.g., p. 333): These diagrams focus on actors (both human and software), and show their typical functions and relationships
  - UML class diagrams (e.g., p. 335): These diagrams, very similar to Booch notation class diagrams or ER diagrams, show classes and their relationships – again, these relationships may be in the form of "associations" (cooperative interworkings between class instances), "generalizations" (inheritance), and

"aggregation" relationships (encapsulation) – each class is shown as a box with three sections (class name, variables, and methods), and arrows with cardinality to show relationships

- UML interaction diagrams: These diagrams show a specific (exemplary) use case, where one class instance cooperates with other instances to achieve a result; this diagram shows exactly how they operate and what kinds of data they exchange – UML sequence interaction diagram (e.g., p. 337) shows a sequential timeline of messages and data passed between objects, including the lifespan of each class instance – UML collaboration interaction diagram (e.g., p. 338) shows the interconnections between objects, with sequences peripherally depicted by sequentially numbering the interactions
- UML package diagrams (e.g., p. 340, top): These diagrams show the functional components of a software invention, which resembles the structure of elements in a typical software patent claim – the components are shown as units, with connections representing unit dependencies – clusters of units that work together as a subcombination may be shown as being housed in a file folder
- UML state diagrams (e.g., p. 340, bottom): These diagrams are very similar to flowcharts; they show the state sequence of a process, along with the interconnection of states – also shown are the features associated with each state; these are described in the format "Event / Action," e.g., "test a condition / write result to file" – also shown are the preconditions for transitioning from one state to the other ("guards"), in "[name]" format – finally, this style allows the illustration of "superclasses" as sets of related classes
- UML activity diagrams (e.g., p. 342): These diagrams are like process flowcharts, but are better at illustrating parallel activities – separate processes can be illustrated as operating separately, and can be shown to cooperate by reaching a "synchronization bar," whereby all must be ready to move on – again, preconditions ("guards") are denoted by "[name]" format – for complex diagrams, it may be helpful to arrange activities into vertical "swimlanes"
- UML deployment diagrams (e.g., p. 344): These diagrams show the physical relationships between hardware and software components – the main components are all hardware boxes, shown as containing different kinds of software – hardware connections (e.g., network connections) are shown as solid lines labeled with communications protocols ("TCP/IP"); interoperating software packages are shown with dotted line connectors – these kinds of diagrams are very helpful for showing software operating with hardware in the physical world, which may help in §101 determinations

Chapter Eight: Drafting Claims of Proper Scope

§8.01 Legal Requirements

- 35 USC §101: Every patent claim must begin by reciting one of the four statutory classes ("process" or "method"; "machine" or "system" or "device"; "manufacture" or "article"; or "composition of matter") – *Parker v. Flook* was interpreted as limiting "methods" to exclude "methods of mathematical calculation," but a better reading is that "methods" that are only "expressions of

laws of nature" are excluded, whereas methods (even purely mathematical ones) that achieve some useful and applied result are patentable – e.g., *Gottschalk v. Benson* (1972) held unpatentable an abstract method of numeric conversion without a useful application, but *State Street Bank & Trust v. Signature Financial Group* (1998) held patentable a mathematical method of computing funds in a mutual pool as a method of tracking pooled investment contributions – this is usually described as the difference between "discovery" and "invention"

- Classes of inventions categorically excluded from §101: Several classes of "inventions" are categorically rejected under 35 USC §101 as nonstatutory, and also as not "useful," as required by this section ("any new and useful process,…") – this list includes "printed matter" (the USPTO regards this as mere information; the applicant is encouraged instead to focus on the apparatus or method for generating the printed matter), "naturally occurring articles" (the USPTO regards this as neither a novel "manufacture" nor a novel "composition of matter," since it existed before the "invention"; e.g., *American Food Growers v. Brogdex* (1930): shrimp prepared for eating are unpatentable; the applicant is encouraged to focus on the novel method or machine that produces the article), and "scientific principles" (e.g., *O'Reilly v. Morse* (1853): "the use of the motive power of the electric or galvanic current, however developed, for making or printing intelligible characters, letters, or signs at a distance" is an unpatentable "scientific principle") – this list used to include "methods of doing business," but this exception has been eradicated (see *State Street Bank & Trust Co. v. Signature Financial Group* (1998))
- 35 USC §112: This section requires that patent applications must be "enabling," i.e., the specification must teach how to make and use the claimed invention to achieve a useful purpose – most rejections citing §101 for lack of utility will also cite §112 for non-enablement – §112 is also cited for the rejection of a host of unpatentable claim styles:
  - "Omnibus claims," e.g., those claiming "a device substantially as shown and described," and "single means claims," where the invention contains exactly one limitation in means form: both styles fail to "particularly point out and distinctly claim" the invention as required by 35 USC §112 ¶2
  - Unsupported functional language, where the applicant indicates that a component performs a certain function or has a certain quality but does not specify how (e.g., "a woolen cloth having a tendency to wear rough rather than smooth" (*In re Fuller* (1929)) – functional language can still be used as part of a "whereby" clause, but this is treated as a non-distinguishing recitation; it is only given weight if it "breathes life and meaning into the claim," and is considered of no weight if it merely describes the necessary result of the foregoing claim language
  - Unduly broad claims, where the claimed invention exceeds the predictable results of the invention as described – the amount of evidence needed to demonstrate "predictable" results varies with the predictability of the field: mechanical and electrical arts are more reliable than chemical and biological arts, and hence the latter have heightened standards – such evidence can take the form of experimental data, a range of alternative embodiments, etc. –

fortunately, software is usually considered a highly predictable science, so broad claims might be supported by a single embodiment
    - Claims that are vague, indefinite, incomplete, prolix, or a mere aggregation of elements without cooperation are similarly rejected under 35 USC §112 as non-enabling (see MPEP §706.03(d) through (j))
    - "Double patenting" claims or unduly duplicative claims are rejected under this section (see MPEP §706.03(k) and (l) and *Ex parte Whitelaw*)

§8.02 Claim-Drafting Process

- Claim drafting: Most patent attorneys begin the patent drafting process with the claims, which will define the metes and bounds of the invention, and hence its scope and position within the context of the prior art – some suggest that drafting the description helps prepare the context for the claims, but this also raises the potential need to revise the description if the claims are not drafted as expected – it may be helpful to sketch out the claims first, perhaps even as a series of drawings of the components, in order to isolate the invention as a minimal set of components

§8.03 Finding the Invention

- Finding the invention: Of course, drafting a patent application requires the applicant to describe the metes and bounds of the invention – different business models will be more or less facilitated by protecting different aspects of the same invention, so first strive to understand the invention in its preferred embodiment – next, identify claim elements essential to the invention – next, draft claim language for each element – finally, add connective language to join the elements into a claimed invention
- Aides to patent claim drafting: Drawing diagrams may help put the invention in context – the inventor's drawings may be particularly useful, and for software patents, the source code may suggest an approach for identifying the elements and organizing them into components – the set of claims should include broad, intermediate, and narrow claims, which broadens the arsenal of weapons that can be used to ensnare an infringer (the most useful claim is the one that claims the invention just broadly enough to ensnare a competitor, but not any more broadly, as this might invite an invalidity attack) – when the claims are fully drafted, it is important to test them by applying them to the claimed invention, by trying to design around them with various modifications, and by applying them to the prior art to ensure that they recite novel and non-obvious subject matter – problems discovered through claim testing are helpful for refining the claims into a more easily prosecuted and allowable set

§8.04 Claim-Drafting Templates

- Claim templates: It is not a good idea to begin drafting claims by adapting the claim set of a close invention, as this obviously invites a §103 rejection – however, it's helpful to consider the claims of others to get a sense of different claim techniques that may or may not pass muster by the USPTO's examination guidelines – this section includes some claim templates for common software inventions
- Claim drafting for "real-time" processes: "Real-time" processes are those that must run in a very specific time frame (e.g., between two hardware interrupts) –

an algorithm element may be recited as a "real-time" element as follows: "An apparatus comprising: … a timing element that defines a predetermined time interval [based on some physical condition or demonstrable state of the system being claimed or its environment]; a real-time element that performs the following steps within said predetermined time interval: (A, B, C)…" – alternatively: "A method comprising: … establishing a predetermined time interval based on some physical condition or demonstrable state of the system being claimed or its environment]; performing the following steps fully within said predetermined time interval: (A, B, C)…"

- Claim drafting for iterative processes: This claim style can closely follow the source code style: "A method comprising: … while x, iteratively performing the following steps: (A, B, C)…" or "A method comprising: … (A, B, C)… iteratively repeating steps B and C until X is true…" or "A method comprising: … setting a counter X to an initial condition; while X is not less than Y, performing the following steps: (A, B, C), and incrementing X…"

§8.05 The "If" Statement Iterator

- Claim drafting for logical control processes: This claim style also closely follows the source code: "A method comprising: … if X is less than Y, then performing the following steps: (A, B, C)…"

§8.06 Recursive Processes

- Claim drafting for recursive processes: This claim style also closely follows the source code: "A method comprising: .. defining a recursive procedure that includes the following steps: (A, B, using said recursive procedure, C)… performing said recursive procedure…" or "An apparatus comprising: … recursive processing element that performs the following steps at least in part by recursively using said recursive processing element: (A, B, C)…"

Chapter Nine: Examination of the Application by the Patent Office

§9.01 Patent Office Organization and Application Process from Patent Officer's Perspective

- USPTO processes for handling patent applications: U.S. patent examination typically takes 12-18 months from filing to issuance, and often longer; of course, many steps are involved in the process
- Step one – receipt: First, the application reaches the Patent Office mail room, which received about 186,000 apps in 1992 – the mail room creates a file and sends the application to the Classification and Routing Branch
- Step two – classification: The Classification and Routing Branch assigns the application an application serial number, checks for completeness, and if complete assigns the file a filing date – next, a group of experts within the Classification and Routing Branch assign the application within the patent classification system and the associated Art Unit – classification is based on the claimed subject matter (more specifically, the most comprehensive claim), and might also be based on the classifications of the closest prior art patents – this group classifies about 121 applications per day (some suggestions are being considered to allow applicants to suggest a classification)

- Step three – application processing: The Application Processing Team makes a more thorough determination of whether the application is complete – if so, the team issues an Official Filing Receipt to the applicant and completes a "coding sheet" with relevant application data – if not, the team forwards the application to the Special Processing and Correspondence Branch, which issues a Notice of Missing Parts – the Application Processing Team's goal is to process each application in 22-23 days
- Step four – review of drawings: The application is next forwarded to an Official Draftsperson, who examines the drawings; if inadequacies exist as per the (very detailed) 37 CFR rules on drawings, the draftsperson prepares a Notice of Drawing Objections and adds it to the file – the drawings don't need to be corrected here; the draftsperson forwards the application regardless of its status
- Step five – archiving: The application is next sent to the Microfilm Processing and Duplication Branch of the Micrographics Division, which records the application – this takes 2-3 days
- Step six – examination: Finally, the application is sent to the Art Unit that will examine the application, and the Supervisory Patent Examiner of the unit assigns the application to an examiner – by this point, the application file wrapper is complete, and application can commence – applications are processed in the order received (except in the case of "special" applications with expedited processing), so each art unit may be handling a large queue of applications that creates a backlog – examining units are organized into groups, which fall under one of the three categories of classification: chemical (groups 1100-1800), electrical (groups 2100-2900), and mechanical (groups 3100-3500) – each group has a Group Director responsible for the performance of the group – also, each group is divided into Art Units, which loosely correspond to a particular classification class; each Art Unit is led by a Supervisory Patent Examiner, who is usually an experienced examiner from the same unit – examination usually focuses on prior art, which is very difficult due to the sheer number of disparate sources of prior art (U.S. patents, foreign patents, technical journals, publications, textbooks, product literature, user manuals and documentation, etc.) – patent examiners must always have a background in science or a technical academic degree (though a background in law is not required; the Manual of Patent Examining Procedure contains everything the examiner needs to know about patent law and prosecution)
- Patent examining group 2300: Most software patent applications will be routed to examining group 2300, though multifaceted software inventions (e.g., combinations of software and mechanical devices) may be routed to different groups –this group handles classes 364 ("Electrical Computers and Data Processing Systems") and 395 ("Information Processing System Organization"), as well as part of class 381 ("Electrical Audio Signal Processing Systems and Devices") – this group has 13 art units, each of which handles an assigned set of subclasses from these patent classes – of course, this group has seen explosive growth in its application volume, and hence a growing queue backlog and protracted prosecution time, and hence the group is rapidly hiring new examiners – these examiners usually have computer science or computer engineering

backgrounds (though as recently as 1994, this group was not allowed to hire examiners with computer science backgrounds, as this credential was considered insufficiently technical)

§9.02 Application Process from Patent Applicant's Perspective

- Application parts: The applicant must prepare and file the application with (1) a specification, (2) at least one claim, (3) all drawings referred to in the specification, and (4) the names of actual inventors; delays in the commencement of prosecution, and even in the award of a filing date, may be caused by substantive or clerical mistakes (e.g.: if the applicant's draftsman originally creates Figure 1 but then later divides it into Figures 1A and 1B, but the application still references Figure 1, the application will not be accepted) – other parts can must be filed eventually, but need not be filed wit the application: the filing fee, a signed oath or declaration from each inventor, and an English translation of any part of the specification filed in a foreign language; omitting these items causes the USPTO to issue a Notice of Missing Parts, to which the applicant must respond within a certain period and with the payment of a surcharge ($65/$130 as of the date of this text) – of course, filing an application without all of these secondary parts secures a filing date in the absence of an inventor, but at a higher cost – another way to expedite examination is by a Petition to Make Special, under 37 CFR §1.102, which secures an expedited examination – petitions must show cause, and several kinds of rationale are available:
  - The inventor is over the age of 65 or in failing health
  - The invention relates to an important public cause: environmental quality ("materially enhance the quality of the environment of mankind by contributing to the restoration or maintenance of the basic life-sustaining natural elements"), energy ("materially contribute to the discovery or development of energy resources, or the more efficient utilization and conservation of energy resources"), recombinant DNA ("relating to the safety of research in the field of recombinant DNA"), or superconductivity ("involving superconductivity materials, and their manufacture and application")
  - The invention is ready to be manufactured in commercially significant quantities, but can't be manufactured until a patent is granted – requires an oath or declaration to a few details, including that the applicant or assignee has made a "careful and thorough" search of the prior art and believes all of the claims to be allowable (see MPEP §708.8)
  - The invention is currently being infringed by products actually on the market or methods actually in use – again, requires an oath or declaration to a few details, including that the *attorney* (not the applicant, as above) has made a "careful and thorough" search of the prior art and believes all of the claims to be allowable; also, must provide factual evidence of this infringement – this makes the patent attorney a prospective material witness, and may create a conflict of interest; thus, it might be wise to hire outside counsel to make this averment

- Duty of disclosure: 37 CFR §1.56 obligates "each individual associated with the filing and prosecution of a patent application" to disclose to the USPTO "all information known to that individual to be material to patentability" – this duty attaches to every pending claim throughout the pendency of the application – the purpose of this duty is to ensure that the applicant cooperates with the USPTO in the examination, since it's logistically impossible for the USPTO to cover the entire spectrum of prior art – materiality is defined as information that establishes or helps establish a *prima facie* case that a claim is unpatentable (this standard is further defined in the statute), or refutes or is inconsistent with a position of applicant on patentability or opposition to a USPTO position of unpatentability – thus, this duty is determined by an objective test, and the applicant is expected to make at least a minimal search (must include all prior art references cited against the applicant, e.g., in related U.S. and foreign patent applications) – this standard does not permit the applicant to withhold information simply because he believes that the claims are distinguished over it; it's good practice to err on the side of disclosing to the examiner, in order to avoid enforcement problems – this objective standard replaced the old test, which requires the applicant to disclose all information that the examiner might consider important; this prior rule was unduly burdensome, and so the standard was lowered to the current rule – for more information on information disclosure statements (IDS), see 37 CFR §§1.97-1.98
- Disclosure of prior art for software patents: In 1994, the Commissioner of the USPTO held public hearings on the state of software patents in Washington, D.C. (complete transcript in Appendix A of this text); common complaints from the diverse audience included the USPTO's inadequate collection of software prior art, and the diversity of sources of software prior art – thus, an applicant for a software patent will greatly help the USPTO and himself by disclosing as much prior art as possible; erring on the side of disclosure is even more important than for other classes – disclosed prior art should include (1) prior versions of the software embodiments that the invention replaces or improves (this can be disclosed by submitting the source code, the user manual and related documentation, or a printed copy of the README on the product disk), (2) trade journal articles, and (3) textbooks that describe algorithms and data structures used in or related to the invention – it is also helpful to disclose the details of experimental use (including beta testing) of the invention; see Chapter 13 for more info
- Appeal process: When an examiner has rejected an application twice (the second time as a final rejection), the applicant may appeal the examiner's actions to the Board of Patent Appeals and Interferences, which is an administrative court staffed by former examiners – its decisions can be appealed directly to the CAFC (35 USC §141), or can be brought before a federal district court by suing the Commissioner of the USPTO, with these decisions appealable to the CAFC (35 USC §145) – a common tactic in deciding how to appeal Board findings: if the appeal involves fact issues or would be facilitated by expert testimony, it should be filed in district court; if it's a straightforward legal issue, appeal directly to the CAFC – however, it may be better simply to file a continuation application and

ask the examiner to reconsider the grounds for rejection; this avoids the finality of a court ruling on the issue

§9.03 Software Patent Examination Guidelines

- Guidelines for Examination of Computer-Related Inventions: These guidelines, released by the USPTO in 1996, indicate how the USPTO examines software patents – the purpose is to establish a uniform procedure for such examination, with the disparate rules and standards properly applied – the Guidelines are now part of the MPEP, and so are used daily by Group 2300 in the examination of software patents – patents issuing from an examination conducted according to these guidelines have a strengthened presumption of validity, so it behooves the applicant to understand and maintain this process (of course, CAFC vacillation continues to undermine the presumption of validity; see Judge Nies's dissent in the order to withdraw the *Trovato* opinion)
- Overview of the Guidelines: The core of the Guidelines is a conceptual framework for examining patent applications for all "computer-related inventions," in the form of a step-by-step flowchart – the steps of the Guidelines flowchart:
    I. Rejections must be based on substantive law: use the *Freeman-Walter-Abele* test to reject claims directed to processes for solving mathematical algorithms – do not reject any patent applications on the basis of any "business method exception"
    II. Determine what the applicant invented and is seeking to patent: identify the practical application; review the disclosure and embodiments; and read the claims
    III. Conduct a thorough prior art search
    IV. Apply 35 USC §101 to determine patent eligibility, especially in light of the statutory category of the invention
    V. Apply 35 USC §112, both ¶1 and ¶2, to determine specification compliance
    VI. Apply 35 USC §102 and §103 to determine novelty and nonobviousness
    VII. Communicate findings, rationale, and conclusions

  The real value of the Guidelines is the amount of detail for standards, case law, etc. to be applied during each step of this process – e.g., the "utility requirement" is heavily supported by examples of useful and non-useful inventions (e.g., how to differentiate a CD-ROM containing patentable software and a CD-ROM containing unpatentable music data – thus, unpatentable material includes "functional descriptive material," "nonfunctional descriptive material," and "natural phenomena"; patentable material includes "products" and "computer-related processes that either result in a physical transformation outside the computer or are limited by claim language to a practical application within the technological arts"
- Analysis of the Guidelines: The practical result of the Guidelines is that an examiner's analysis must begin by classifying the invention as a statutory class (step IV), and must orient his arguments around that determination – this focus moves away from more subjective tests like *Freeman-Walter-Abele*, and is in better compliance with recent CAFC rulings on §101 patentability – thus, a

software patent applicant may wholly avoid any §101 problems by classifying the invention strictly as a machine or computer system; this prevents a characterization of the invention as an "abstract idea" or "law of nature" – also, the invention must have been reduced to practice (either by creating a working embodiment or by submitting a patent application in compliance with §112 ¶1)

- Annotated Guidelines: See pages 417-440 for an annotated text of the Guidelines

§9.04 Examination Guidelines for Computer-Related Inventions – Full Text

- Full text of the Guidelines: See pages 440-469 for the full text of the Guidelines without annotations

§9.05 United States Patent Office Training Worksheets

- Training worksheets: See pages 469-482 for examples of inventions and worksheets developed under the Guidelines

§9.06 Design Patent Protection for Computer-Generated Icons

- Icon design patents: 35 USC §171 authorizes the issuance of patents with 14-year terms for ornamental designs (must be embodied in an article of manufacture, and must "appeal to the eye as a product of aesthetic skill and artistic conception") – software developers have successfully obtained patents for computer-generated icons (e.g., Xerox obtained design patents in 1988 for 22 icons) – however, in 1989 the USPTO began rejecting these patents, claiming that the icons were not "embodied in an article of manufacture" – in *Ex parte Strijland* (1992), the Board clarified this "embodiment" rule: where the icon is "not merely a displayed picture, but an integral and active component in the operation of a programmed computer displaying the design," then a design patent can be issued – claim styling: "an ornamental design for an Information Icon or the Like" is unpatentable; but "an ornamental design for an Information Icon for a Display Screen of a Programmed Computer System or the Like" is patentable

§9.07 Patent Office Guidelines for Design Patent Applications for Computer-Generated Icons

- USPTO guidance on icon design patents: Following up on *Ex parte Strijland*, the USPTO issued a set of Interim Guidelines for the examination of icon design patents – patentability is thus contingent on (1) the presence of a computer monitor depicted in dotted lines (as the "article of manufacture") in the design patent drawing, and (2) the inclusion of the article of manufacture in the title of the design patent ("Computer Icon" is unpatentable; "Computer Screen with an Icon" is patentable)

§9.08 Interim Guidelines for Examination of Design Patent Applications for Computer-Generated Icons – Full Text

- Full text of the Interim Guidelines: See pages 486-488 for the full text of the Interim Guidelines

Chapter Ten: International Patent Prosecution for Software

§10.01 Worldwide Software Patent Protection

- Overview of worldwide software protection: The U.S. is the heaviest proponent of software protection, owing to its central role in the U.S. economy ($37 billion in worldwide sales in 1992) – Europe and Asia recognize software as partially protectable IP, but do not have systems as well-developed as the U.S.'s

§10.02 European Patents

- European patents: Europe grants patents for new inventions that are "susceptible of industrial application" – novelty is determined by the "inventive step" test, which simply holds that the invention but be new and useful to one of skill in the art – the patent term is 20 years from the filing date, and unlike the U.S., European patent offices publish all pending patent applications 18 months after filing (note: the U.S. is moving in this direction) – European patent laws do not declare certain classes of inventions to be patentable, but do rule out certain kinds of inventions, including "schemes, rules and methods for performing mental acts, playing games or doing business, and programs for computers" (European Article 52(2)(c)) and "presentations of information" (European Article 52(2)(d)) – this reluctance was first indicated in the Patent Cooperation Treaty (PCT), which, when passed in 1971, expressly authorized international search authorities to decline the examination of prior art involving software; the EPC then passed the European Patent Convention of 1971 with this exception of software from patent eligibility – these decisions relied on case law and European patent office guidelines that discouraged software patents; some countries even viewed the exception as unnecessary: Austria, Switzerland, and the Netherlands had national case law strongly adverse to software patents; UK rejected a 1965 application for a linear programming solution algorithm, characterizing it as not a "vendible product" and thus a product of the "mental steps" doctrine (this case, *Slee & Harris's Application*, is Europe's equivalent to *Gottschalk v. Benson*) – however, Germany was an advocate of software patents; they even reached the opposite decision in their branch of the *Gottschalk* litigation

§10.03 History of Software Patents in Europe and the European Patent Convention

- *Vicom/Computer-Related Invention* (1987): A surprising interpretation of EPC Article 52(2) arose from this case, which involved a patent application for a method of processing image bitmap data, and as an apparatus – however, the applicant admitted that the apparatus could be a general-purpose computer, prompting the EPO to reject both claims under Article 52(2) on the grounds that the novelty-conferring claim language "would only add a mathematical concept and would not define new technical subject-matter in terms of technical features" – the Board of Appeal reversed:

> A basic difference between a mathematical method and a technical process can be seen, however, in the fact that a mathematical method or a mathematical algorithm is carried out on numbers (whatever these numbers may represent) and provides a result also in numerical form... No direct technical result is produced by the method as such. In contrast thereto, if a mathematical method is used in a technical process, that process is carried out on a physical entity (which may be a material object but equally an image stored as an electrical signal) by some technical means implementing the method and provides as its result a certain change in that entity. The technical means might include a computer comprising suitable hardware or an appropriately programmed general purpose computer.

Thus, the Board relied on the last two words of Article 52(3): "The provisions of paragraph 2 shall exclude patentability of the subject matter or activities referred to in that provision only to the extent to which a European patent application or European patent relates to such subject matter or activities as such."

§10.04 Software Patents Under the EPC, *Vicom* Decisions

- Post-*Vicom* cases: In both *IBM/Data Processor Network* (1990) and *IBM/Computer-Related Invention* (1990), the Board of Appeal affirmed the patentability of software as a method (for coordinating network communication and displaying messages, respectively) – however, in *IBM/Document Abstracting and Retrieving* (1990), the Board of Appeal affirmed a rejection to a patent application for a data representation of a document that was conducive to retrievals by queries, holding this algorithm excluded under "schemes, rules and methods for performing mental acts"; the Board based its decision on its finding that the transformed data did not represent a physical object, and contained only information – similarly, *IBM/Semantically Related Expressions* (1989) involved a text processing algorithm for text analysis, which the Board rejected as only a linguistics technique, and not an invention
- *Vicom* reaffirmed: *Koch & Sterzel/X-ray Apparatus* (1988): This case involved a software algorithm for controlling an x-ray apparatus – competitors sought to have the patent invalidated by asserting that the "technical effect" occurred only at the end of the process and constituted "inconsequential post-solution activity" – the Board held the algorithm patentable: "the only fact of importance is that [the technical effect] occurs at all" – the appellants then cited Article 52(2) and challenged the rationale behind *Vicom*, which the Board supported thus: "While an ordinary computer program used in a general-purpose computer certainly transforms mathematical values into electric signals, the electric signals amount to no more than a reproduction of information and cannot in themselves be regarded as a technical effect… [and are] hence excluded from patentability by Article 52(2)(c) EPC. But if the program controls the operation of a conventional general-purpose computer so as technically to alter its functioning the unit consisting of program and computer combined may be a patentable invention."
- "Computer program product" and "signal" claims in Europe: *IBM/Windowing Environment* (1999): This case involved software patent claims for both a "computer program product comprising a computer readable medium" and a "computer readable medium," much like *Beauregard* – the Board affirmed both claims, citing 52(3) in denying that the algorithms were being claimed "as such" – however, the Board noted that the embedded software still had to have at least "the potential to produce a technical effect," noting that the mere physical transformation of data within the computer was insufficient – the Board even went as far as to suggest the patentability of signal claims (referring to an older case, *Colour Television Signal/BBC* (1990), in which TV signals were held not to be excluded from patentability by Article 52(2))

§10.05 EPO Guidelines

- EPO guidelines: Much like the USPTO, the EPO has published guidelines for the examination of software and computer-related patents – see pages 504-505 for relevant excerpts

§10.06 Japanese Intellectual Property System

- Overview: The Japanese intellectual property system has always been narrower, owing to a philosophical view of intellectual property as a "common good," and thus allowing much more latitude for infringement and imitation – e.g., in 1982, IBM exposed Hitachi as infringing on IBM patents, but the Japanese viewed Hitachi as appropriately using technology essential to its "socially valuable activity" – however, the Japanese patent system has been evolving rapidly toward the U.S.'s system, both in general and for software
- Features of the Japanese patent system: Japan's patent system features a first-to-file scheme and a term lasting the lesser of 15 years from publication and 20 years from filing – no duty upon applicant to cite prior art, but competitors may engage in lengthy pre-grant opposition that may whittle away the patent term – Japan also offers "utility model" protection that is limited to 10 years, but that is easier and cheaper to obtain and has the same scope of enforcement – the key difference is the threshold of inventorship: utility models are granted for "the advanced creation of a technical idea," while patents are theoretically limited to "highly advanced creations of technical ideas"
- Features of Japanese patent examination: Japanese patent law does not require the "best mode" disclosure, but merely enablement; in fact, many Japanese patents do not include a "best mode" when filed, but still translated into English and filed in the USPTO; thus, many U.S.-filed patents translated from Japanese applications are invalid – Japanese patent law allows the addition of new matter after filing, as long as the new matter does not contradict the application as filed – most non-Japanese applicants encounter serious issues with language translation, leading to mistranslations causing denied, delayed, or invalid Japanese patents

§10.07 Software Patents Under Japanese Patent System

- Software patents in Japan: Japanese patent law requires inventions to be "highly advanced creations of technical ideas by which a law of nature is utilized" in order to be patentable – the law expressly allows patents for software algorithms that control hardware resources or processing, algorithms that perform processing on "the physical or technical nature or properties of an object" (where "object" = "a signal, character, image, picture, data, layout, pattern, shape, hardware or the like") and inventions utilizing hardware in any way – this breadth is supported by the Ministry of Trade and Industry (MITI, the equivalent of the USPTO Commissioner), which in 1975 issued patent law guidelines authorizing patents for software that utilizes a "law of nature" – by contrast, unpatentable inventions include "information processing based on mathematical methods, schemes, rules or methods for doing business or performing mental acts," and hardware uses constituting an "inevitable restriction imposed by hardware resources" – unfortunately, MITI has been inconsistent over the years, occasionally proposing copyright or even *sui generis* protection for software

§10.08 Statutory Subject Matter Under Japanese Law

- Patentable algorithms in Japan: Based on the foregoing comments, an algorithm will be patentable if (1) it "utilizes a physical law of nature when processing information" (the key factors here appear to be the practical applicability of the algorithm, and the abstractness of the "law" by which the algorithm operates) or

(2) it "substantively utilizes hardware resources" (and not based solely on an "inevitable restriction") – Japanese patent courts have relied on some of the same rationale expressed in *Parker v. Flook* ("data-gathering steps," "post-solution activity," etc.), and many of the same substantive arguments can be applied to these arguments – unlike the CAFC, Japanese courts have been inclined to look past the claim language and consider the "claim as a whole," meaning the core of the invention, in deciding patentability – thus, the form of claim is not very relevant under Japanese patent law, although claims to programming languages, computer programs *per se*, and "computer program products" will be categorically rejected

§10.09 The Specification of a Japanese Patent Application

- Japanese patent specification: The specification of a Japanese patent identifies the "object" of invention, indicates its "advantageous effect," and demonstrates its "constitution," or method of construction and operation (enablement) – as noted, no "best mode" requirement – like U.S. patents, the Japanese patent specification is divided into typical headings (field of utilization, prior art, etc.) – it is important to have a well-detailed "objective" section; arguments to overcome prior art rejections must not only identify structural or functional differences, but must be related and promote the objective and benefit/improvement of the invention

§10.10 Drawings in a Japanese Software Patent Application

- Japanese patent drawings: Reliable drawings can be critical for overcoming prior art rejections, and can even be used to circumvent issues of mistranslation – however, source code is not very useful in Japanese patent prosecution

§10.11 International Acceptance of Software Patents

- Software patents in other countries: As of 1997, most foreign nations appeared to favor software patents, notably including Brazil, France, the UK, and Germany; the most strident opponents are Spain, Switzerland, China, Malaysia, and Egypt – many countries even support the patentability of "purely abstract data manipulation algorithms" – disparity continues

Chapter Eleven: What Software Is Patented?

§11.01 Software Patents of Microsoft Corporation

- Microsoft's patent strategy: Microsoft's appreciation of software patents has steadily grown over the years (official statement during 1994 USPTO hearings on patent protection: "Microsoft believes that the software patent law will continue to mature and we would trust rapidly enough to effectively support growing industry awareness and use of software patents") – it has taken this position despite losing a $120MM infringement suit (*Stac Electronics v. Microsoft Corp.* (1994)) – its early patents (1985-1995) were oriented around their operating systems, followed by applications and user interface elements – its interest in hardware and interactive television have grown over time, and networking has been a secondary interest
- The rest of this section summarizes Microsoft's patents, including abstracts and claim structure, during this period

§11.02 Other Software Patent Examples

- USPTO classification of software patents: Patent Office Group 2300, the examining group responsible for software and computer-related patents, issued 3,613 patents in 1993 – these early software patents philosophically inherit from Morse's early patent to the telegraph (held valid by the U.S. Supreme Ct in *O'Reilly v. Morse* (1854))
- Other software patentees: Borland holds patents to circumventing the 640K MS-DOS memory boundary – Lotus holds patents related to spreadsheet and object-oriented database technologies, each connected to its products – WordPerfect Corporation holds patents to word processing technologies
- Other software patent types: Software patents are now issuing for data structures (p. 587), for some "pure" algorithms (though placed in a useful context – p. 590), for user interfaces (p. 592), for multimedia algorithms (p. 595), and for business methods (p. 596)

Chapter Twelve: Strategic Software Patent Protection

§12.01 The Strategic Patent Portfolio

- The importance of patent portfolio strategy: Whereas claim drafting and prosecution resemble materials engineering, patent portfolio strategy resembles architecture – strategic pursuit and use of patents is the mechanism by which patents are applied to industry – an effective strategy must identify commercially important inventions and avoid commercial obstacles

§12.02 Analyzing the Topography of a Software Patent Portfolio

- Portfolio analysis: Patents only have meaning within the context of the landscape comprised of commercial, prior art, and competing patents – one good way of understanding this interaction is with a chart of clustered patent claims, relying either on the USPTO classification system or a commercial product (e.g., Mapit is a patent clustering software package relying on lexical analysis) – of course, understanding the scope of even a modest patent portfolio requires the use of a database – also, it is important to apply the information gleaned from patent claim clusterings and landscape analysis to the shaping of a patent portfolio development strategy

§12.03 Drawbacks of Classic Invention Disclosure Programs

- Invention disclosure quirks: Most companies must craft their IP strategy around their invention disclosure program, which offers some kind of bonus to employees who disclose inventions (sometimes with further benefits if a patent issues) – however, this process is discouraging and dismissive of inventors with little time, poor communication skills, or a lack of appreciation of the program or intellectual property – furthermore, patent filing decisions are often driven by office politics and a desire to encourage disclosure, rather than a patent strategy; resulting patent portfolios may be broad but not well-designed – this lack of focus is often reflected in the metrics by which the program is measured (rate of disclosure, number of patents, turnaround time, etc.)

§12.04 Developing Innovation with a Business Model

- Innovation development system: Rather than focusing on invention disclosures, patent strategists should find an opportunity to incorporate a patent strategy as part of the company business plan – a good opportunity for this assessment is

during the annual budget review, when patent analysis and budgets can be tied to specific research projects that are of value to the company; the patent group can then proactively keep tabs on the progress of these projects, and can identify key points for patenting technologies – this technique shifts the focus of the patent group onto the company research and development plan

§12.05 Putting It All Together

- The patenting process: One good way to conceptualize and present the patent process is as a conduit between raw assets (technology developments, know-how, databases, license agreements, etc.) and key patents – assets are sucked into the process by disclosure and innovation review programs; the results have to go through a threshing and ranking mechanism, and a cutoff point should be established based on the patent budget – thus, managing this process merely involves tweaking the processing components and adjusting the cutoff point (does the company believe that unworthy inventions are being patented, or that worthy inventions are not being patents?: adjust budget and cutoff accordingly)

§12.06 Metrics for Measuring and Improving the Effectiveness of Your Intellectual Property Strategy

- Metrics: Management of the patent strategy requires measurement of both qualitative and quantitative factors – relevant points: (1) what assets are available? (2) what is the value of each asset? (3) how are those assets being protected? (4) how effective is that protection? – thus, first consider what assets are available, and identify the protection covering each; this will provide a quantitative mapping, but doesn't really indicate the strength of protection for any asset – next, identify the extent to which the protection enhances value, including (1) the profitability of the asset, (2) the market share of the asset, and (3) licensing revenue of the asset (over and above profitability) – next, identify the degree of protection for the asset, including (1) the kinds of protection that the asset needs to compete or secure further development (including the difficulty an inventor would have in developing a competing product), and (2) the kinds of protection available and currently covering the asset – these assessments might not be accurately shown by a dollar value; a "good/better/best" ranking system may be more helpful – in the end, this assessment should cover the hierarchy of assets within the company, and should be a useful assessment of the IP strategy

§12.07 Fine-Tuning the Intellectual Property Strategy

- Fine-tuning: The assessments described above must be periodically reevaluated based on changing markets and business plans, and even on the changing availability and effectiveness of different forms of IP protection – a numerical score of each factor can be compiled for each asset, compiled into an asset matrix, and yielding a final score

§12.08 Summary of How to Measure Effectiveness of Intellectual Property Strategy

- Stepwise process: In summary, the process described above includes: (1) identifying assets, (2) identifying obtained and available forms of protection, (3) rating the value of each asset, (4) rating the level of protection of each asset, and (5) constructing a matrix and summing the results

§12.09 Architectural Patterns for Building a Software Patent Portfolio

- Architectural software design patterns: In order to assess the value of a software company's patent portfolio, one must understand the strategy behind its design process – many software packages primarily an information flow that the company conducts or transforms (digital video, e-commerce solutions); these kinds of projects are best promoted by protecting the means of information transfer – e.g., video-on-demand TV technologies can critically depend on a proprietary satellite data processing technique or an audio/video codec; in other words, the company is most valuably positioned between the parties on the ends of this communications pipeline – in other cases, the company fundamentally grapples with a time-vs.-space problem (e.g., competing techniques for processing large quantities of data, or competing techniques for communicating quickly, or compressing data most quickly and effectively, etc.); such industries inexorably pursue optimization, so identifying and protecting valuable innovation in these optimizations is a good IP strategy

Chapter Thirteen: Beta Testing Software Inventions

§13.01 Beta Testing and the Public Use and On-Sale Bars

- The dangers of beta testing: Under 35 USC §102(b), certain public actions trigger a one-year deadline for filing a patent application in the U.S., including (1) receiving a patent on the invention anywhere in the world, (2) publishing the invention anywhere in the world, (3) publicly using the invention in the U.S., and (4) selling or offering for sale the invention in the U.S. – public beta testing of software may, but does not *de facto*, trigger the third or fourth barriers – *City of Elizabeth v. American Nicholson Pavement Co.* (1878): the public use of an invention necessary to perfect it does not constitute a §102(b) public use; *TP Laboratories v. Professional Positioners, Inc.* (1984): "experimental use," even if public, is not a §102(b) public use – this is not exactly an exception to public use; rather, it makes the use not "public" at all (thus, this is a one-step test with the burden falling on the accused infringer to prove §102(b) "public use") – the details of the public beta test program are controlling for determining §102(b) "public use"

§13.02 Checklist for Analyzing a Beta Test Program for Compliance with the On-Sale and Public Use Bars

- Beta test features relevant to §102(b):
  - Was the beta test subject properly informed of the experimental nature of the invention? – if so, this helps establish experimental use (*LaBounty Mfg., Inc. v. United States ITC* (1992))
  - Was the beta test relationship reduced to writing? – if so, this helps establish experimental use (*RCA Corp. v. Data General Corp.* (1989))
  - Does the beta test relationship require secrecy? – if so, this helps establish experimental use (*TP Laboratories Inc. v. Professional Positioners, Inc.* (1984))
  - Is the beta test user being charged to participate in the beta test? Is the developer profiting from the beta test? – if so, this is not an experimental use (*U.S. Environmental Products v. Westall* (1990))

  - Has the software been fully reduced to practice? – if so, this is not an experimental use (*UMC Electronics Co. v. U.S.* (1987) vs. *Atlantic Thermplastics Co. v. Faytex Corp.* (1993))
  - Is the testing primarily intended to determine commercial suitability? – if so, this is not an experimental use (*LaBounty Mfg., Inc. v. United States ITC* (1992))

§13.03 Developing a Beta test Program That Avoids 102(b) Bars

- Beta test balancing: The public use rules strive to balance the inequities of depriving an inventor of patent rights, and depriving the public of an invention that has been committed to the public domain – this chapter describes the process of strengthening the beta test program to survive a §102(b) attack

§13.04 The Beta Test Plan

- Experimental purpose: First, establish (preferably in writing) the features of the invention that are being tested during the beta process – if the nature of the invention is hard to define during the beta, then this lack of distinctiveness can be expressed as an inadequate reduction to practice – one good approach involves listing the features and advantages of the software over competing products, and then describing the kinds of testing to be performed on each – also, explain the value and necessity of involving members of the public (following the rationale of *City of Elizabeth*) – it is also helpful to have a beta tester qualification program, in order to select beta testers who are (technologically) well-situated to fulfill the testing purposes of the beta
- Harvesting beta test results: It's critically important for the company to harvest and utilize the results of the beta test program – proactivity and solicitation are practically required: don't just rely on testers to respond with feedback – it's helpful to staff customer service people for soliciting beta test feedback; engineers are usually overwhelmed at this stage of development – it's helpful to record the dates on which feedback items were received, dates on which feedback items were forwarded to engineers, and the dates and degree of engineer responses – version-control software can be helpful for this task

§13.05 The Inventor's Degree of Confidence

- "Degree of confidence": The developer's impressions and statements about the state of the invention at the time of the beta can determine the existence of a §102(b) public use or on-sale bar – *Micro Chemical, Inc. v. Great Plains Chemical Co.* (1997): CAFC rejected a public use attack "because the inventor was not close to completion of the invention at the time of the alleged offer and had not demonstrated a high likelihood that the invention would work for its intended purpose upon completion" – contrast with *UMC Electronics Co. v. U.S.* (1987): CAFC found public use in part because a portion of the invention had been "sufficiently tested to demonstrate to the satisfaction of the inventor that the invention as ultimately claimed would work for its intended purpose" – these cases set forth this test: "even though the technical requirements of a reduction to practice have not been met, a sale or a definite offer to sell a substantially completed invention, with reason to expect that it would work for its intended purpose upon completion, suffices to generate a statutory bar… on the other hand, if the inventor had merely a conception or was working towards development of

that conception, there is not yet any 'invention' which could be placed on sale." – thus, the developers' mental state is relevant and should be documented with supporting evidence

§13.06 Auditing the Beta Test Plan

- Auditing the test plan: As the beta test progresses, it is important to confirm that it remains focused on testing, and that software developers are documenting testing and refinement actions that support an experimental use claim – in particular, try to capture test data that evidences the developers' "degree of confidence" in the conception and reduction to practice of the invention (e.g., include findings and data indicating that a component is still not working as intended and needs further development) – note: engineers may be reluctant to admit to software flaws, and may try to blame tester behavior, etc.; these statements can operate against the purpose of the beta test, and can raise a patentability bar

§13.07 The Beta Testing License Agreement

- Beta test license agreements: The beta test license agreement should bind the testers to secrecy, and should restrict the extent to which the software can be distributed – the license should also obligate the testers to create and provide feedback, and should obligate the engineers to accept and consider them – the agreement should (plausibly) characterize the testing as focused on technical, not commercial, viability – the length of the testing process should be fixed, and testers should be required to submit reports and return/destroy the software at the end of the program – payment by beta testers will operate against experimental use, but will not be fatal; similarly, payment *of* beta testers will facilitate, but not conclusively establish, experimental use

§13.08 Drafting Claims Commensurate With Beta Test Plan

- Patent drafting for beta-tested software: If the features of the invention were outlined in formulating the beta test process, it may be easy to draft similarly structured claims – at the very least, the claims should recite (as a limitation) one feature of the software that was a primary subject of the beta test – if developments critical to proof-of-concept and refinement occurred during the beta test, these should be discussed in the specification – by contrast, if the beta test identified certain elements as incomplete or uncertain, it might be damaging to omit them from the specification

§13.09 Forms of Testing That May Not Support Experimental Use

- Non-experimental-use public testing: Testing primarily intended to achieve certification by a regulatory agency is not a non-public-use "experimental use" – this includes tests for FDA or FCC approval, Underwriters Lab, etc. – particularly relevant to software, tests to achieve ISO 9001 certification (quality assurance for testing, development, production, installation, and servicing) rarely constitutes "experimental use"

Chapter Fourteen: Internet and E-Commerce Patents

§14.01 Internet Building Blocks

- Internet building blocks: The Internet is a vast network of connected machines; understanding how it works is key to understanding patents to inventions related thereto

§14.02 TCP/IP Protocol

- TCP/IP protocol: TCP/IP is the communications standard underlying all communication on the Internet – the TCP protocol specifies how information is transmitted in packets, while the IP protocol specifies routing and addressing

§14.03 Routers, Bridges, Hubs, and Gateways

- Routers, bridges, hubs, and gateways: Between any two devices on the Internet sit a host of routing devices that convey packets according to the IP protocol – each router contains a routing table indicating which way packets should be routed to reach certain destinations

§14.04 The Domain Name System

- The domain name system: The IP protocol specifies that all devices connected to the Internet have a four-digit IP address – because these addresses are cumbersome to remember, a "domain system" is layered on top of them that maps real-world, user-friendly names to specific IP addresses – each "domain" links to the first two digits of the IP address, and the rest of the user-friendly name determines the last two digits of the address – thus, loquitur.com maps to 165.55, and res.ipsa indicates a particular machine that the loquitur.com domain has assigned to 128.60; thus, res.ipsa.loquitur.com translates to IP address 165.55.128.60

§14.05 Client-Server Architecture

- Client-server architecture: This technology describes the communications structure of the Internet, where "servers" wait to provide services to the public (web pages, text, data, files, etc.), and members of the public ("clients") contact them to access those services – e.g., the USPTO hosts a server that offers webpages and patents to visitors – most servers do nothing but wait for contact by clients, while most human users of the Internet operate as clients that consume desired services

§14.06 Push Technology

- Push technology: This mechanism slightly breaks the mold of the client-server architecture – when clients indicate that they want to subscribe to a particular service, some servers on the Internet will spontaneously "push" data to them when it becomes available or updated – (this was new technology as of the date of writing, and was conceptually primitive, with not much detail)

§14.07 Cookies

- Cookies: Most client-server interaction is "stateless," i.e., the server begins the interaction by assuming that it knows nothing about the client – "cookies" change this model: a server may ask a client to retain a small packet of data called a "cookie" after the client disconnects from the server; when the client reconnects to the server at a later date, it provides the cookie to the server, which can operate differently based on this "remembered" information

§14.08 Streaming Audio and Video

- Streaming media: Older computers used to require an entire media file to be delivered to the client before it could use it – "streaming" media is a technology whereby a client machine starts receiving media data, stores it in a buffer, and begins displaying it from the buffer while continuously replenishing the buffer with additional data received from the server – thus, the client may begin playing

the media stream long before it has been fully received, perhaps almost immediately upon commencement of sending data

§14.09 The Internet Prior Art

- Internet prior art: Since the Internet is a bundle of different technologies, it's difficult to identify "Internet patents" without first identifying which technologies actually comprise the Internet – terminology typically used to describe Internet technologies (TCP/IP, HTML, etc.) began to appear in patent titles and specifications in the 1980's, and grew explosively throughout the 1990's

§14.10 Legal Issues Concerning Internet Patents

- Legal issues with Internet patents: Many Internet patents involve one or several of the following issues: (1) jurisdiction over an infringer; (2) identifying the infringer (is it the end user, the ISP, etc.); (3) might the infringer have statutory defenses to business method claims; (4) is the patent vulnerable to attack as reciting unstatutory subject matter

§14.11 Infringement and Territorial Issues Involving Internet Patents

- Territorial issues of infringement: 35 USC §271 specifies several legal techniques for enforcing a U.S. patent against a non-U.S. infringer (inducement of infringement, contributory infringement, exporting infringing devices to the U.S., exporting non-infringing devices to the U.S. that were made overseas by a method patented in the U.S., etc.)

§14.12 Statutory Subject Matter Issues with Internet Patents

- Eligibility of software for patenting under 35 USC §101: Although the USPTO and CAFC have established a firm basis for resolving §101 issues for software and business method patents, courts and patentees continue to struggle with the issue – e.g., *AT&T v. Excel Communications* (1998): the district ct of Delaware invalidated a patent for a telecommunications method as unpatentable under 35 USC §101; the CAFC took the issue on appeal to correct it, but of course this step required considerable legal costs

§14.13 Claim-Drafting Considerations

- Claim drafting for Internet patents: Internet-related inventions can be characterized as processes ("a method of…"), machines ("a computer comprising a general-purpose processor and a memory containing instructions for carrying out a process…"), or articles of manufacture (*In re Lowry*, *In re Beauregard*, etc.) – also included as a "manufacture" is a "signal" embodying an algorithm – surprisingly, this argument has fared well under §101: a signal is admittedly not an "article" of manufacture, but §101 extends patents to "manufactures," not only "articles"

§14.14 Propagated Signal "Carrier Wave" Claims

- "Signal" claims: One of the examples in the USPTO Examination Guidelines for Computer-Related Inventions featured a claim to "a computer data signal embodied in a carrier wave," which the USPTO treated as a "manufacture" eligible for patenting – the excerpt from the USPTO Examination Guidelines reflecting this example is on pages 668-673

§14.15 Some Specific Examples of Internet Patents

- Examples of Internet patents: See pages 673-685 for examples of various Internet patents, and pages 685-689 for template claims for different kinds of Internet-related technologies