A Proposal to Revise the Alice Test for Software Patents

By

Mojdeh Farsi

A dissertation submitted as part of the
requirements for the degree of

Doctor of the
Science of Law

of the
University of California, Berkeley

Committee in charge:

Professor Robert P. Merges, Chair
Professor David J. Teece
Professor Richard M. Buxbaum

Spring 2020

A Proposal to Revise the Alice Test for Software Patents

By Mojdeh Farsi

ABSTRACT

Most human innovations begin from an abstraction, a judicial exception that by itself is not patent-eligible. Abstract ideas are considered basic tools of scientific and technological work, which courts prefer not to award with a monopoly because exclusive ownership of these essential elements would only serve to hinder economic and innovative progress. The problem concerning patent eligibility under §101 is that we do not have any measurement or indicator to differentiate when a claim consists of only an abstract idea and when a claim does not. Currently, courts rely on precedent that is ambiguous rather than relying on more concrete measures based on the specific type of underlying software. Lacking this measurement, coupled with inherent software complexity, has resulted in inconsistent jurisprudence that is often inscrutable. These inconsistencies in the jurisprudence of cases involving computer programs stem from lacking a proper framework based on the functional nature of computer programs. There are several different proposals with inconsistent explanations as to how to apply the judicially created framework in determining patent eligibility. Still, none of them suggest a practical test specifically for software fields based on the functional nature of computer software to solve the patent eligibility problem. Additionally, proper disclosures are needed to combat high invalidity rates due to the lack of quality of patent applications.

This study addressed the confusion regarding the existing judicial framework (*Alice's* framework) in determining patent eligibility (§101) of software inventions. It proposed a new framework for determining patent eligibility in which the meaning of an abstract idea is defined in a practical way, as routine building blocks, based on the functional characteristics of software programs. These routine building blocks are not patent-eligible because the only embodiment is its fundamental components that are common in all inventions in that type of software. Granting any patent to claims of these essential elements will preempt other's innovative progress in that specific type of software. The functionality of this framework was tested on some software patent cases dealing with the §101 issue, as well as on one actual AI (Artificial Intelligence) pending patent application. Additionally, this study suggested proper disclosures through examples to improve the quality of patent applications by specifying some factors that might affect the validity of a software patent regarding the sufficiency of its disclosure.

Giving robust indicators to the patent examiners, judges, jurists, patentees, and practitioners to increase the certainty of predicting the outcome of the case based on the content quality of the underlying claims is the main contribution of this study. The result is significant because, with this legal paradigm, the unavoidably subjective assessments of computer programs in determining patent eligibility are formalized.

Keywords: Intellectual Property, Software Patent, Abstract Idea, Alice Test, Patent Eligibility Test, Sufficient Disclosure, Artificial Intelligence, Building Blocks, Section 101, Section 112, U.S. Patent Act.

Dedicated to

My Beloved Parents for their Blessings
And
My dear husband and soulmate for his love and endless support

TABLE OF CONTENT

LIST OF FIGURES

INTRODUCTION

The field of software patent has received a lot of attention over the past three decades has been debated extensively because of its complicated nature. Although it is now generally well-accepted that innovations in computer programming are best protected by patent law, a few decades ago this was far from apparent.[1] The confusion regarding whether copyright law or patent law is best suited for protecting software innovations stemmed from attempts to draw analogies between this new arena of software innovation and the more familiar fields of human creativity, namely tool-making and expression of ideas in the form of language or art. By accepting that patent law is best suited for protecting computer-related innovations, another unique challenge arises when it comes to the current patent eligibility regime. The current patent eligibility jurisprudence is comprised of a case-by-case, subjective analysis by judges and jury members who have different levels of knowledge in software programming. As a result, many software-related claims are invalidated inconsistently or even wrongfully, making the outcome of future cases uncertain.

The complexity of software patents increased in 2014 when the Supreme Court in *Alice Corp. Pty. Ltd. v. CLS Bank Int'*[2] invalidated a software patent for a business method, viewing it as an abstract idea[3] under 35 U.S.C § 101.[4] The Court's defined framework in determining patent eligibility made the issue even more complicated. After *Alice*, there was confusion as to how to apply the *Alice* two-step patent eligibility test on inventions, specifically software patent inventions.[5]

The problem of patent eligibility under §101 is that we do not have any measurement or indicator to differentiate when a claim covers an abstract idea and when a claim does not. Determining the patent eligibility of a disputed patent or claim is dependent on previous cases that are decided by the court, and it is not based on the specific type of its underlining software. Lacking this measurement coupled with software inherent complexity has resulted in inconsistent jurisprudence that is often inscrutable. These inconsistencies in the jurisprudence of cases involving computer programs stem from lacking a proper framework based on the functional nature of computer programs. Also, a lack of specific guidelines for determining efficient disclosures in patent applications affects the validity rate of software innovations.

There are several different proposals with inconsistent explanations as to how to apply the judicially created framework in determining patent eligibility. Still, none of them suggest a practical test specifically for software fields based on the functional nature of computer software to solve the patent eligibility problem.[6] Additionally, there has been almost no proposal out there to recommend any specific disclosure to improve the quality of patent application to this day. This study will examine the following research questions:

1- How does having a concrete indicator (test) in determining patent eligibility in software patent based on the functional nature of software affect the certainty of predicting the outcome of the case consistently?

---

[1] Julie E. Cohen & Mark A. Lemley, *Patent Scope and Innovation in the Software Industry*, 89 Cal. L. Rev. 1, 4 (2001).
[2] Alice Corp. Pty. Ltd. v. CLS Bank Int' l, No. 13-298 (Jun. 19, 2014); *see also* Molecular Pathology v. Myriad Genetics, Inc., 569 U.S. _, 133 S. Ct. 2107, 2116, 106 USPQ2d 1972, 1979 (2013). *See Also* MPEP § 2106.04 .
[3] *See generally* Interval Licensing, LLC v. AOL, Inc., 896 F.3d 1335, 1343 (Fed. Cir. 2018)
[4] *See generally* Molecular Pathology v. Myriad Genetics, Inc., 569 U.S. , 133 S. Ct. 2107, 2116, 106 USPQ2d 1972, 1979 (2013). *See also* MPEP § 2106.04 for detailed information on the judicial exceptions.
[5] David O. Taylor, *Patent Eligibility and Investment*, Cardozo L. Rev.,(forthcoming), http://ssrn.com/abstract=3340937.
[6] *See generally* Jeff Lefstin, Peter Menell, & David Taylor, *Final Report of the Berkeley Center Workshop: Addressing Patent Eligibility Challenges*, 33 BERK. TECH. L.J. 551 (2018).

2- What factors of disclosure in patent applications make a software patent more or less likely to be invalidated?

This study addresses the confusion regarding the existing judicial framework in determining patent eligibility (§101) of software inventions. It proposes a new framework for determining patent eligibility in which the meaning of an abstract idea is defined in a practical way based on the functional characteristics of software programs. Additionally, it suggests proper disclosures through examples to improve the quality of patent applications by specifying some factors that might affect the validity of a software patent regarding the sufficiency of its disclosure.

The key connection between the first part of my proposal on §101 and the second part of my proposal on disclosure §112 is the patent claim(s). Claims can be invalidated under §101 if the claims are not supported and justified in more detailed disclosure under §112. Detailed disclosures are needed to justify detailed claims, and if a claim is not supported by sufficient disclosure, it might be invalidated under both §101 and §112.

Giving robust indicators to the patent examiners, judges, jurists, patentees, and practitioners to increase the certainty of predicting the outcome of the case based on the content quality of the underlying claims is the main contribution of this study. The result is significant because by having a concrete framework based on the functional nature of the software, the outcomes of the disputed patents are made consistent. This consistency happens because there is no substantial confusion on how to apply the framework to future cases. Additionally, by having a unified and efficient template of software patent applications, the "inventive step" of the program will be more easily apparent and applied. As a result, applications themselves would be shorter and more concise, and judges/examiners will not need to spend so much time and effort combing through past cases/precedent to try to determine validity that end up often producing inconsistent findings.

In my research, I formalize the definition of software program and some patent eligibility terms that appear in this study, such as an abstract idea, and conventional and routine building blocks. After defining the problem with the existing judicial framework for the patent eligibility test under 35 U.S.C § 101, I formally propose a concrete and practical framework in determining patent eligibility in software patents based on the functional nature of software. I apply this framework on some software patent cases to show the applicability of my proposed test. Also, I propose proper disclosure to improve patent applications, and illustrate some examples to show the efficiency of this proposed disclosure. Finally, I apply the proposed disclosure on a specific field of software, AI (Artificial Intelligence) inventions.

Part I of this study is pertinent to clarify the meanings, definitions and concepts of some words and expressions, which will be used many times in this work. It explores the problem with the existing judicial framework for the patent eligibility test under 35 U.S.C § 101, and its inherent confusion in applying the test to software patent cases. It also proposes a concrete framework in determining patent eligibility in software patents based on the functional nature of software. The functionality of this framework will be tested on some software patent cases dealing with the §101 issue, as well as on one actual AI pending patent application.

Part II describes the problems of current patent applications and proposes a hypothetical disclosure as an example to improve the quality of patent applications. This proposed disclosure specifies some factors that might affect the validity of a software patent regarding the sufficiency of its disclosure. This hypothetical disclosure is applied on some recent cases to illustrate the efficiency of the proposed disclosure.

Part III Applies the patent eligibility proposal on one actual AI pending patent application. It also describes the hypothetical disclosure on Artificial Intelligence (AI) related inventions with slightly different changes based on the nature of the AI inventions, as well as applying it on the same actual AI pending patent application.

ACKNOWLEDGEMENT

First and foremost, I thank God for all his blessings in my life.

I would like to take this opportunity to give special appreciation to my mentor, sponsor, supervisor, and the chair of my Dissertation Committee, Professor Robert P. Merges, who has brought me all the way to my doctorate degree. I will always remember him as a person who gave me an opportunity of a lifetime. His supervision, insight, and knowledge of the subject matter accompany me through this research. His faith in me has been a constant source of motivation.

I am grateful to my Dissertation Committee members, Professor David J. Teece and Professor Richard M. Buxbaum, for their generosity, time, and guidance.

I cannot adequately express my gratitude to Professor Malcolm Feeley, my professor, and mentor, for his endless support and being a great source of inspiration for me. He is an example of a wonderful human being.

I am so grateful to the Dean of Berkeley School of law, Dean Chemerinsky, for his friendship, dedication, and fatherly support.

Most of all, I am grateful to Mrs. Evelyn Borchert, the director of the JSD program and the Academic Advisor. For so many years, she has provided guidance and endless support for many students, including myself. Her beautiful smile always brings me comfort.

I am grateful to other faculty members and staff at UC Berkeley, who have helped me along the way.

I am so grateful to all of my professors and staff at my previous University, the Bahai institute for higher education (BIHE), for all of their sacrifices in preparing me to achieve this accomplishment.

I am grateful to all of my professors and staff at the University of Delhi in India for helping me to get to where I am now.

I am so thankful to my sister, Marjan Farsi, for always being there for me. This project would have never been accomplished without her intellectual insights, detailed criticism, and scholarly discussions that helped me sharpen my thinking and define my arguments.

I am so grateful to my cousin, Tanya Zabeti, who has edited and improved this manuscript in countless ways. Her commitment and dedication have surpassed any expectations I could have hoped for.

I am so thankful to my friend, Dr. Holakou Rahmanian, for his intellectual help and support.

I wish to thank my Father, Jamshid Farsi, whom this dissertation mattered to the most, for his unfailing support and always being a source of inspiration to me. My love and prayers to my Mother, Jinous Yazdani, who is not in this world, but even now, I feel her constant and unwavering love and support.

I am so grateful to my Brother, Matin Farsi, who renders unending inspiration on how to be strong and not giving up during hard times.

My biggest appreciation is to my loving and supportive husband, Afshin Taghipoor, who helped me to fulfill my dream. This achievement would have never been accomplished without his understanding, encouragement, and utmost love and care.

I am so thankful to my Aunt, Janet Zabeti, who showered me throughout the years with her love, guidance, and endless support.

My special thanks to my dear friend, Ruhi Tavakoli, who is an exceptional woman with a kind and loving heart, for her love and support during the progress of this manuscript.

I am so grateful to my in-laws, who have supported my choices and gave me their utmost love.

My thanks to my dear friend, Anush Gasparyan, who has shown me, by her example, what a strong woman should be.

My sincere thanks to my aunts and uncles for their love and support.
Last but not least, I am indebted to my grandparents for their blessings and being wonderful role models in my life.

I am so fortunate to have all of you in my life.

A NOTE ON THE NATURE OF COMPUTER SOFTWARE

The definition of software program as a "functional machine part", comprised of algorithms, in combination with the specific function, the application, and the platform, on which they are employed, comprise a machine that may be tested for patentability.

Although most human innovations begin from an abstraction, the protection afforded by law has historically depended upon the form, in which ideas are cast. If an idea is implemented for the purpose of performing a useful function or task, the particular application, not the idea, may be suitable for grant of a patent. If an idea is expressed in some tangible medium, the individuating attributes of the particular expression of the idea, not the idea itself, may be protected by the copyright law.

Although it is now generally well-accepted that innovations in computer programming are best protected by the Patent law, a few decades ago this was far from being apparent.[7] The confusion regarding whether copyright law or patent law is best suited for protecting software innovations stemmed from the nature of computer programs; while they are machines, created to perform specific tasks, their inner workings are comprised of written language, that is to say, ideas are cast in the medium of notation. The Copyright Act defines a computer program as "a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result."[8] The combination of the words "written instruction" and "result" in the definition of a computer program reveals the dual nature of the identity of software programs. Computer programs are "Written statements, directions or instructions that directly or indirectly bring about a certain result." [8] If a computer programs is simply a form of writing in the traditional sense, it would be an expression of ideas and concepts, designed to interact with the human mind, in order to transfer and disseminate the knowledge contained therein. This is clearly not the case with the computer program. Although computer programs contain information within their forms, structures, algorithms and formulae, these abstractions are, with rare exceptions, connected to a function or a set of functions.

To complicate the matter further, while some of the code may be directed at machines or other programs, some of the notation is intended to interact with the human mind. The parts of the code, which interact with a machine or another program, are meant only to trigger a function in that machine or program. As such, they are a part of the "machine". Alan Turing[11] apparently thought so when he called his software model a "machine." The particular expression, form, format, language and notation of the code are not directly relevant to the function that is performed.

The parts of the code that are meant to interact with the human mind, and which can be conceived as expressions of ideas, may be designed to do more than create impressions on the human mind; they may be designed to trigger a response or a feedback from the human mind, in which case the expression is connected to a function of the machine and is, therefore, part of the machine function.

The holding of *Baker v. Selden*[9] that provides that blank forms are not copyrightable, even if the structure of the forms reflects the essence of an original work of literature, shows that those aspects of a work, which are incidental to the idea, and which are inextricable from the function, system or process described by the work, are not copyrightable.

---

[7] Julie E. Cohen & Mark A. Lemley, *supra* note 1.

[8] Cohen *et al*., *Copyright in a Global Information Economy*, (3d ed. 2010) Wolters Kluwer, United States.

[9] Baker v. Selden, 101 U.S. 99 (1879).

In *Lotus Dev't Corp. v. Borland Int'l*[10], the court held that the human interface of the program is not copyrightable, chiefly because it is tangled with the "method of operation", and the interaction of the program with the human mind is considered a part of the machine function.

It is largely undisputed that the literal aspects of the code, its notation and expression, are protected under the copyright law.[18] In computer software innovations, which protection is sought lies entirely on the functional aspect of the program; the notation and expression are largely irrelevant. According to the Merger Doctrine[11], if the expression is inextricably merged with a function, it would not be protected by copyright law, but rather by the patent law.

In the *Baker* case, although the book was protected as original literary expression, the form was held to be a means of putting the ideas contained in the book into practice, i.e. a function. Unlike patent law, since Copyright law only covers expression, here it is not a suitable paradigm to protect the functionality of the *Baker's* invention.

Copyright protection for software innovation permits to use of a particular expression if based on efficiency or external exigencies that expression is the only way, or one of a few ways, that a function can be achieved. In *Oracle v. Google*[12], the portion of the Google code, which was copied verbatim from Oracle's code was considered necessary for the performance of the program's functions within the constraints of the Java API. Google had re-written 97% of the code in its program in order to achieve the very same functions that the Oracle program's methods and subroutines achieved. It was in substantially the same order, with the same structure as the Oracle's program. It was successfully argued by Google that compatibility with the Java Application Program Interface (API) dictated the use of not only the structure and the functions, but also the 3% of the code that was copied verbatim. The code in question was comprised of headers, required by java to introduce the particular methods that followed. The structure was more or less dictated by the Java environment, and was proven to be the most efficient structure for the achievement of the functions. Moreover, the structure merged with the functions of the program, which precluded it from protection under the copyright law. None of the emulated functions were the property of Oracle under the patent law, and copyright law did not protect these functions. The court in this case held that:

> So long as the specific code used to implement a method is different, anyone is free under the Copyright Act[13] to write his or her own code to carry out exactly the same function or specification of any methods used in the Java API. It does not matter that the declaration or method header lines are identical.[14]

Inventing a new method to deliver a new output can be creative, even inventive, including the choices of inputs needed and outputs returned. The same is true for classes. But such inventions—at the concept and functionality level—are protectable only under the Patent Act. Duplication of the command structure is necessary for interoperability.

---

[10] Lotus Dev't Corp. v. Borland Int'l516 U.S. 233 (1996).

[11] Pamela Samuelson, *Questioning Copyrights in Standards*, Boston College L. Rev, 48:193, 215 (2007).
As Samuelson puts it: "The merger doctrine holds that if there is only one or a very small number of ways to express an idea, copyright protection will generally be unavailable to that way or those few ways in order to avoid protecting the idea." Oracle v. Google, 847 F.Supp.2d 1178 (2012).

[12] Oracle Am., Inc v. Google Inc, 847 F.Supp.2d 1178. (2012).

[13] Copyright Act 1976, 17 U.S.C.A. § 101 et seq.

[14] Oracle Am., Inc v. Google Inc, 872 F. Supp. 2d 974 (N.D. Cal. 2012).

On May 9, 2014, the Federal Circuit Court partially reversed the district court ruling and remanded the issue of fair use back to the district court.[15] The Federal Circuit Court held that declaring code and the structure, sequence, and organization of the API packages are entitled to copyright protection.[16]

In my opinion, APIs are designed to trigger a response or a feedback from the human mind, in which case the expression is connected to a function of the machine and is, therefore, part of the machine function. As a result, APIs are patentable not copyrightable.

One of the more interesting attempts to stretch the scope of the copyright law to cover computer software is the substantial similarity test.[17]According to this test, in order to examine two software programs for similarity of expression, one must analyze the "golden nugget" of expression that is copyrightable by removal of certain elements of the program from consideration. Abstractions, such as mathematical and logical algorithms are the first to be removed from consideration.[18] Next, one must filter out what exists in the public domain. The elements of the program that are dictated by efficiency - that is to say, the best way of achieving a function - are removed from consideration, as they represent a merging of expression and function. External exigencies, such as the mechanical specification of the machine that is being addressed, compatibility requirement of operating systems and other programs, computer design standards, demands of the industry serviced, and widely accepted industry practices are also omitted from consideration in the comparison.

In *Computer Associates Int'l, Inc. v. Altai, Inc*[19] the test of substantial similarity was applied and, not surprisingly, after all of the filtrations and removals from consideration, not much was left to compare, and no relief was provided to Computer Associates Int'l, Inc.

The dual nature of computer programs and their inherent complexity has resulted in inconsistent jurisprudence. Describing the nature of computer programs as a functional machine part, created to perform a specific task, help us to make an appropriate framework to protect software innovations, and overcome the problems of the existing regime.

---

[15] Oracle Am., Inc v. Google Inc - Cross - Appellant. Court of Appeals for the Federal Circuit. May 9, 2014. (last visited May10,2020). *See*, Rosenblatt, Seth (May 9, 2014). *Court sides with Oracle over Android in Java patent appeal*. *CNET*. (last visited May 10, 2020)

[16] *Ibid.*

[17]*Altai* introduced a new test for determining Substantial Similarity for Computer Program named, Abstraction-FiltrationComparison. "Initially, in a manner that resembles reverse engineering on a theoretical plane, a court should dissect the allegedly copied program`s structure and isolate each level of abstraction contained within it. This process begins with the code and ends with an articulation of the program`s ultimate function." https://www.bitlaw.com/source/cases/copyright/altai.html (last visited May 28, 2020).

"For filtration, as Professor Nimer suggests, a "successive filtering method" for separating protectable expression from non-protectable material. This process entails examining the structural components at each level of abstraction to determine whether their particular inclusion at that level was "idea" or was dictated by consideration of efficiency, so as to be necessary incidental to that idea; required by factors external to the program itself; or taken from the public domain and hence is nonprotectable expression. " Peter J Groves, *Sourcebook on Intellectual Property Law*, p.304(1997) Cavendish Publishing limited, London.

"The third and final step of the test for substantial similarity entails a comparison.

Once a court has sifted out all elements of the allegedly infringed program which are "ideas" or are dictated by efficiency or external factors, or taken from the public domain, there may remain the core of protectable expression."

https://www.bitlaw.com/source/cases/copyright/altai.html. (last visited May 28, 2020).

See, Cohen *et al*., *supra* note 8.

[18] Cohen *et al*., *supra* note 8.

[19] Computer Associates Int'l, Inc. v. Altai, Inc, 982 F.2d 693 2d Cir. 1992.

CHAPTER ONE: PATENT ELIGIBILITY PROPOSAL

1.1 Patent Eligibility Related Terms

In the following, I redefine some of the patent eligibility terms that appear in the court holdings and patent-related documents for use in my proposal. I also define a new term, "Routine Building Block", that applies in my patent eligibility test proposal:

- Abstract Idea: Basic or fundamental concept or idea, the routine building block
- Conventional: Generally accepted standards
- Routine: Customary or regular course of procedure; general steps of a process
- Building Block: Basic or essential element(s) or component(s) of a system
- Routine Building Block (RBB): Basic and essential element(s), steps, or component(s) of a process or system. High-level functions in each type of computer software, capable of operating for its primary purpose.

Before diving into the main point, it is necessary to differentiate between the "building block of human ingenuity" mentioned by the court in *Alice Corp. Pty. Ltd. v. CLS Bank Int'* [20] and the "routine building block" concept I define. The "building blocks of human ingenuity" include the abstract idea, natural phenomena, and laws of nature, whereas "routine building blocks," as I define here, mean the ***basic, primary, or minimum steps*** for accomplishing the primary action of the specific system. As an illustrative example, with the help of a recent case, *Berkheimer v. HP Inc.*[21], and knowledge of the skilled person in the art, the routine building block activities for asset management systems is determined as follows:

1. Converting the file format to one that is acceptable by conventional computers.
2. Extracting data with conventional tools available in programming languages.
3. Storing data into a data model.
4. Retrieving, searching, sorting, editing, and deleting.

These are necessary procedures for this asset management system, with which the system functions for its primary purposes. Even a small and trivial contribution beyond these building blocks will permit a claim to pass §101 under my proposal.[22]

[20] Alice Corp. Pty. Ltd. v. CLS Bank Int' l, No. 13-298 (Jun. 19, 2014) (the specification of Alice`s invention is: "The management of risk relating to specified, yet unknown, future events by employing an intermediary to facilitate simultaneous exchange of obligations. "In this case, the court stated that "there is no significant "inventive concept" in the asserted method claims. All the method steps exist in the prior art (use of an escrow.) The *Alice* provide the abstract idea of third-party intermediation and adding the words: "apply it" on a computer." The Supreme Court holds that Alice's patent claims "are drawn to a patent-ineligible abstract idea under 35 U.S.C. § 101" http://www.scotusblog.com/case-files/cases/alice-corporation-pty-ltd-v-cls-bank-international/)

[21] Berkheimer v. HP Inc., 881F.3d1360 (Fed. Cir. 2018) (holding that claims 1–3 and 9 of the '713 patent are ineligible under 35 U.S.C. § 101.) "*Id*., cert. denied, F.3d (U.S. Jan.13.2020)". (the Supreme Court has denied certiorari in *HP Inc. v. Berkheimer* (18-415) patent eligibility petition.)

[22] I will give more examples of my approach, starting on page 11.

1.2 Analyzing the problem of Abstractness in patent eligibility under § 101

The reason why an abstract idea is not patent-eligible and would preempt other innovations is that the only embodiment is its fundamental components that are the building blocks common in all inventions in that type of software. Since these building blocks are high-level functions that exist in all inventions in that type of software, they are considered too broad and not patent-eligible. Therefore, by granting a patent for these essential elements, other innovations are hindered from patenting. The problem lies not only in patenting these basic ideas that are mainly the sum of these basic essential elements, but also the potential harm in preventing others from protecting their novel and technological innovations.

In my opinion, the problem of patent eligibility under §101 is we do not have any measurement or indicator to differentiate when a claim covers an abstract idea and when a claim does not. Lacking this measurement coupled with software inherent complexity has resulted in inconsistent jurisprudence that is often inscrutable.

The complexity of software patents increased in 2014 when the Supreme Court in *Alice* invalidated a software patent for a business method, viewing it as an abstract idea.[23] The Court's defined framework made the issue even more complicated. The *Alice* framework consists of two steps[24]: 1) whether the claimed invention falls into the § 101 judicial exception of an abstract idea; and if so, 2) determine if there is any significant limitation in the components of the claimed invention or the invention as a whole to transform the abstract idea into the patent-eligible application.

After *Alice*, there was confusion on how to apply the *Alice* two-step patent eligibility test on inventions, specifically software patent inventions.[25] There are several different proposals with inconsistent explanations as to how to apply *Alice's* two-step test in determining patent eligibility.[26]

Professor Robert Merges states in his article entitled "Go Ask *Alice*,"[27] that the problem lies in determining "when does a claim cover an abstract idea and when does it not?" With this ambiguity surrounding the *Alice's* two-step test, I propose a concrete way to address the problem and determine patent eligibility in a more clearly defined manner: by determining the definition of an abstract idea in a practical way to be applied consistently. Then I establish the relationship between step two of the *Alice* test with other patentability provisions.

[23] Alice, *supra* note 20.
[24] *See generally* Mayo Collaborative Services v. Prometheus Laboratories, Inc., 566 U.S. 66 (2012)
[25] Taylor, *supra* note 5.
[26] *See generally* Jeff Lefstin, Peter Menell, & David Taylor, *Final Report of the Berkeley Center Workshop: Addressing Patent Eligibility Challenges*, 33 BERK. TECH. L.J. 551 (2018)
[27] Robert Merges, *Go Ask Alice: What Can You Patent After Alice v. CLS Bank?*, Berkeley Technology L.J. SCOTUS blog. (2014).

1.3 Patent-Eligibility Proposal

In considering whether an invention is patent-eligible, we should determine if it is a basic idea in its essence; that is, whether there is nothing more than a basic idea and routine building blocks in the method, component, or any combination of components. If so, then it is patent ineligible under § 101. If it has anything more than these routine building blocks, trivial or not, then it is patent-eligible under § 101. Determining whether these additional elements have an inventive step, or is novel and non-obvious enough, should be a question of patentability under §§ 102 and 103 rather than the existing court method of using §101 as the eligibility test.

Here, the difference between my proposal and the *Alice* framework is that *Alice's* two-step test for determining patent eligibility requires measuring and assessing that "something more" than an abstract idea to pass §101. In contrast, my proposal suggests that "anything," trivial or not, beyond routine building blocks, makes the software claim pass the §101 threshold. For measuring the inventive step of the invention, the claim should be examined under §§102, 103, and 112 for testing its patentability rather than §101 as the eligibility test. The reason for that is "anything more" than routine building blocks is not the same as building blocks. Therefore, the provisions that the routine building blocks will be examined should be different from the provisions that non-routine building blocks will be assessed. As a result, this anything more than routine building blocks should be assessed under other patent provisions such as §§102, 103, and 112 rather than 101.

Some scholars question the existing practice by asking, "Why should there be a step two in an abstract idea analysis at all? If a method is entirely abstract, is it no less abstract because it contains an inventive step?"[28] These scholars believe that

> If a claim recites 'something more,' an 'inventive' physical or technological step, it is not an abstract idea and can be examined under established patentability provisions such as §§102 and 103. Step two's prohibition on identifying something more from 'computer functions [that] are well-understood, routine, conventional activity [ies] previously known to the industry,' is essentially a §§102 and 103 inquiry. Section 101 does not need a two-step analysis to determine whether an idea is abstract…§ 101 requires further authoritative treatment.[29]

This acknowledgement of shortcomings of existing practices is what my proposed test is attempting to improve. Therefore, based on my proposal, determining the "inventive step" of the claimed invention falls under patentability provisions §§ 102, 103, and 112 and not the patent eligibility test under § 101.

Another major difference between my proposal and the existing practice is that I define the abstract idea in a practical way, as routine building blocks, which help me to propose a particular measurement based on the nature of the underlying type of software. This measurement is for differentiating between the routine building blocks and non-routine building blocks (anything more). Routine building blocks as I defined are the ***basic, primary, or minimum steps*** for accomplishing the primary action of the specific system. The non-routine building blocks are the ones that, although without them the invention's primary functionality is fulfilled, they are considered "anything more" than the routine building blocks to be evaluated under other

[28] Menell, Peter S. and Lemley, Mark A. and Merges, Robert P., *Intellectual Property in the New Technological Age*, p.316 (2019).
[29] *Ibid*.

patentability provisions. Further explanation on this concept will be provided in the following section.

In determining patent eligibility subject matter under 35 U.S.C § 101, I suggest the following one step proposal:

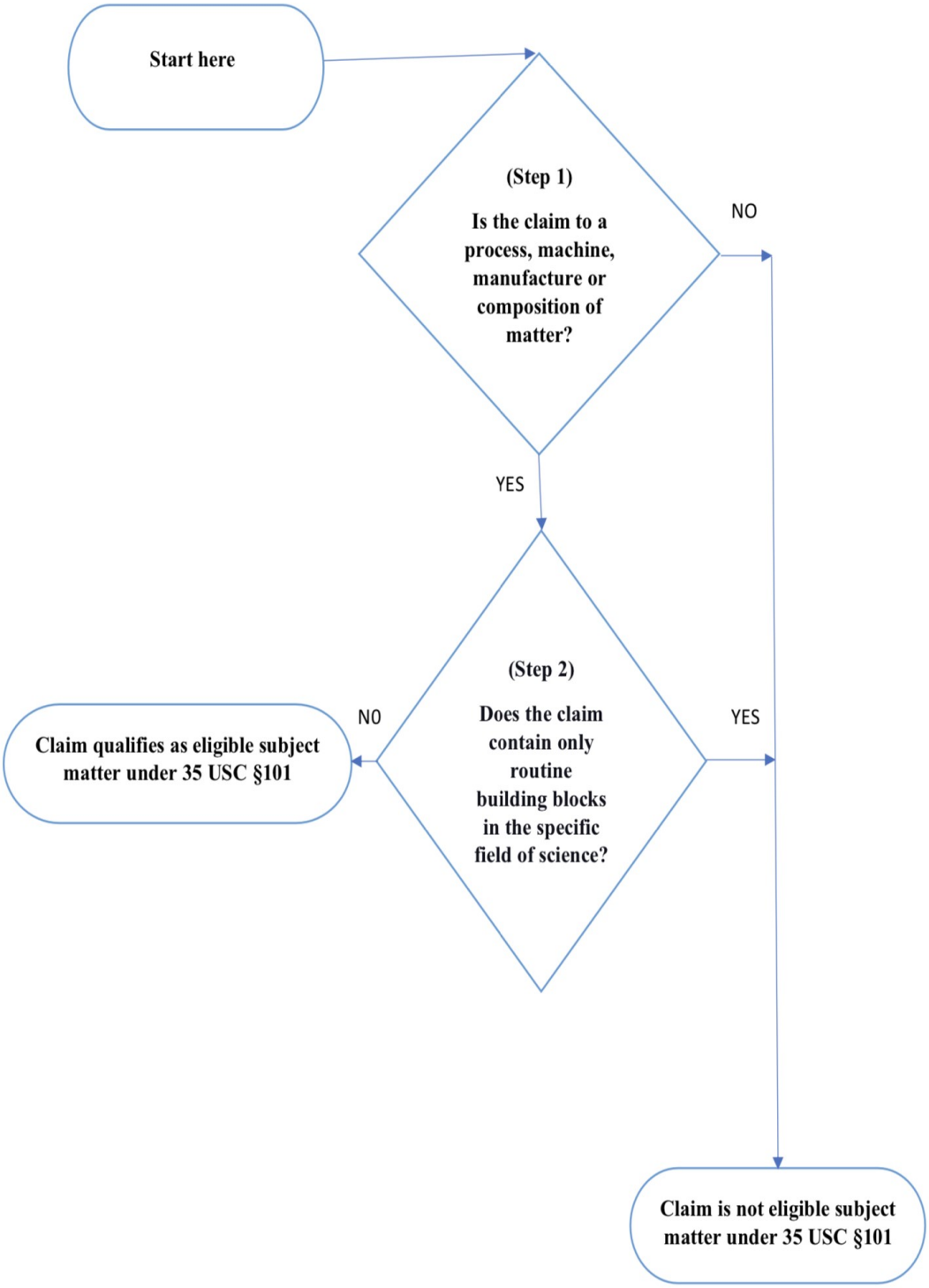
1.3.1 Proposed Patent Eligibility Test: Routine Building Blocks Determination

In order to ascertain the abstractness of a specific claim/invention, we need to determine the fundamental and essential steps or components of the underlying software, the "Routine Building Blocks" of that particular type of software. These Routine Building Blocks are so essential as to be incapable of removal without destroying the invention's primary functionality. For example, a "data asset management system" being separated from its necessary procedures, such as converting the document to the standard input file format, extracting data, storing data into a data model, retrieving, searching, editing, and deleting, would impair its functionality and essential purpose of the system.

In determining these building blocks, the examiners or the courts need to be more familiar with the routine building blocks in the field of computer science or software engineering that they are assigned to evaluate. Helpful guidance includes court precedent,[30] expert input, and knowledge of a skilled person in the art. By defining the routine building blocks of the underlying software, we are able to more narrowly delineate an unambiguous construction of the term abstract idea into a tighter framework. Not to mention, that routine building blocks of software innovation may be different from one type of software to another. The following flowchart (FIG.1) illustrates my patent eligibility test proposal:

[30] Merges, *supra* note 27.

Figure 1: Proposed Patent Eligibility Test Flowchart



1.3.1.1 Routine Building Block determination: *Berkheimer v. HP Inc.* case

As I mentioned in the section titled "patent eligibility related terms," the building block activities for asset management systems consists of:

5. Converting the file format to one that is acceptable by conventional computers.
6. Extracting data with conventional tools available in programming languages.
7. Storing data into a data model.
8. Retrieving, searching, sorting, editing, and deleting.
9.

These are necessary procedures for any asset management system, with which the system functions for its primary purposes.

The U.S. Patent No. 7447713 (''the '713 patent'') relates to digitally processing and archiving files in a digital asset management system. In this case, "Berkheimer has sued HP alleging infringement of the '713 Patent. The allegedly infringing products and services are HP's enterprise document automation software and platforms, such as HP EXSTREAM."[31]

The Patentee in the *Berkheimer* case considered "reducing redundancy in his asset management system and enabling one-to-many editing as the purported improvements".[32] The Federal Circuit held that the "claims are directed to the abstract ideas of parsing and comparing data (claims 1-3 and 9), parsing, comparing, and storing data (claim 4), and parsing, comparing, storing, and editing data (claims 5-7) based upon a comparison of these claims to claims held to be abstract in prior Federal Circuit decisions." [33]

Although the Court tries to determine the routine and conventional activities of the system, which make it patent-ineligible, there is no specific measurement or concrete indicator that the court follows in distinguishing an abstract idea to be able to further recognize these "abstract ideas" in future cases. In my proposal, instead of using the word "abstract idea," which is vague and unclear, I use "routine building blocks" for the underlying software. The word "routine" implies the base process steps in an invention. Therefore, the "routine building blocks" describes the **basic, primary, or minimum steps** required for accomplishing the primary action of the system in each type of computer software. The reason that I emphasize the "underlying software" is that these building blocks vary from one type of software to another. For example, building blocks of data asset management systems are different from the building blocks of database systems.

In *Berkheimer's* case, the Court should be more precise in determining the system building blocks as I have enumerated in my applied proposal. The Court held that "claims 4-7 do contain limitations directed to purported improvements described in the specification (e.g., claim 4 recites "storing a reconciled object structure in the archive without substantial redundancy," which the specification explains improves system operating efficiency and reduces storage costs), raising a genuine issue of material fact as to whether the purported improvements were more than well-understood, routine, conventional activity previously known in the industry."[34]. The Federal Circuit, therefore, "reversed the district court's decision on summary judgment that claims 4-7 are patent-ineligible and remanded for further fact-finding as to the eligibility of those claims."[35]

[31] Berkheimer v. HP Inc., 881F.3d1360 (Fed. Cir. 2018) (Mem.)(2018), https://www.uspto.gov/sites/default/files/documents/memo-berkheimer-20180419.PDF.(last visited Mar. 27,2020).
[32] Berkheimer, *Ibid.*
[33] Berkheimer, 881 F.3d at 1366-6.
[34] *Id.* at 1370.
[35] *Ibid.*

Regardless of whether the Federal Court's finding was correct, and Berkheimer's invention improves the system efficiency and could, therefore, be patent-eligible, this determination should be examined under §§ 102, 103, and 112 and not under §101 as the Court decided. I will further discuss this below.

In the following, I apply my patent eligibility test proposal to the *Berkheimer* case to show how Berkheimer's invention will be rejected under my analysis.

The routine building blocks of the *Berkheimer* case are as follows:

1- Converting the documents into standard input formats: This step is considered a routine building block since, to access the data, we need to convert it to a standard format that is understandable for the computer (e.g., pdf, gif, jpeg).

2- Parsing and tagging (extracting data from the standard format input document): This step is also considered a routine building block. For manipulating the data in a document (editing, removing, adding, and deleting), it is essential to parse and read (extract) the document and element properties and values by pre-defined tools and libraries in conventional programming languages.

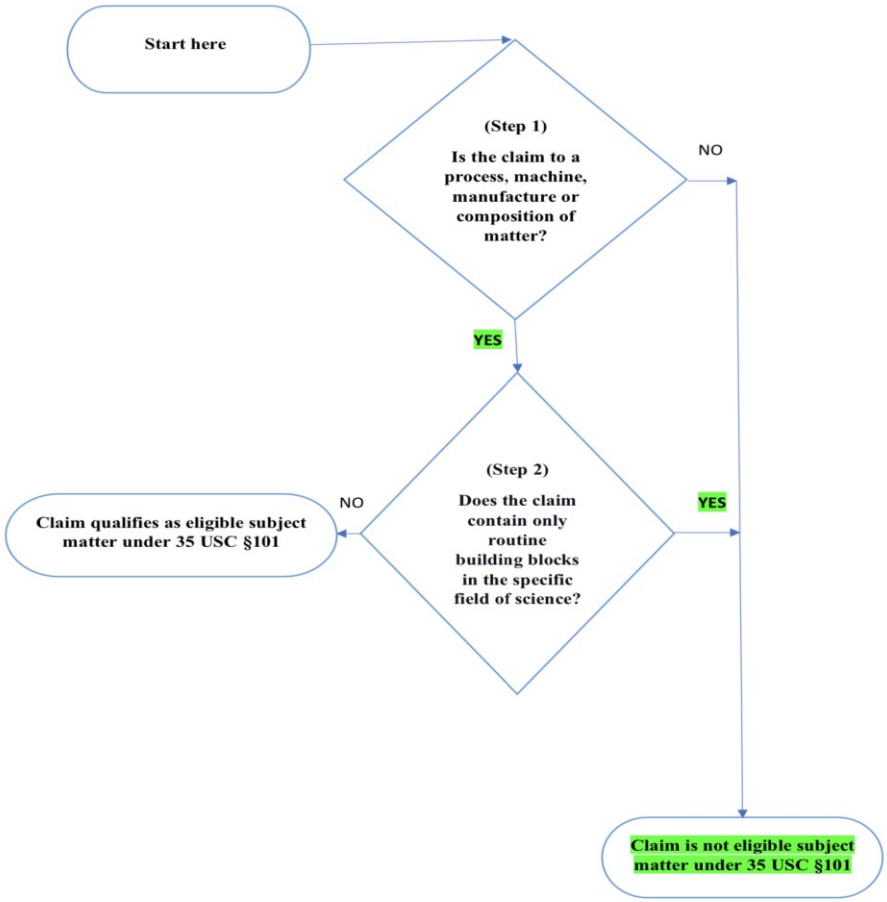
3- Defining rules for data storing and manipulation: Defining rules is also another routine building block. Here, inventor designs and implements a User Interface (UI), which allows the user to define the rules himself to store and manipulate (e.g., editing, reconciling, deleting) the data automatically or manually.

4- Creating a data model and storing data in a data model: This is a routine building block as well because after you extract the data from the document by performing the above steps, there is no way but to store this data into storage called a data model. Here, the inventor uses a conventional data model, object-oriented, to store and organize the data.

5- Data manipulation functionalities (edit (reconciliation), delete, add) and data retrieving and searching: These should be considered routine building blocks since these functionalities are essential for every asset management system.

All other actions, such as comparing, notifying, downloading/uploading are secondary functionalities derived from previous steps. Once you get access to the content or data and store it on a storage media, you may compare it with other contents and notify a user about the differences or similarities of the contents or you can download/upload it with the conventional commands or libraries available in each programming language. The following flowchart (FIG.2) is a visual demonstration of my patent eligibility test as it applies to the *Berkheimer* case:

Figure 2: Application of Proposed Patent Eligibility Test: *Berkheimer* Case



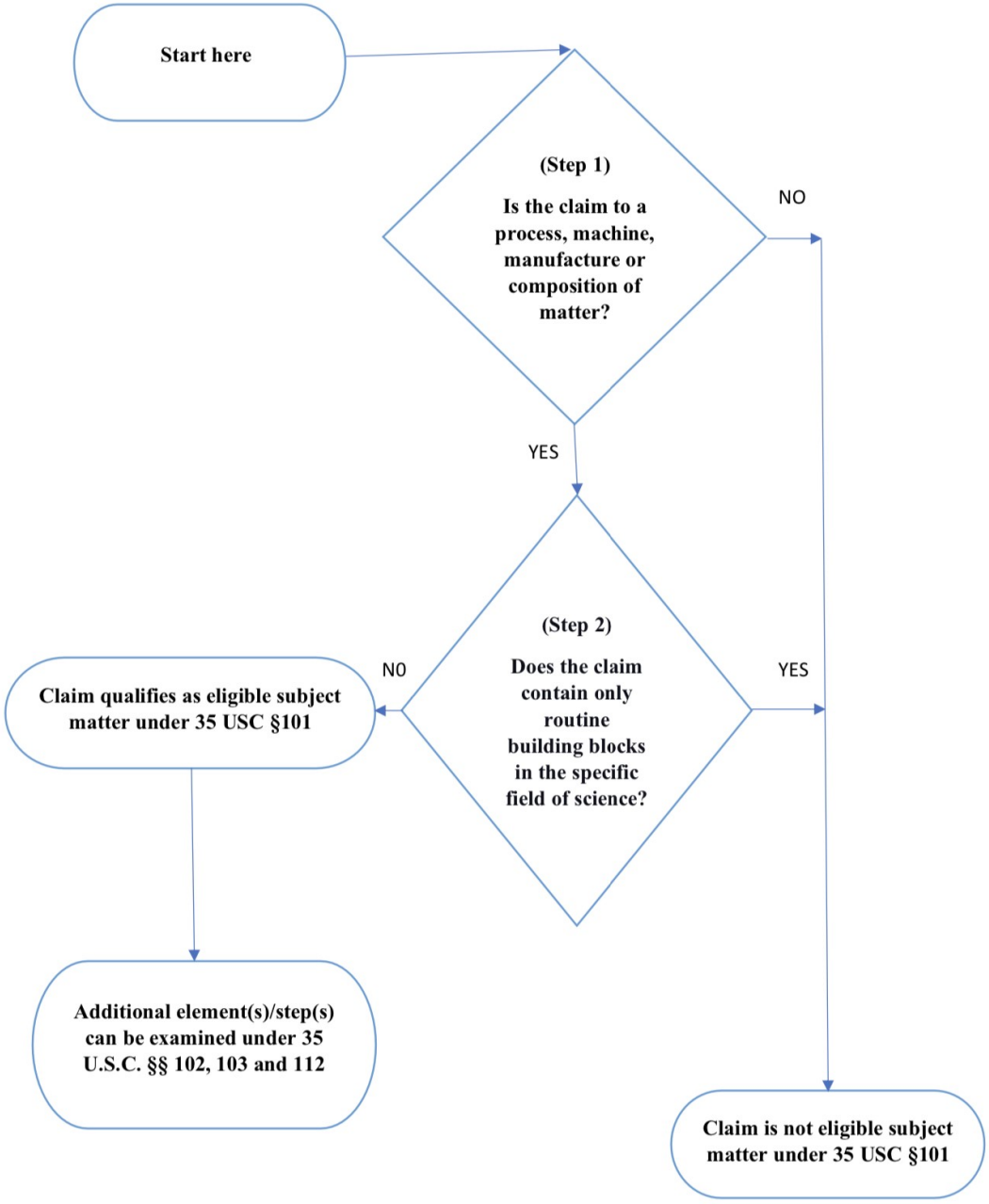
As illustrated, the test of routine building blocks was applied to *Berkheimer* and, not surprisingly, after all the filtrations and removals from consideration, nothing was left to be patentable, and no relief can be provided to him. Each claim and combination of all claims in the application was routine and conventional.[36] Therefore, Berkheimer did indeed wrongfully obtain a patent on a very basic idea of asset management systems.

1.3.2 Determining "anything more than Routine Building Blocks"

There is a possibility that in addition to the basic steps (routine building blocks), the inventor introduces new components or methods. If so, the claim passes the § 101 threshold. The reason for passing the § 101 threshold is because there is something in the claim that is more than routine building blocks of the specific type of the software. Then, I propose that the additional elements should be examined under §§ 102, 103, and 112 to see if they qualify for granting a patent. The following flowchart (FIG.3) illustrates my patent eligibility test proposal and its relationship with other patentability provisions:

[36] Berkheimer v. HP Inc., 881F.3d1360 (Fed. Cir. 2018) ("The question of whether a claim element or combination of elements is well-understood, routine and conventional to a skilled artisan in the relevant field is a question of fact. Any fact, such as this one, that is pertinent to the invalidity conclusion must be proven by clear and convincing evidence.") (quoting *Microsoft Corp. v. i4i Ltd.* P'ship, 564 U.S. 91, 95 (2011)).

Figure 3: Proposed Patent Eligibility Test in Relation with other Patentability Provisions



As an illustrative example for adding a new step to the initial building blocks of an invention, if we assume the steps of searching, choosing, paying and receiving the product as building blocks of the online shopping process, any additional steps or any specific improvement or creativity in

the shopping routine steps may pass the patent eligibility criteria. For example, if someone adds the feature or step "Undo your purchase" in online shopping, it goes beyond the enumerated routine building block steps. It means by eliminating this step, the primary functionality of the online shopping is remaining. Therefore, the online shopping invention that has this limitation (Undo your purchase) passes the § 101 threshold. Then the novelty or non-obviousness and sufficiency of the disclosure of this additional element should be examined under §§ 102,103 and §112. In another example, adding a limitation in the searching step of the online shopping (which is one of the routine building block steps) makes it patent-eligible under § 101. For example, assuming the routine building block of searching in online shopping is searching one database for generating an appropriate purchase order, the claimed invention that searches between more than one database for the specific purpose of matching products to customer queries passes the § 101 threshold. Determining the degree of inventiveness, obviousness, or novelty of that specific searching method/algorithm can be examined under established patentability provisions such as §§ 102 and 103.[37]

1.3.2.1 Determining "anything more than Routine Building Blocks" : *Berkheimer case*

There is no additional component(s), element(s) or step(s) in Berkheimer's invention other than routine building blocks. Features such as "workflow, version controlling system, and reporting system" are out of the scope of Berkheimer's claimed invention. They are some independent systems that are integrated into this system, and all of them are found in the prior art. Therefore, he should not have explained these features in his application in detail.

Berkheimer's invention can be summarized into "Many-to-one and One-to-many relationship": by applying specific system design and rules on a pile of data/documents (many), a system or model, here, object-oriented system model (one) was created to serve and organize new input documents (many) in the system. Here, only a conventional expression was used to explain how the system components are related. Therefore, his argument[38], refusing to accept claim one as a representative, and asserting that dependent claims 4–7 include inventive limitations ("reducing redundancy and enabling one-to-many editing",) is not legitimate. None of his dependent claims "transform the nature of the claim into a patent-eligible application."[39] Therefore, there was no additional component(s), element(s) or, step(s) other than Routine Building Blocks residing in his claimed invention or the invention as a whole.

Since *Berkheimer*'s invention does not have anything other than routine building blocks, the relationship between my patent eligibility test and other patentability provisions cannot be demonstrated. Therefore, to properly demonstrate the full applicability of my proposed test, I will apply my patent eligibility test on the following software patent cases, *Enfish, LLC v. Microsoft Corporation, et al.*[40] and *McRo*, *Inc*. *v*. *Bandai Namco Games America*[41]*:*

---

[37] Menell *et al.*, *supra* note 28.
[38] *Berkheimer v. HP Inc*., 881F.3d1360 (Fed. Cir. 2018)
[39] *Ibid*. (quoting *Mayo Collaborative Servs. v. Prometheus Labs., Inc*., 566 U.S. 66, 78–79 (2012)).
[40] Enfish, LLC v. Microsoft Corp., 822 F.3d 1327, 2016 U.S. App. LEXIS 8699, 118 U.S.P.Q.2D (BNA) 1684
[41] McRO, Inc. v. Bandai Namco Games Am. Inc., 837 F.3d 1299, 1302, 2016 U.S. App. LEXIS 16703, *1, 120 U.S.P.Q.2D (BNA) 1091, 1093

1.4 Applying Patent Eligibility Test on Two software patent cases

1.4.1 *Enfish, LLC v. Microsoft Corporation*

The U.S. Patent No. 6151604 ("the '604 patent") and U.S. Patent No. 6,163,775 ("the '775 patent") are directed to a self-referential model for a computer database, "a method and apparatus configured according to a logical table having cell and attributes containing address segments."[42] The *Enfish*[43] invention's abstract is as follows:

> The information management and database system of the present invention comprises a flexible, self-referential table that stores data. The table of the present invention may store any type of data, both structured and unstructured, and provides an interface to other application programs. The table of the present invention comprises a plurality of rows and columns. Each row has an object identification number (OID) and each column also has an OID. A row corresponds to a record and a column corresponds to a field such that the intersection of a row and a column comprises a cell that may contain data for a particular record related to a particular field, a cell may also point to another record. To enhance searching and to provide for synchronization between columns, columns are entered as rows in the table and the record corresponding to a column contains various information about the column. The table includes an index structure for extended queries.[44]

To analyze the *Enfish* invention and investigate its proposed features, we need to review the following common concepts and definitions in conventional relational database systems:

Database: "A database is a collection of information in tables that are organized in terms of rows and columns so that it can easily be accessed, managed, and updated."[45]

Relational Database

In a relational database, each table has a primary key that "uniquely identifies a record in the table"[46] and could act as a foreign key in another table. Through these foreign keys (which are like primary keys, but with different tasks[47]), different tables can interconnect. In the illustration below (FIG 4)[48], "Customer No" is a primary key in the "Customers Table", which at the same time is used as a foreign key in the "Orders" table to connect "Customers Table" to the " Orders" table. Table "Orders" by itself has "OrderNo" as its primary key, which is used as a foreign key to connect with other tables.

---

[42] "U.S. Patent No 6,163,775 (issued Dec 19, 2000)" http://patft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6163775.PN.&OS=PN/6163775&RS=PN/6163775 (last visited May 25, 2020).

[43] Enfish, LLC v. Microsoft Corp., *supra* note 40.

[44] "U.S. Patent No 6,163,775" *supra* note 42.

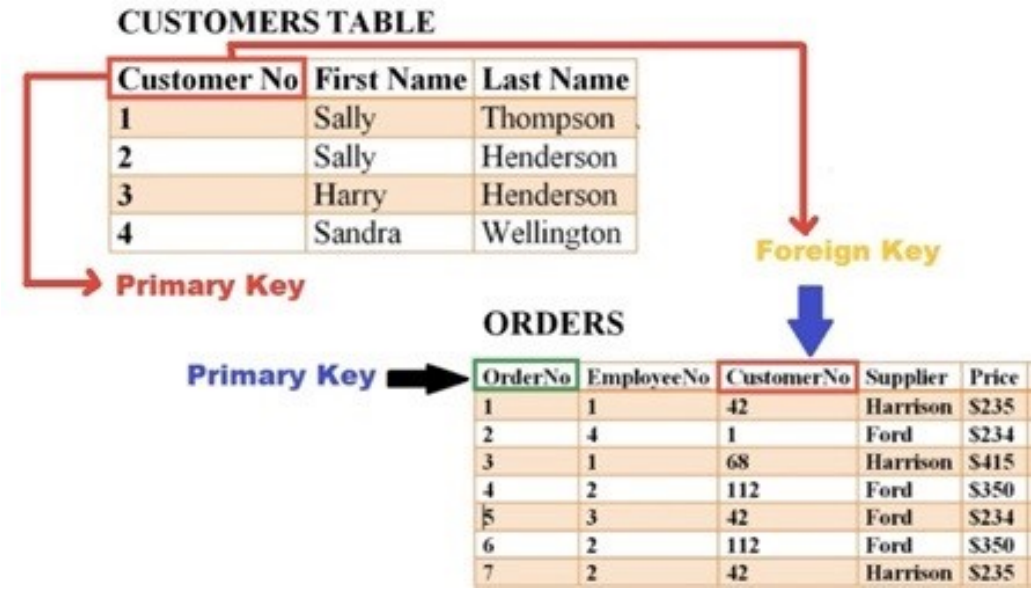
[45] Margaret Rouse, *database (DB),* TechTarget, https://searchsqlserver.techtarget.com/definition/database (last visited May 25, 2020)

[46] Shailendra Chauhan, *Difference between Primary Key and Foreign Key*, DotNetTricks, https://www.dotnettricks.com/learn/sqlserver/difference-between-primary-key-and-foreign-key (last visited May 25, 2020)

[47] *Ibid*.

[48] Figure resource: https://medium.com/analytics-vidhya/programming-with-databases-in-python-using-sqlite-4cecbef51ab9 (last visited May 25, 2020)

Figure 4: Primary key and the Foreign key in Database



A user of the database could apply a query and obtain a view of the database that fits the user's needs. For example, a manager might like a view or report on the list of the names of the customers. An accountant of the same company could, from the same tables, obtain a report on the list of the suppliers.

Self-referential table

A self-referential table is a table in which there is one or more fields or columns pointing to the records from the same table. All entities in this invention, in contrast to a relational model, are accumulated in one table and the logical relationships between the elements of these tables (rows and columns) are established by pointers (OIDs or Object Identifications). Also, the self-referential model can define the "table's columns by rows."[49]

The following illustration (FIG.5) [50] provides an example of how the tables in a relational database are accumulated in one table to form a self-referential table (FIG.6)[51]:

[49] "U.S. Patent No 6,163,775" *Supra* note 42 .
[50] *Ibid.*
[51] *Ibid.*

Figure 5: Relational database tables

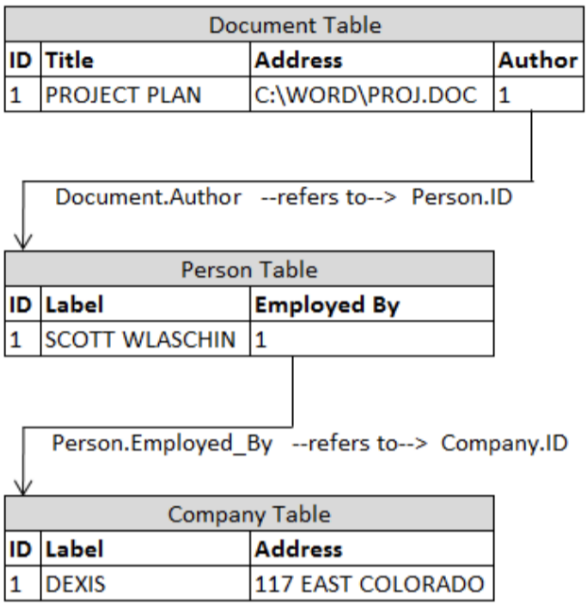


Figure 6: Self- Referential Table

SELF-REFERENTIAL TABLE						
ID	**Type**	**Title**	**Label**	**Address**	**Employed By (#4)**	**Author**
#1	DOCUMENT	PROJECT PLAN		C:\WORD\PROJ.DOC		#2
#2	PERSON		SCOTT WLASCHIN		#3	
#3	COMPANY		DEXIS	117 EAST COLORADO		
#4	FIELD		EMPLOYED BY			

In the above table, for example, the "Author" column is pointing to a record with ID #2. This means that the table is referring to itself. This specific table architecture, with its self-referential innovative aspect will be explained in greater detail in the following section.

Summary of the invention

The system used in this invention can be divided into two main related parts:

1- Backend: comprised of "Table structure" (data storage) and the "Indexing system," that consists of the "Knowledge base and Thesaurus system."

2- User Interface: comprised of the "Word processor system."

1- Backend components

a. Table structure

"The information management and database system of the present invention comprises a flexible, self-referential table that stores data...The table of the present invention may store any type of data, both structured and unstructured"[52]

b. Indexing system

The indexing system in any database helps to enhance the search process like the book indexing system that enhances the searching in a book. There are some conventional methods of indexing[53] in the database field, but the invention suggests the new indexing system - the self-referential indexing system - which is new and not similar to any of the existing indexing systems. In this invention, the new indexing system consists of the following two steps:

i. Key phrase extraction system

"To generate a sorted list of keywords, the system must first extract the key phrases or words from the applicable cells."[54] There are four different key phrase extraction systems available: full text extraction, column extraction, automatic analysis extraction and manual selection extraction. "In full text extraction, every word is indexed, which is typical for standard text retrieval systems. In column extraction, the whole contents of the column are indexed which corresponds to a standard database system. According to a third type of extraction, automatic analysis, the contents of the text are analyzed, and key phrases are extracted based on matching phrases, semantic context, and other factors. Finally, in manual selection extraction, the user or application explicitly marks the key phrase for indexing."[55]

Referring to the application claims in *Enfish*, "In this invention, the combination of structured information and text allows various combinations of key phrase extraction to be used."[56]Therefore, all four extraction systems were applied to the invention.

---

[52] "Structured data vs. unstructured data: *structured data* is comprised of clearly defined data types "such as primary data types like string, integer, etc. that "whose pattern makes them easily searchable; while *unstructured data* – "everything else" – is comprised of data that is usually not as easily searchable, including formats like audio, video, and social media postings." https://www.datamation.com/big-data/structured-vs-unstructured-data.html

[53] Some conventional indexing systems are as follows: Dense Indexing, Sparse Indexing, Single-Level Indexing including (Primary indexing, Clustered indexing, and Secondary indexing), and Multilevel Indexing including (B Tree and B+ Tree) https://www.datacamp.com/community/tutorials/introduction-indexing-sql.

[54] "U.S. Patent No 6,163,775" *supra* note 42.

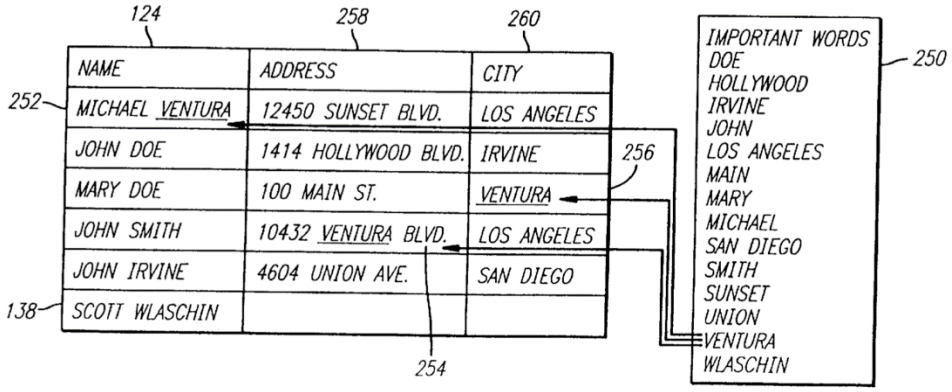
[55] *Ibid.*

[56] *Ibid.*

ii. Indexing the extracted text

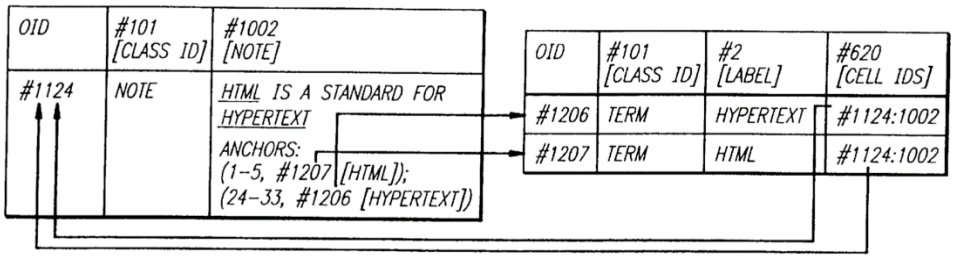
The extracted key phrases from the previous step are stored in the important words list. Any keywords that occur in any cells are saved into the important words list with the pointer to the cells where this keyword resides. For instance, as illustrated below[57] (FIG.7), the word "Ventura" occurs in different cells and this word is saved into the keyword list with the pointer to the cells that this keyword resides.[58]

Figure 7: The correspondence between cells of the table and a stored key word index



Similarly, the record containing that specific key word has a pointer to the same keyword in the important words list. Therefore, "the associations between the list of records with text and the list of key phrases is two-way since the cells that include text point to the key words."[59](FIG.8)[60] illustrates this two-way relationship, "each record can point to multiple key phrases, and each key phrase can point to multiple records." For example, as illustrated below, the record #1124 in the main table has a pointer to record #1206 and #1207 for 'HTML' and 'HYPERTEXT' key words in the key phrases table. Note that all these tables reside in one table (table 100), and only for the purpose of illustration, the tables are showed separately.[61]

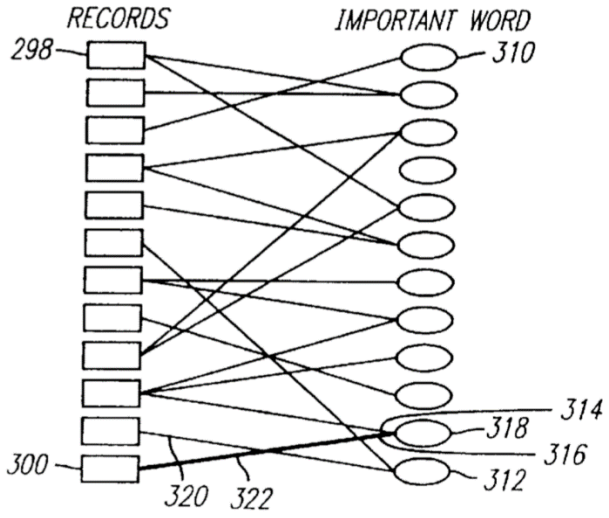
Figure 8: The relationship between certain data records and key word index records



Therefore, as illustrated below (FIG.9), there are many-to-many relationships between the record containing that specific key word and the same keyword in the important words list.

[57] *Ibid.*
[58] *Ibid.*
[59] *Ibid.*
[60] *Ibid.*
[61] *Ibid.*

Figure 9: The relationship of Figure 8 in graphical form



c. How data is retrieved based on the key phrases

"The present system uses a new approach (Hypertext system) based on a dynamic association between records. Hypertext systems typically associate a region of text with a pointer to another record. In the preferred embodiment, each hypertext region is associated with a key phrase, not a normal record." When a user clicks on a hypertext, all the records associated with the key phrase are retrieved and ranked using any associative search techniques. "If the target record is absent, the hypertext jump will fail. The application can then display on the display screen either the highest ranked item [more relevant words or concepts] or present all the retrieved items and allow the user to pick the one to access."[62]

d. Knowledge Base and Thesaurus System

The invention includes a knowledge base or thesaurus system to improve the indexing system capabilities and enhance searching.[63] The point is that the database system, including this knowledge base and thesaurus system, is stored in one table. Key phrases (terms) and concepts are stored as records in this system.[64]

The invention, by weighing the relevancy between the concepts and key words in a table, provides a knowledge base or thesaurus system. The keywords (terms) and the relevant words with the same concepts are connected through pointers (OID's). For example, the key phrase "IBM" is related to the concept "IBM PC" with an assigned weight of 100%, where the weight percentage reflects the relevancy and similarity between the initial term "IBM" and the related concept "IBM PC." Also, the term "IBM" is related to the concept of "Microsoft" with a weight of 70%, "where the weight percentage reflects the relevancy and similarity between the initial term "IBM" and the related concept 'Microsoft.'[65]Here, "each important term (IBM) included within the thesaurus

[62] *Ibid.*
[63] *Ibid.*
[64] *Ibid.*
[65] *Ibid.*

contains a pointer to a `concept` record [here, the concept of "Microsoft"], and to the terms that are included within the bounds of that concept [e.g. computer companies]."[66]

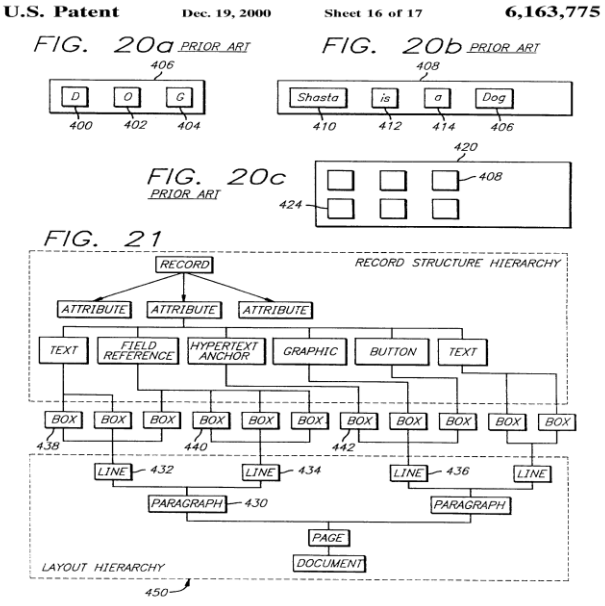
2. User Interface: Word processor system

"The structured word processor of the present invention uses the 'boxes and glue' paradigm. Boxes may be attached to other boxes with 'glue.'"[67] According to this paradigm, a page of text is created by starting with individual characters and concatenating characters (each character considers as a box) to form a word, concatenate (glue) words to form lines, concatenate (glue) lines to form a paragraph, and concatenate (glue) a paragraph to form a page of a text, and eventually to form documents.[68]

> When a record of the table is edited, each word and field definition is converted into boxes. The system organizes these boxes into a tree structure of line boxes and paragraph boxes. Each box is kept as a bitmap... so, the system displays the tree structure by displaying all of the bitmaps corresponding to the boxes in the tree. To edit the tree structure, a user may click a cursor on a part of the text…The system locates the word box or glue that is being edited by a recursively descending through the tree structure. If the tree is changed for example by adding a new word, only the new word box and a relatively small number of adjacent boxes need to be recalculated.[69]

This word processor system is one of the inventive steps in the invention that makes the invention unique. In the following, the word processing system is illustrated[70] (FIG.10).

Figure 10: Structure of the word processor of the present invention
(20 a: A Character and word box structure, 20 b: The word and horizontal and the horizontal line box structure, 20c: The vertical box structure)



[66] *Ibid.*
[67] *Ibid.*
[68] *Ibid.*
[69] *Ibid.*
[70] *Ibid.*

1.4.1.1 Routing Building Block Determination: *Enfish, LLC v. Microsoft Corporation, et al.*

The routine building blocks of the *Enfish* case are as follows:

1- **Converting the external resources to plain text:** This step is considered a routine building block since, to access the data, we need to convert it to a standard format that is understandable for the system.

2- **Having a data structure (table):** This step is also considered a routine building block because the extracted data should be stored in a data structure. The data structure should have the capability of storing, editing, and deleting the data.

3- **Designing a User Interface (UI):** This step is also considered a routine building block. For manipulating the data in a document (editing, adding, and deleting), the user and database must be interacting with each other via UI.

4- **Retrieving data with indexing system:** This is a routine building block as well because after you extract the data from the document and store it in data storage, you should be able to retrieve the data through a search system equipped with the indexing system to make it more efficient.

In this invention, the **unique indexing method**, and the **word processor system** are something more than these routine building blocks, which take the invention out of §101 threshold.

1.4.1.2 Determining "anything more than Routine Building Blocks" : *Enfish case*

The *Enfish's* invention adds improvements to some of the building blocks of the system, which takes the invention out of §101 threshold. These additional features occur in both backend (table structure and the Indexing system consist of the knowledge base and thesaurus system) and the User Interface (which comprises the "Word processor system") of the system.

In the backend, the **unique indexing method** is something more than routine building blocks. The uniqueness of this indexing system is its self-referential feature and its knowledge base and thesaurus system. This knowledge base and thesaurus system resides in the indexing system, which makes it different from the basic and conventional indexing systems. This knowledge base and thesaurus system, equipped with the filtering and weighting features, make the search and retrieval system unique. In this invention, the knowledge base system is created based on the key phrases extracted from the table records. The weighting feature is for ranking the relevancy of the retrieved record.

> The present invention includes a thesaurus and knowledge base that enhances indexed searches. The thesaurus is stored in the table and allows a user to search for synonyms and concepts and also provides a weighting mechanism to rank the relevance of retrieved records.[71]

Another additional feature is the **word processor system** residing in *Enfish's* invention. This word processor system, with its "Boxes and Glue" model and bitmap data structure, combines data (structured and unstructured) with different data types (text, graphic, hyperlink, number, etc.).

[71] *Ibid.*

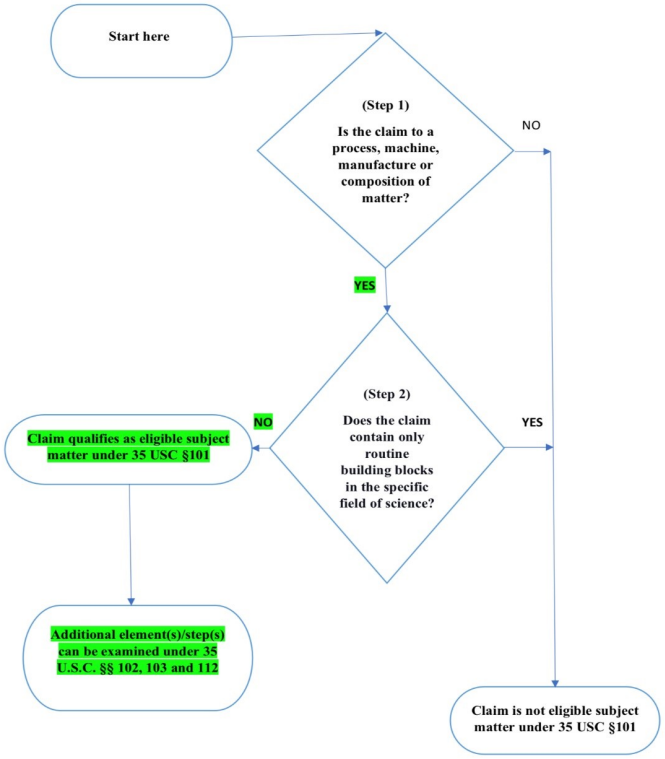
Then the system stores them in one cell of the table. In the conventional version of data storing, only one plain data type value can be stored in one cell of the table.

A bitmap representation in the tree structure of the display system provides a user interface in which the data can be edited. " The novel word processor of the present invention is integrated with the table of the present invention to allow cells to be edited with the word processor." [72] This editing method makes the system unique since the user, by clicking on the specific point on the screen, can change the content of that specific spot. It happens without affecting the rest of the words, lines, or paragraphs. In this way, for example, by adding a new word in the line, "only the new word box and a relatively small number of adjacent boxes need to be recalculated. Similarly, line breaks or restructuring of a paragraph does not alter most of the word boxes, which may be reused, and only the line boxes need to be recalculated."[73] Only the edited characters or words need to be recalculated and saved after editing, not the whole record.

The Court of Appeals for the Federal Circuit in the *Enfish* case held that "the district court erred in finding all claims patent-ineligible under 35 U.S.C.S. § 101 because the claims were not directed to an abstract idea within the meaning of step one of the *Alice* analysis. Rather, they were directed to a specific improvement to the way computers operate, embodied in the self-referential table."[74]

Here, the novelty or non-obviousness and sufficiency of the disclosure of these additional elements should be examined under §§ 102,103 and §112. The following flowchart (FIG.11) is a visual demonstration of my patent eligibility test as it applies to the *Enfish* case:

Figure 11: Application of Proposed Patent Eligibility Test: *Enfish* Case



[72] *Ibid.*
[73] *Ibid.*
[74] Enfish, LLC v. Microsoft Corp., *supra* note 40.

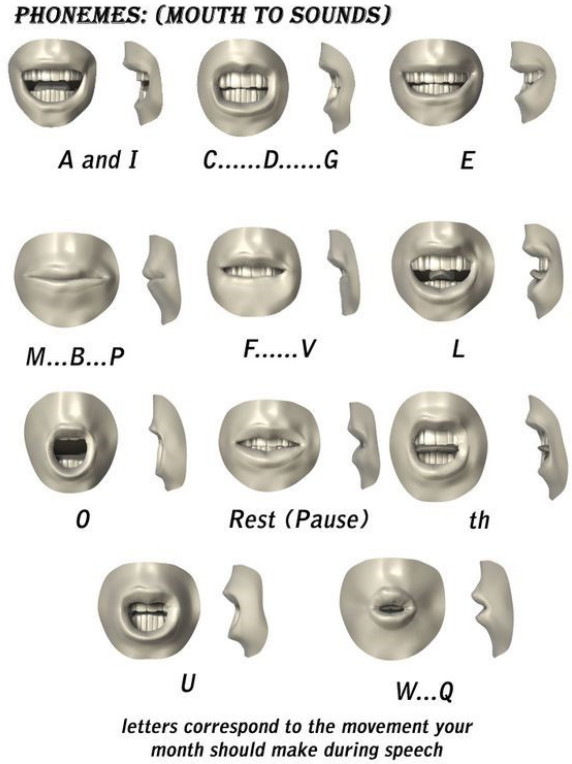
1.4.2 *McRo, Inc. v. Bandai Namco Games America*[75] *(McRo Case)*

"The U.S. Patent Nos. 6,307,576 (''the '576 patent'') and 6,611,278 (''the '278 patent'') were both issued to Maury Rosenfeld and are both titled 'Method for Automatically Animating Lip Synchronization and Facial Expression of Animated Characters.'"[76] In the following, after determining some definitions related to computer animation, I will apply the proposed patent eligibility test on the *McRo* case.

Definitions

Phoneme: "A "phoneme" is defined as the smallest unit of speech, and corresponds to a single sound such as "th," "aah," "ee,", "oh", etc. The goal of lip synchronization animation is to match the character's lip and facial expression to spoken dialogue."[77] (FIG.12)[78]

Figure 12: Mouth positions for Phonemes

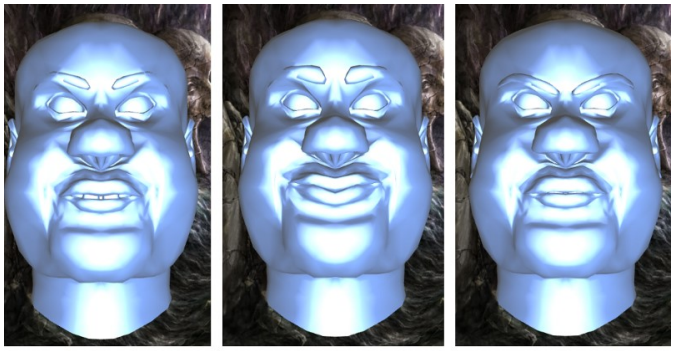


Viseme or morph target: "A viseme is an image of a face speaking a certain phoneme. In computer animation, the artist starts with the neutral model and then creates a library of visemes showing what that character looks like when speaking the various sounds [or phonemes]."[79] "Accordingly, a morph target has the same number of vertices as the neutral model, and each

[75] McRO, Inc. v. Bandai Namco Games Am. Inc., *supra* note 41.
[76] https://www.bitlaw.com/source/cases/patent/MCRO.html
[77] Reply brief of Petitioner McRO, Inc, McRO, Inc. v. Bandai Namco Games Am. Inc.(docket number 755) (2016)(Nos. 15-1080, 15-1099). Retrieved from https://www.law.berkeley.edu/wp-content/uploads/2016/05/McRO-v-Bandai-Namco-appellate-argument-FJC-Program.pdf
[78] Figure resource: https://images.app.goo.gl/teRwSUTgC2vfnpH3A. (last visited Mar.6, 2020)
[79] *Supra* note *77.*

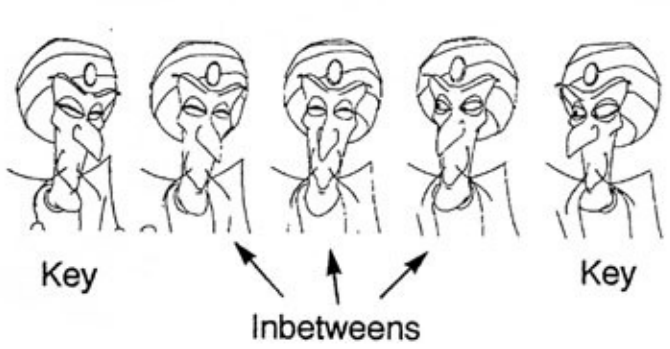
vertex on the morph target corresponds to a vertex on the neutral model."[80] "A morph target [FIG.13] [81] is created by manipulating the vertices of the neutral model to form a different facial shape."[82]

Figure 13: Morph targets



Keyframe: The keyframe (FIG.14)[83] is the important moment in an animation sequence where the motion is at its extreme. It means at this moment the specific phoneme is pronounced and its corresponding morph target is displayed.[84]

Figure 14: Keyframes

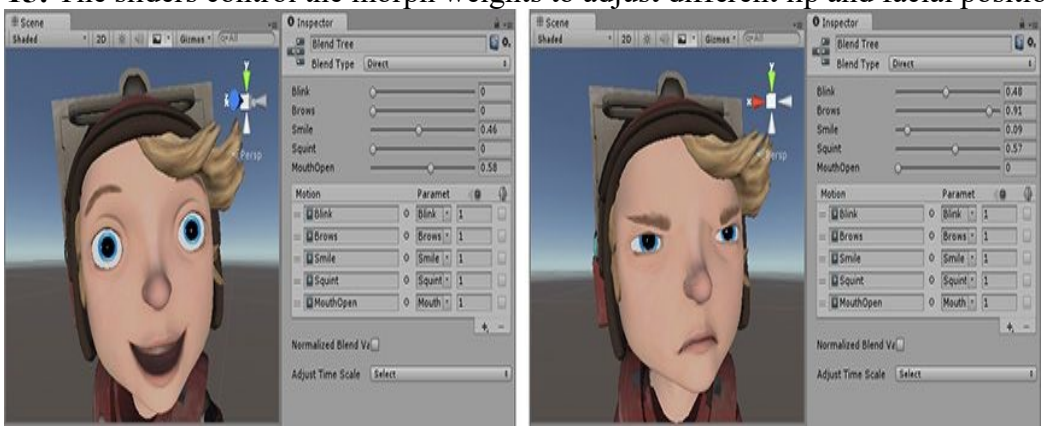


Morph weight: "A morph weight is a value, between 0 (0%) and 1 (100%) that is applied to determine how far the neutral model's vertices should be moved toward the morph target. A morph weight represents the degree to which the lip or facial shape corresponding to the morph target is expressed. For example[85], if the morph target for the "oh" viseme is assigned a morph weight of 1, the neutral model would be modified to look exactly like the "oh" viseme, but if the "oh" viseme is assigned a morph weight of .5, the neutral model's vertices would be moved only halfway toward the "oh" viseme."[86] In the prior art, "Manual approach", the artist uses a slider to assign a number between 0 and 1 as a morph weight to determine the degree to which the lip should

[80] *Ibid.*
[81] Figure resource: https://images.app.goo.gl/QE12sb9a7roF5MEy5 (last visited Mar.6, 2020).
[82] *Supra* note *77.*
[83] Figure resource: https://images.app.goo.gl/WQ4qPcm6rjjj4ega7 (last visited Mar.6, 2020).
[84] *Supra* note *77.*
[85] *Ibid.*
[86] *Ibid.*

be moving. In the illustration below, (FIG.15)[87] the artist uses the slider to adjust different lip and facial positions:

Figure 15: The sliders control the morph weights to adjust different lip and facial positions



In the second approach in the prior art, which is more automated, "each viseme is given a morph weight of 1 or 0, with nothing in-between, the viseme is either fully expressed or not expressed at all. The software then uniformly interpolates between those keyframes to generate an image at each frame of the animation."[88]

Morph weight set: "A morph weight set is a set of values in which each entry represents the weight of one of the morph targets (visemes). As the district court explained, when a morph weight set is "applied" to the neutral model, it "transform[s]" the model to a particular facial expression."[89]

Setting a keyframe: "Assigning a morph weight set to a specific time in the animation is called "setting a keyframe."[90]

TAPT: "Time aligned phonetic transcriptions (TAPTS) are a phonetic transcription of a recorded text or soundtrack, where the occurrence time of each phoneme is also recorded.[91]For example, a TAPT representing the spoken word "hello" could be configured as it is illustrated in table 1[92]:

Table 1: Time aligned phonetic transcriptions (TAPTS)

Time	Phoneme:
0.0	silence begins
0.8	silence ends, "h" begins
1.0	"h" ends, "eh" begins
1.37	"eh" ends, "1" begins
1.6	"1" ends, "oh" begins
2.1	"oh" ends, silence begins.

[87] Figure resource: Internet.
[88] *Supra* note *77.*
[89] *Ibid.*
[90] *Ibid.*
[91] "U.S. Patent No 6,307,576(issued Oct. 23,2001)." http://patft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6307576.PN.&OS=PN/6307576&RS=PN/6307576
[92] Table resource: *Ibid*.

Morph weight set stream: This term, which is used in this new invention, is a sequence of morph weight sets, each set enclosed into a time frame. Morph weight set stream works as a function of phoneme sequence and time of said phoneme sequence.

As table 1 shows, for example, at the 1.6 seconds of animation, the "l" phoneme ends, and the "oh" phoneme begins. This new invention is using TAPT to create an output morph weight set stream.

As illustrated in table 2[93], for example, at the 0.8 and 0.98 second, the ("h") phoneme should be pronounced. The correspondence morph weight set for ("h") phoneme is 100000. At the 1.037 and 1.333 second (keyframe) of animation, the "eh" phoneme should be pronounced. The correspondence morph weight set for a ("eh") phoneme is 010000. The collection of these morph weight sets along with their associated times (keyframes) will form the morph weight set stream.

Table2: Output morph weight set stream

Time	D.S.1 ("h")	D.S.2 ("eh")	D.S.3 ("l")	D.S.4 ("oh")	D.S.5 (aux"oh")	D.S.6
0.0	0	0	0	0	0	0
0.78	0	0	0	0	0	0
0.8	1	0	0	0	0	0
0.98	1	0	0	0	0	0
1.037	0	1	0	0	0	0
1.333	0	1	0	0	0	0
1.403	0	0	1	0	0	0
1.667	0	0	1	0	0	0
1.74	0	0	0	1	0	0
2.1	0	0	0	1	0	0
2.14	0	0	0	0	0	0

In the new invention, this binary sub-sequence (morph weight set), which represents a phoneme, is assigned to the corresponding graphical representation (morph target). The timed Sequence of these morph weight sets (morph weight set stream) is then sent to the graphic section to map to the morph targets in animation characters. In the following, the morph weight set stream is illustrated (FIG.16)

Figure16. Morph Weight Set Stream

Keyframe morph weight set

Time	("h")	Time	("h")	Time	("eh")	Time	("eh")
0.8	100000	0.98	100000	1.037	010000	1.333	010000

[93] Table resource: *Ibid*.

Meaning of Sequence of 0 and 1:

In computer science, we can assume that "1" means on or the existence of something and "0" means off or a lack of something. Here in this invention, this convention is used to show if a phoneme exists or not. It means that we put a number one under the phonemes, which are supposed to show at the specific time, and we should use zero wherever we do not want the pronunciation of those phonemes. As a result, we can show the number of phonemes that are pronounced by characters by creating a stream of 0 and 1. For example, in the third and fourth line from table 2, the 1 0 0 0 0 0 stream means that the "h" phoneme is pronounced at 0.8 seconds and continues until 0.98 seconds. Considering the fifth and sixth line and the 0 1 0 0 0 0 stream, we can see that at 1.037 seconds of animation the" eh" phoneme is pronounced and continues until 1.333 seconds.

Time Aligned Emotional Transcriptions (TAET):

TAET corresponds to facial models of emotions (surprise, anger…), which can be implemented by creating an additional morph weight set stream to represent the special emotional state.

> A further extension of the present method is to make a parallel method or system that uses time aligned emotional transcriptions (TAET) that correspond to facial models of those emotions. Using the same techniques as previously described, additional morph weight set streams can be created that control other aspects of the character that reflect the facial display of emotional state.[94]

The present method is extensible by adding new rules that use Time Aligned Emotional Transcriptions (TAET) that correspond to the facial models of different emotions like "surprise," "disgust, "embarrassment," "timid smile" etc. This will affect the output data (morph target) according to additional rules (emotional rules which are converted to the sequence of 0 and 1) that take these emotemes into account. Each rule comprises criteria and function, as in an "if . . . then . . . else" construct.

> In addition, the TAET data can be used in conjunction with the lip synchronization secondary rules to alter the lip synchronization output stream. For example, Criteria: An "L" is encountered in the TAPT (Time Aligned Phonetic Transcription), and the nearest "emoteme" in the TAET is a "smile." As a result, several parallel systems may be used on different types of timed data, and the results concatenated."[95]

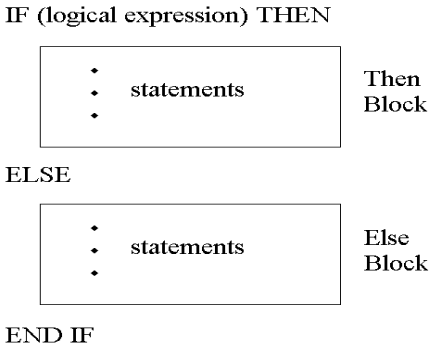
Rule (If … Then … Else):

In conventional programming languages, there are several conditional control structures. One of them is an If…Then…Else flow controlling feature. By applying these conditional structures or rules, we can control the flow of the program and determine under which condition (criteria) things should be done (function). To explain how this feature works, consider the following schematic (FIG.17)[96]:

---

[94] *Ibid.*
[95] *Ibid.*
[96] Figure resource: https://images.app.goo.gl/9oqcwaSv6X7qoJrVA. (last visited Mar.6, 2020)

Figure 17. Rule (If … Then … Else)



In this new invention, this very popular programming feature, "if … then … else" is used to generate and apply the appropriate rules to automatically control and determine the animating lip synchronization and facial expression of three-dimensional characters for films, videos, cartoons, etc.

Each "Criteria" is equivalent to "If" clause, and each "Function" is equivalent to the "Then" clause in Pseudocode. To make this test case more readable, the inventor could also provide these samples of rules in the form of "If… Then….Else." For example, the default rules for the first two lines can be written as follows:

IF Encounter an "h" as in "house" **Then**
Use morph weight set (1,0,0,0,0,0) as transition target.

The equivalent Pseudo code format of this rule is as follows, which has been already drafted above in the Pseudocode section of this paper:

```
If p[i] == "h" then
        MorphWeight[i] = 100000;
```

In this rule, we check a condition in the "if" clause. If the condition is fulfilled, then the "then" clause will be executed; otherwise, the "else" condition will be performed. The "if "clause or criteria works as a gatekeeping. If the gatekeeping criteria are satisfied, then the program determines its proper rule to apply. The "if" section would be including the condition or criteria under which the specific phoneme should be pronounced in the "then" section. For example:

"Criteria or **if**: Encounter an "h" as in 'house.'"[97]

The "then" section would be including the appropriate sequence of 0 and 1 as a morph weight set to assign a phoneme to a specific morph target to represent the pronunciation of "h."
"Function or **then**: Use morph weight set (1,0,0,0,0,0) as transition target."[98]

In "else" clause(s) or other condition(s), the next phoneme(s) which should be pronounced are determined.
"Next Criteria or **else**: Encounter an "eh" as in "bet": Use morph weight set (0,1,0,0,0,0) as transition target."[99]

[97] *Ibid.*
[98] *Ibid.*
[99] *Ibid.*

In the following, different rule categories which are used in the present invention is explained:

Default, Secondary(auxiliary) and Post-processing rules:

"The rules of the present method may be categorized into three main groupings; default rules, auxiliary rules, and post-processing rules. The default rules must be complete enough to create valid output for any TAPT encountered at any point in the TAPT. The secondary rules are used in special cases; for example, to substitute alternative morph weight set correspondences and/or transition rules if the identified criteria are met. The post-processing rules are used to manipulate the morph weight set stream further after the default or secondary rules are applied, and can further modify the members of the morph weight sets determined by the default and secondary rules and interpolation."[100]
Default rules can be generated as required to represent different lip and facial positions. Secondary and post-process rules are complementary rules that can be included in the emotional state of different characters. The following example shows how, and by which order, the default, secondary, and post-processing rules are executed. Note that "If a timed phoneme or sub-sequence of timed phonemes do not fit the criteria for any of the secondary rules, the default rules are applied" [101]

In the following example, the default rule condition (Criteria_1) should be checked, but nothing happens until the software checks the secondary rule condition (Criteria_2). If the animation has a secondary rule, then the secondary rule function (Function_2) will be executed; otherwise (Else_2), the software runs the function of the default rule (Function_1). This means that the software first checks in the sequence of rules whether the secondary rule condition exists or not. If the secondary rule's condition is true, the software runs the secondary rule function; otherwise, it performs the default rule function in the "else" section. This happens because the secondary rule is a complementary rule of the default rule and expresses it in a specific way. "For example, if a character says "HELLOOOOOOO!" then a more exaggerated "oh" model,"[102]as a secondary rule, should be executed. This rule is complementary to the default rule about the "oh" phoneme.

"Post-processing rules can be applied before interpolation or after interpolation"[103], and like the default and secondary rules, they generate a stream of 0 and 1 as a morph weight in order to produce a specific auxiliary morph target. "In post-processing rules, for example, the auxiliary morph target might be used in a cyclical manner over time. For instance, if the auxiliary morph target had the character's mouth moved to the left periodically, the output animation would have the character's mouth cycling between centers to the left as he spoke."[104]This approach can be extended by generating more default rules, secondary rules, or post-process rules to get more and diverse lip and facial positions. All of these rules are essentially extendable.

[100]*Ibid.*
[101]*Ibid.*
[102]*Ibid.*
[103]*Ibid.*
[104]*Ibid.*

Default rule: If (Criteria) _1: Encounter an "oh" as in "old"
Secondary rule: If (Criteria) _2: Encounter an "oh" longer than 1 second long.
Then (Function) _2: Insert transitions between a defined group of morph weight sets at 0.5-second intervals, with transition duration's of 0.2 seconds until the next "normal" transition start time is encountered. (0100000)

Else (Otherwise) _2
Then (Function) _1: Use morph weight set (0,0,0,1,0,0) as transition target.
Post-processing rule: If (Criteria) _3: Auxiliary morph target had the character's the mouth moved to the left cyclically
Then (Function) _3: output animation would have the character's mouth cycling between centers to left as he spoke.

Note: In this new invention, the morph weight set corresponding to each phoneme are created based on 7-bit binary codes in which each bit can be represented by two states of 0 and 1. As a result, in total, we can have the maximum number of 2*2*2*2*2*2*2= 2^7=128 different facial and lip positions. By increasing the number of bits, it is possible to increase the number of different positions in the lips and face of the actors in animation. In earlier inventions, this extension could not have been accomplished easily, because instead of each new face or lip position, the artist had to calculate a new delta set as a morph weight and then assign it to the new lip or facial position"

Interpolation:

It is a mathematical term, which is "an estimation of a value within two known values in a sequence of values."[105] This method, which is used in the prior art manual approach, was applied to show a state or value between two keyframes and creating the morph weight and eventually corresponding morph target for that value to make the animation more natural. "Because interpolation generates the property values between keyframes, interpolation is sometimes called "tweening.'"[106]

1.4.2.1 Routine Building Block Determination: *McRO, Inc. v. Bandai Namco Games Am. Inc* Case

The building block activities for computer animation is determined as follows:

1- Using a Time aligned phonetic transcriptions (TAPT)
2- Generate morph weight for each phoneme in TAPT
3- Interpolating between keyframes
4- Assigning the sequence of morph weights to the morph targets at the keyframes determined in TAPT

These are necessary procedures for any animation computer, with which the system functions for its primary purposes.

---

[105] Margaret Rouse, extrapolation and interpolation, TechTarget, https://whatis.techtarget.com/definition/extrapolation-and-interpolation, (last visited May, 2020)

[106] *About spatial and temporal keyframe interpolation,* Adobe, https://helpx.adobe.com/after-effects/using/keyframe-interpolation.html, (last visited Aug. 2016)

1.4.2.2 Determining "anything more than Routine Building Blocks": *McRo Case*

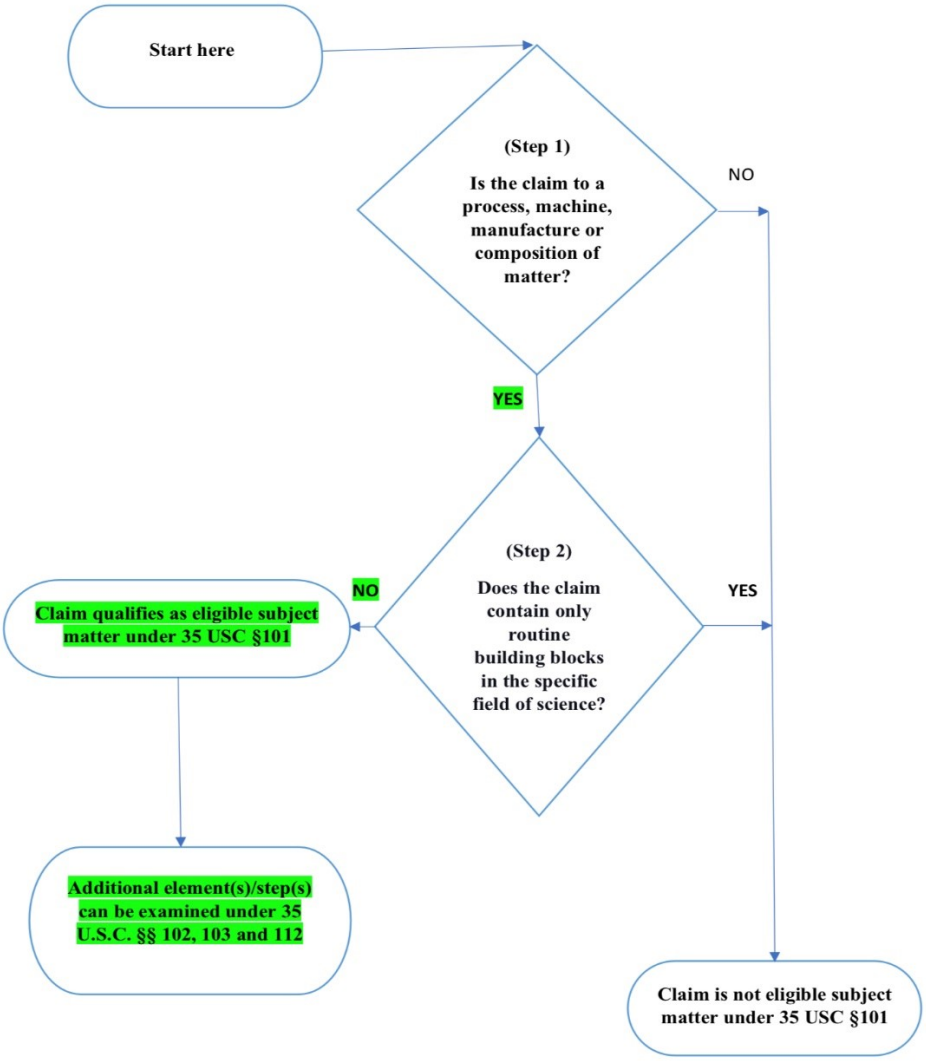
In the McRO case, the "**binary sub-sequence rule-based**" method, is a specific way for automatically animating the lip synchronization and facial expressions of 3D characters and has an "inventive concept" sufficient to transform the abstract idea into a patent-eligible application.

The Federal Circuit District held that: "In a patent infringement case involving patents relating to automating part of a preexisting 3-D animation method, the district court erred in finding that asserted patent claims were invalid as directed to patent-ineligible subject matter under 35 U.S.C.S. § 101 because the ordered combination of claimed steps, using unconventional rules that relate sub-sequences of phonemes, timings, and morph weight sets, was not directed to an abstract idea and was therefore patent-eligible subject matter under § 101."[107]

The following flowchart (FIG.18) is a visual demonstration of my patent eligibility test as it applies to the *McRo* case:

[107] McRO, Inc. v. Bandai Namco Games Am. Inc., 837 F.3d 1299, 1302, 2016 U.S. App. LEXIS 16703, *1, 120 U.S.P.Q.2D (BNA) 1091, 1093

Figure 18: Application of Proposed Patent Eligibility Test: *McRo* Case



1.5 Addressing some challenges in the field

1.5.1 The nature of the software vs. the breadth or scope of the invention

In my patent eligibility proposal, determining the routine building blocks of the software is based on the nature of the specific type of software, which is different from the breadth or scope of the invention. If the court or examiner decide that the invention contains only routine building blocks, and as such preempts all the inventions in that specific type, narrowing the breadth of the invention does not change the nature of the software. As an illustrative example, assume that the underlined invention is a data asset management system that does not have anything more than routine building blocks. Here, if the patentee claims to narrow the breadth of his invention by confining the applicability of his invention to retail stores only, for example, such a claim does not change the nature of the software. Determining the building blocks and determining the breadth of an invention are two completely separate matters.

Here, the patentee, by narrowing the breadth of his invention, claims that he limits the number of software programs that his invention covers - in other words, anything with the same structure that is used outside of the narrow field he has attempted to define would not be covered by this patent. This narrowly claimed invention is still patent ineligible because it is an abstract idea which includes only routine building blocks, the ***basic, primary, or minimum steps*** required for accomplishing the primary action for any database systems.

Therefore, as I mentioned before, the key question in determining the patent eligibility under § 101 is whether or not the patent covers only building blocks. Although the outcome of not allowing patents on building blocks may help prevent preemption, preemption is not the key question. The key question is preventing to patent building blocks.

1.5.2 Preemption rational

Although preemption has a fundamental role in the § 101 analysis and is regarded as the primary policy in determining patent eligibility,[108] it is not the essence of my test, but a secondary consideration.

<u>Preemption Type 1: Abstract Idea</u>

The first type of preemption concerns granting a patent on an abstract idea or routine building blocks. Based on my proposed eligibility test, after determining the routine building blocks or abstract ideas of the specific type of underlined software, if the invention has anything more than these routine building blocks, it passes § 101. If not, it will be rejected under § 101.

The reason why an abstract idea is not patent-eligible and would preempt other innovations is because the only embodiment is its fundamental components that are the building blocks common in all inventions in that type of software. Since these building blocks are high-level functions that exist in all inventions in that type of software, they are considered not patent eligible, and the claim has the risk of preempting future inventions in the same field.[109] Therefore, by granting a patent for these essential elements, other innovations are hindered from patenting. Thus, the problem lies not only in patenting these basic ideas that are mainly the sum of these basic

[108] Alice, 2014 U.S.134 S. Ct. at 2354.
[109] *CLS*, 717 F.3d at 1282 (Lourie, J., concurring), cited approvingly in *Accenture Global Services,* GmbH v. Guidewire Software, Inc., 728 F.3d 1336 (Fed. Cir. 2013).

essential elements, but also the potential harm in preventing others from protecting their novel and technological innovations.

Preemption Type 2: Broadness

The second type of preemption concerns granting a patent on a very broad claimed invention. This type of preemption involves when the "something more" mentioned above is too broad and preempts other innovation in the same field or any other fields. The problem here is that we do not know what the practical definition of broadness is and the extent to which it is applied. In my opinion, broadness can be divided into the following two categories:

1- Where the "something more" includes only *mathematical formulas* or *pure or general algorithms*, both will be rejected under § 101.

Mathematical formulas are considered laws of nature and are therefore too broad to be patent eligible. Viewed similarly as mathematical formulas, pure or general algorithms, such as "Decrease and Conquer algorithms," are also patent ineligible because they can be used in different applications and can solve a variety of problems. Thus, both are too broad and rejected under judicial exception of § 101.

2- Where the invention has something, anything more than building blocks, but it is *too broad*, it may still pass § 101.

If the particular method used is, so far, the only way of implementing the specific type of ideas in the same field, it should not be rejected under § 101 based on the preemption rational. The reason is because we do not know what will happen in the future, whether there are other ways of implementing the same functions of the claimed invention or not. Therefore, in my opinion, the courts or examiner should not invalidate the patent in this case. In his article titled" Alice on Software Patents: Preemption and Abstract Ideas," Robert Saches discusses:

> The "known unknown" preemption that we cannot determine because we cannot know what will happen in the future: whether the technology will be successful in the marketplace, whether others will adopt it, or design around it, or any other myriad factors that influence how "fundamental" an invention will be. A court or a patent examiner certainly cannot evaluate the level of "known unknown" preemption, and hence should not use speculations (or hand-waving) about this kind of preemption risk to invalidate a patent.[110]

Therefore, one way to determine that an invention does not preempt other inventions in that specific type of software is when the patentee can show other alternative methods for achieving the same result in his application. For example, in *McRO v. Bandai Namco*[111], the district court held that the claims are patent-eligible by stating that, "there are numerous ways to perform lip-synchronization animation, automated lip synchronization, and even automated, rules-based lip synchronization, that do not infringe the patents." Therefore, "it is clear that the patents do not seek to monopolize anything remotely resembling the 'building blocks of human ingenuity' or 'the

[110] Robert Saches, *Alice on Software Patents: Preemption and Abstract Ideas*,IPWatching, Jun. 29, 2014 https://www.ipwatchdog.com/2014/06/29/alice-on-software-patents-preemption-and-abstract-ideas/id=50210/

[111] McRO, Inc. v. Bandai Namco Games America Inc. (Fed. Cir. 2016) ("The U.S. Patent Nos. 6,307,576 (''the '576 patent'') and 6,611,278 (''the '278 patent'') were both issued to Maury Rosenfeld and are both titled "Method for Automatically Animating Lip Synchronization and Facial Expression of Animated Characters." The Federal Circuit held that "the ordered combination of claimed steps, using unconventional rules that relate subsequences of phonemes, timings, and morph weight sets, is not directed to an abstract idea and is, therefore, patent-eligible subject matter under § 101.")

basic tools of scientific and technological work.'"[112] However, even if the patentee could not show other alternative methods for achieving the same result in his application, it should still pass the § 101 threshold because we cannot expect that the inventor be so burdened to invent other possible inventions around his patent in order to prove that his invention does not preempt all other inventions in that specific field in the future.

Preemption Type 3: Metes and Bounds

The third type of preemption, known as the " metes and bounds"[113] of the invention is the essence of the patent system "to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries."[114] A monopoly is, therefore, granted to an individual or a corporation to exclusively practice the art while simultaneously depriving (or preempting) the public at large from this privilege. The economic justification for grants of intellectual property is to ensure that sufficient incentive exists for authors and inventors to invest time and resources in pursuit of innovations.[115]

1.5.3 Something more than routine building blocks but nonetheless trivial

Assume that, in our previous example, a patentee invents the database, which has something more than routine building blocks, but this something more is too trivial and small. For example, the inventor adds to his database one additional step, which allows the user to choose the color of the display of the database. It is not an essential component for databases. Therefore, in my proposal I name it as "anything more" than a routine building block. The color selection feature is in a prior art, and is conventional, and although this "anything more" passes the § 101 threshold, it will be invalidated under § 102 and § 103 because it is not novel and too obvious.

[112] *Alice*, 134 S. Ct. at 2354.
[113] Saches, *supra* note 110.
[114] U.S. Const. Art. 1. § 8 cl.8.
[115] Merges, Menell, Lemley , Intellectual Property in the New Technological Age, 5nd ed, 2010),Wolters Kluwer. [3] Federico, Pasquale J., *Origin and early history of patents*., J. Pat. Off. Soc'y11 (1929): 292.

CHAPTER TWO: HYPOTETICAL DISCLOSURE

2.1 The relationship between § § 101 and 112 of U.S. Patent Act in this Study

The sufficiency of disclosure has a substantial role in the validity of a software patent. Sufficient disclosure is the "heart of Patent Act § 112".[116] "The standard of adequate disclosure is to allow a notional worker in the field, one with an expected level of skills, to implement the method as disclosed [enablement]."[117] "It also requires that the inventor must describe what she claims and must claim what she represents."[118] Furthermore, "it requires that others could determine the boundaries of the claimed invention [scope of the claim]."[119]

The key connection between the first part of my proposal on §101 and the second part of my proposal on disclosure §112 is the patent claim(s). Claims can be invalidated under §101 if the claims are not supported and justified in more detailed disclosure under §112. Detailed disclosures are needed to justify detailed claims, and if a claim is not supported by sufficient disclosure, it will be invalidated under both §101 and §112.

Here, the consequence of insufficiency under §112 on rejection under §101 will be examined. To explain this matter, assume that the patentee invents a data asset management system with five claims in total. The first four claims of the invention are routine building blocks. In the fifth claim, the patentee claims, "I configure this data asset management system in a way it works in retail stores only." He claims that his invention has something more than routine building blocks, but does not disclose any corresponding structure of it anywhere in the application. The question here is whether his application will be rejected under §101, §112, both §101 and §112, or none of them?

Compare this example with our previous "online shopping" illustration, where we assume that the steps of searching, choosing, paying, and receiving the product are building blocks of the online shopping process. Here patentee claims, " I add the step of "Undo your purchase" into the routine building blocks of online shopping." He claims that his invention has something more than routine building blocks but does not disclose any corresponding structure of it anywhere in the application. Again, the question here is whether his claim should be rejected under §101, §112, both §101 and §112, or none of them?

In response to this question, I divide the software patent cases into two categories: ***"easy cases"*** *and* ***"hard cases."*** As a result, under the proposed eligibility test, their treatments are different. We have ***"easy cases"*** in determining "anything more" than routine building blocks such as our "Undo your purchase" example, and we have ***"hard cases"*** like our "retail store" example. When there is no prior art associated with the new feature in the invention, I consider it an "easy case". In other words, when the invention has feature(s), component(s), or step(s) that did not exist before, something more than routing building blocks is easy to recognize. Therefore, even if the patentee does not disclose any correspondence structure of that new feature, here "undo your purchase," the case passes the §101 threshold. Therefore, the insufficiency of the disclosure will not affect the validity under §101.

In contrast, I consider other cases as "hard cases" if, based on the inventor's claims, we realize that adding something more than routine building blocks occurred in existing functionality or subsystems. It is a "hard case" if determining "anything more" than routine building blocks

[116] Robert Merges. P. & John Duffy Fitzgerald, *Patent Law and Policy: Cases and Materials*, (7th ed. 2017)
[117] Maram Suresh Gupta, *Sufficiency of Disclosure in Patent Specification*, 14 Journal of Intellectual Property Rights. 307,316 (2009).
[118] Merges & Duffy, *supra* note 116.
[119] *Ibid.*

requires further analysis of the case, and the functionality of the claimed features, components, or steps are not apparent.

In our retail store example (our hard case scenario), it seems that the patentee only narrows the scope of the invention by saying that "I configure this data asset management system in a way it works in retail stores only." In this situation, further examination is required to determine whether he actually adds anything more than routine building blocks. Here, we should carefully examine the specifications to see what the corresponding structure to that feature or improvement is. Here is where §112 comes into play. If the patentee only claims that he designs a specific data asset management system for retail stores without any support in the specification, the patent will be rejected under both §101 and §112. It is rejected under §112 since the inventor does not sufficiently describe what he claims. Therefore, due to the lack of proper disclosure, our attempt to determine something more than routine building blocks fails. In other words, if the patentee does not disclose the corresponding structure of that specific feature, we do not know whether it contains anything more than a routine building block or not. It seems that it is just a trick to getting around building blocks, which, based on my proposal, is not patent-eligible.

Note that in our hard case scenario, the proposed eligibility test should be applied on that specific software type as a whole (here, data asset management system for retail stores). The inventor might design a particular customized data asset management system for retail stores. He might add some step(s) or feature(s) to his routine building block of the asset management system to optimize the system for retail stores. If this is the case, in determining the routine building blocks of the invention, which is our key focus, we should see the whole invention "asset management system for retail stores" as one software type. We should determine whether in designing the customized data asset management system for retail stores, the inventor adds "anything more" than routine building blocks or not. If so, it passes the §101. If not, it is rejected under §101.

Now with a more consistent and efficient test for patent eligibility, the patentee can choose to include some or all of the following factors (components), depending on the nature of the underlying claim in the patent application to fulfill the sufficiency of the disclosure requirement under both §101 and §112. In the following, I propose a hypothetical disclosure as an example to improve the quality of patent applications.

2.2 Hypothetical Disclosure Factors

Some factors, as I have discussed below, might affect the validity of a software patent, which I will examine in my proposal. Inadequacy or lack of these factors qualitatively affects the validity of software patent cases and would fail the §101 test if the disclosure is not detailed enough to narrow the claim. I will give robust indicators to the patent examiners, jurists, and practitioners to increase the certainty of predicting the outcome of the case based on the content quality of the underlying claims.

Factors:

2.2.1 Pseudocode Algorithm

2.2.2 Test case/ Best mode

2.2.3 UML diagrams[120]

- **a. Structural Diagrams**
  - i. Class diagram
  - ii. Object diagram
  - iii. Component diagram
  - iv. Deployment diagram
- **b. Behavioral Diagrams**
  - i. Use case diagram
  - ii. Sequence diagram
  - iii. Collaboration diagram
  - iv. State chart diagram
  - v. Activity diagram

2.2.4 Method used by the invention

Explaining the Factors:

In the following, I will explain each of these factors in detail and apply the model to some existing software patent cases (e.g., *Berkheimer* case[121]). This hypothetical disclosure model can be used by patentees to test the patentability of their product and establish the boundaries of sufficient disclosure before applying for the patent. It can be used by courts to evaluate the validity of patent claims, and it may eventually become the basis for national and international standards in software patenting.

1- Pseudocode

Any implementation of the software algorithm could be explained by pseudo code.

[120] "UML has provided features to capture the dynamics of a system from different angles. Sequence diagrams and collaboration diagrams are isomorphic; hence they can be converted from one another without losing any information. This is also true for Statechart and activity diagram." https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm.(last visited Mar. 27,2020).
[121] *Berkheimer v. HP Inc*., 881F.3d1360 (Fed. Cir. 2018).

a. What is Pseudocode?

"An outline of a program, written in a form that can easily be converted into real programming statements."[122] Pseudocode is not a complete coding of the system, or it is not a real code, "…but it looks like code. It helps people to understand a problem domain or solution better without having to add all the baggage necessary when using a real language."[123]

b. Advantages of the Pseudocode

Pseudocode shows the functionality of each function or subsystem of the software in a way that is more understandable and readable for humans rather than reading the actual code. Therefore, nothing will be left undefined or vague since the Pseudocode gives us a complete and detailed view of the functionality of the whole system.

For a software patent, the patentee should show the "inventive step" of his program and other parts of his invention. By generating the Pseudocode in software patent application, the "inventive step" of the program will stand out, and its position with other parts of the software will emerge.

2- Test Case/ Best mode

"Test case" has a specific meaning in software engineering: "[It] is a set of conditions or variables under which a tester will determine whether an application or software system is working correctly or not."[124]

The best mode, in the patent system, is considered "the most advantageous or preferable mode"[125] compared to other disclosures in the patent application. "The best mode disclosure requirement helps to improve the quality of the patent grant and helps to ensure that the public receives a full and honest disclosure in return for the grant of exclusivity."[126]

In my proposal, the " best mode" disclosure is different and distinct from the "test case" disclosure, and both are required to have sufficient disclosure. I define the best mode as the best and comprehensive example that shows how different parts of the software work together. It is like an integration test applied to the whole system to show the system's behavior from the beginning to the end. A test case, however, is an example that can be used for clarifying and explaining how each part of the system is working individually. In other words, the test case is for unit testing that demonstrates the functionality of any individual unit of the system.

In my opinion, the functionality of the system can be properly described using the combination of the test case and the best mode in the patent applications.

---

[122] *Definition of 'Pseudocode'*. The Economics Time, https://economictimes.indiatimes.com/definition/pseudocode (last visited Mar.30, 2020).
[123] *What is Pseudocode? Software Engineering*, 2/22/12, https://softwareengineering.stackexchange.com/questions/136292/what-is-pseudocode/136354 (last visited Mar.30, 2020).
[124] *How to write a good test case*, Apache Open Office, 7/27/2012, https://wiki.openoffice.org/wiki/QA/Testcase/How_to_write_test_case (last visited Mar.30, 2020).
[125] Bingbin Lu, *Best Mode Disclosure for Patent Applications: An International and Comparative Perspective*, Journal of Intellectual Property Rights Vol 16, September 2011, pp 409-417. Available at SSRN: https://ssrn.com/abstract=1938859
[126] *Ibid*.

3- UML diagram

By UML (Unified Modeling Language] diagrams[127], we can determine the different parts of the system and their relations with each other and with the whole software. The inventive step of the program will appear, and as a result, the examiner or the judges can easily decide if there is an inventive step that resides in the invention in comparison with the prior art. In fact, the most common tools between the designer and the programmer to communicate and understand each other regarding software production are UML diagrams. Depending on the nature of the invention, patentee should disclose different diagrams as much as the functionality of the system fully described.

UML diagrams are useful for decreasing the invalidity of software applications because: "UML diagrams describe the boundary, structure, and the behavior of the system and the objects within it."[128] "The Unified Modeling Language (UML) was created to forge a common, semantically and syntactically rich visual modeling language for the architecture, design, and implementation of complex software systems both structurally and behaviorally."[129] "These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process."[130](See Appendix A for different types of UML diagrams)

4- The method used by the invention

Software patent cases are mostly invalidated under §112 (lack of sufficient disclosure) since the patentee does not usually explain any corresponding structure for achieving his claimed invention. To prevent invalidity in this sense, mentioning the specific way for implementing the software in the application in one sentence or two will be helpful and essential. I named this specific way, the method used by the invention.

2.3 Applying Hypothetical Disclosure Factors on Some Cases

2.3.1 Hypothetical disclosure: *Berkheimer* case

2.3.1.1 Hypothetical Pseudocode: *Berkheimer* case

In the following, a hypothetical Pseudocode of the *Berkheimer* case is illustrated. To explain Berkheimer's Object-Oriented data model, I propose an example of an Object-Oriented design of his system to explain how this type of disclosure (Pseudocode) could be helpful to understand the system better. For implementing the Berkheimer system, I define three classes, one base class, and two other classes that are derived or inherited from the base class. Each inherited class possessed all the attributes and methods of the base class, and it may or may not have additional attributes or methods to the base class. Here, in this design sample, both of our inherited

[127] Although UML has a direct relation to object-oriented analysis and design, " it can be well applicable for non-object-oriented software architecture such as procedural paradigm." *See* https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language*; also see*, https://www.sparxsystems.eu/resources/project-development-with-uml-and-ea/introduction-to-uml/ https://stackoverflow.com/questions/29145545/is-uml-for-object-oriented-only https://stackoverflow.com/questions/26212374/are-uml-diagrams-only-for-object-oriented-approach?noredirect=1&lq=1 https://softwareengineering.stackexchange.com/questions/78407/software-design-for-procedural-programming

[128] *What is Unified Modeling Language?* Lucidchart, https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language. (last visited Mar 30, 2020).

[129] *Ibid.*

[130] *Ibid*.

classes, Document and CommonElements do not have any additional attributes or methods. For making the Pseudocode more understandable, I simplify the Object-Oriented notation. For example, here, I ignore the data type of each attribute or return value data type of the method.

Notational Conventions:

The following notational conventions are used throughout this pseudocode:

- Comments or explanation of the pseudocode in green is enclosed by /* */
- The "reserved" or "keywords" used in programming languages are in blue.
- "D" or "doc" an abbreviation of "document," "prop," an abbreviation of "property," and "elem" an acronym of "element."

1- Create the Base class

```
Class Base
{
Attributes:     /* "Attributes are data stored inside a class or instance and represent the state or quality of the class or instance." */
        header: "",
        footer: "",
        body: "",
        imageUrl: https://Pics.com/img1.jpeg,
Methods:    /* class functions also known as methods*/
        Edit(object, element, value );    /* Object, element, data are the parameters of these functions. When you call these functions, you should pass a value to these parameters. The object represents the thing that you want to manipulate. The element represents the specific element in that object, such as header, footer, or body that you want to make a change in it. Value is the new value that you want to give to that element. */
        Delete(object, element );
        Add(object, element, data );
}
```

2- Create inherited classes

```
Class Document: Inherit Base
{
}
Class CommonElements: Inherit Base
{
}
```

3- Create objects (instances) from the class "Document" and class "CommonElements."

By creating an object, we create an instance of the class, and through this object, we can have access to the attributes and methods of that class. Therefore, we can store each doc data into an object. Therefore, we need a data structure for storing a bunch of objects. In doing so, we create an array of doc classes.

```
/* In order to be able to use the attributes and methods of a class, first we need to create an
object from that class. We create an object by using "new" as a keyword and assign it to an
object with a custom name, such as "dObject" */
Document dObject = new Document();
CommonElements cObject = new CommonElements();
Array documentObjArray = new Array (Document)          /* create an array of objects*/
documentObjArray.add(dObject);                         /* Add each object into an array*/
```

4- Converting the documents into a standard input format that is understandable for the computer (e.g., pdf, gif, jpeg)

```
/* start the process of getting and organizing data into Object-Oriented Data model which is
created above. */
Input(file);
htmlFormatFile = convertToHtml(file);  /*Converting to Html file and assign it to a
"htmlFormatFile" variable*/
```

5- Parsing and tagging (extracting data from the standard format input document).

For manipulating the data in a document (editing, removing, adding, and deleting), it is essential to parse and read (extract) the document and element properties and values by pre-defined tools and libraries in conventional programming languages.

```
docProps = getDocProps(htmlFormatFile);  /* extract doc properties and values from the Html
format file*/
docElementsProps = getElementProps(htmlFormatFile);  /* extract doc elements properties
and values from Html format file */
addMetaData(docProps, docElementsProps)    /* Add Meta data */
```

Output demonstration

```
/* docElementsProps structure */
Print(docElementsProps);
Output: docElementsProps = {Prop1 = Value1, Prop2 = Value2, …, Prop(n) = Value(n)}
Prop1 == header, Prop2 == footer, Prop3 == body …
```

6- Defining rules for data storing and manipulation

Here, inventor designs and implements a User Interface (UI), which allows the user to define the rules by himself to manipulate (e.g., adding, reconciling, deleting) the data automatically or manually.

```
/*Getting rules from the user into RuleSet array*/
 RuleSet[] = getUserRules();  /* Array of RuleSets*/
```

Ruleset [1]: for organizing and storing data

```
/* Initialize common object(s)*/
If (htmlFileFormat is a common document)  //e.g. Ruleset [1].rule1 is "htmlFormatFile is a common document"
{
cObject.header = docElementsProps.Prop1;  /*cObject= Common object*/
cObject.footer = docElement.Prop2;
cObject.body = docElement.Prop3;
}
```

```
/*Initialize other document objects*/
Else if (htmlFileFormat is not a common document)
/* one-to-many:  As you can see, here, by creating an Object-Oriented class
(one) we can organize many documents data (many) into its instances ("dObject" s) */
          If (docElementsProps .header != cObject.header)   /*!= means is not equal*/
             dObject.header = docElementsProps.header;
          Else
            dObject.header = cObject.header;
```

Ruleset [2]: for manipulating data (Edit, delete, add, sort)

```
If (RuleSet [2].rule1 == true)
          Edit (dObject, header, value1)
If (RuleSet [2]. rule2 == true)
          Delete(dObject, elem)
If (RuleSet [2].rule3 == true)
          Add (dObject, footer, value2)
```

```
/*Example of other functionalities in the system, the result set should be an array as well because the result of these functions, search and sort, could be more than one record, so it should be kept in an array.*/
ResultSet[]  = Search(documentObjArray, value)  // Search the value in all objects
ResultSet[]  = Sort(documentObjArray, elem, Acending) // Sort all objects based on their specific element or property in acending order
```

2.3.1.1 UML Diagram v. Flowchart: *Berkheimer* case

Diagram Criticism:

In the following, I will mention some of the deficiencies in Berkheimer's drawings that he uses for explaining his invention:

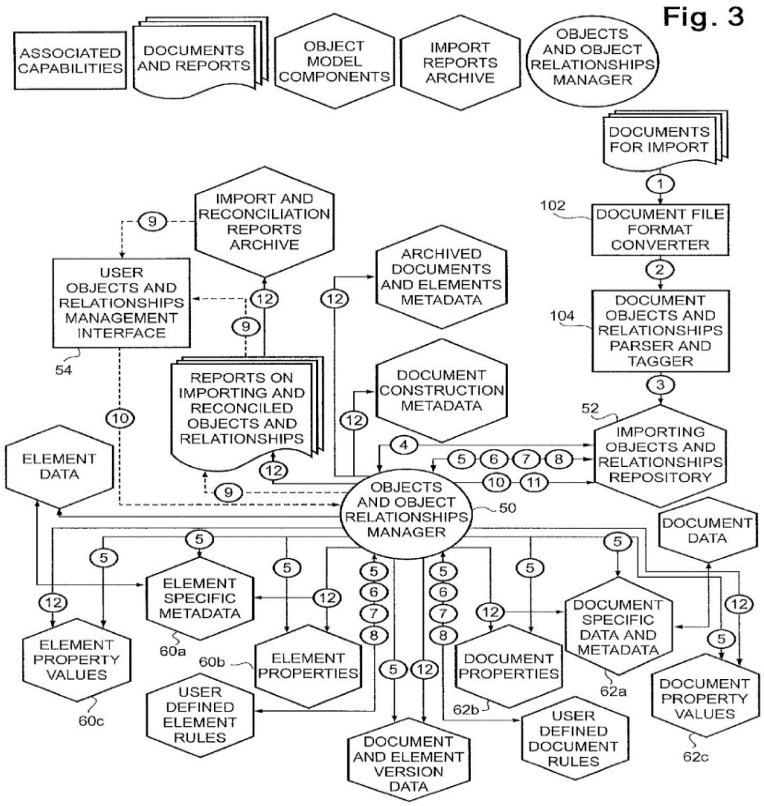
1- The number of diagrams is too many (9 diagrams). The diagrams are almost the same; for example, diagram 2.B is identical to diagram 4.

2- Diagrams are meant to be used for better clarification and explanation of a system, but here Berkheimer's use of flowcharts confuses the reader. I propose that instead of using flowcharts for these kinds of complicated systems, we should instead use UML diagrams that are generally clearer and more understandable.

3- The patent application diagrams should focus mainly on explaining the important and inventive parts of the system. Few drawings would be enough to show the relation or interaction of the inventive parts with each other and the whole system. But here, lots of unnecessary details are displayed in the diagrams, which make it more ambiguous and complex.

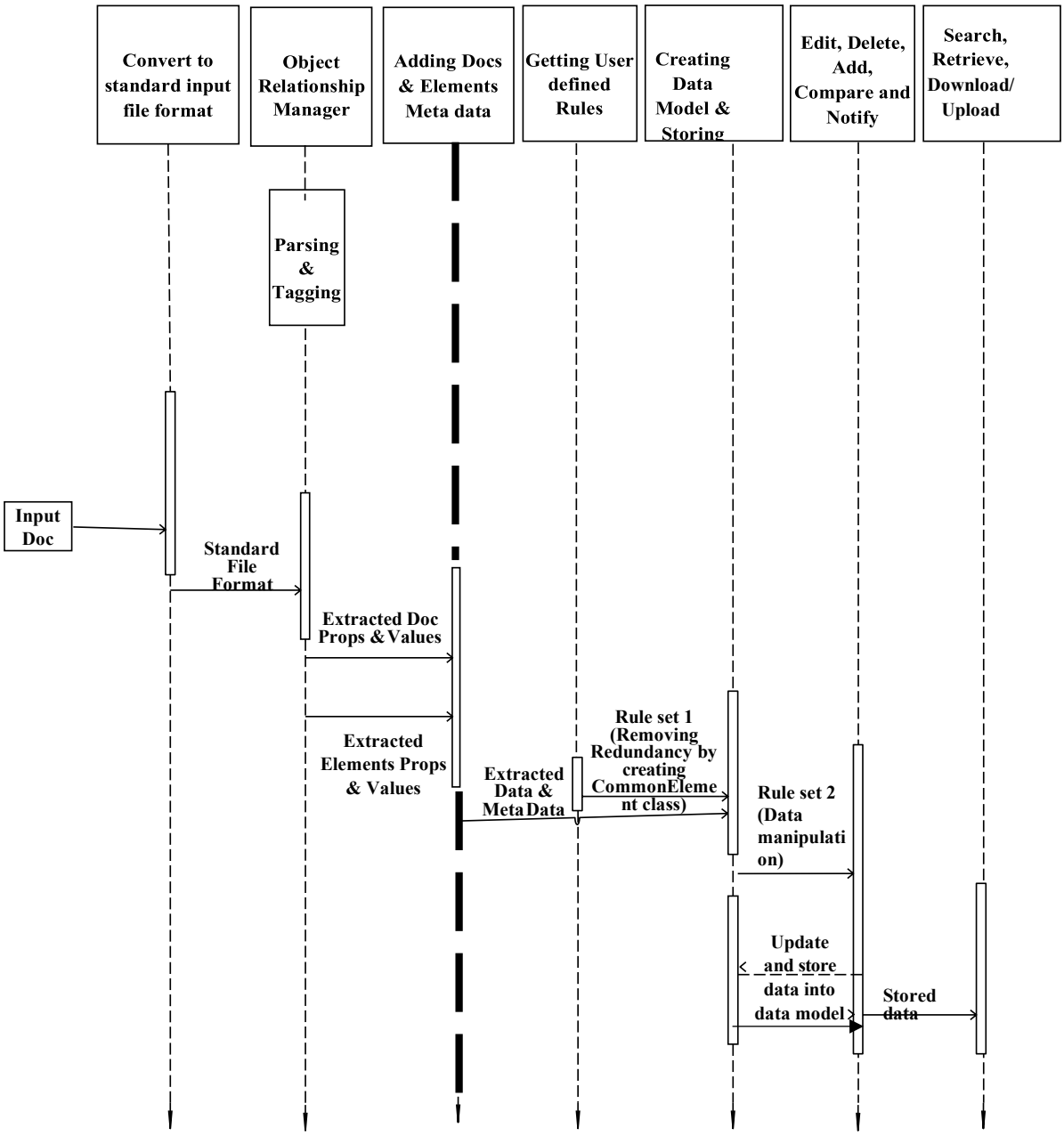
In the following, I will rewrite one of Berkheimer's drawings (Figure 19 of this paper and Figure 3 of his original patent application) to compare the clarity of the flowcharts that patentee discloses to his application with two suggested UML diagrams (Sequence and Activity diagrams) as an example to show how these diagrams operate and improve the quality of the patent application.(FIG.20&21)

Figure 19. Original flowchart from *Berkheimer* patent application



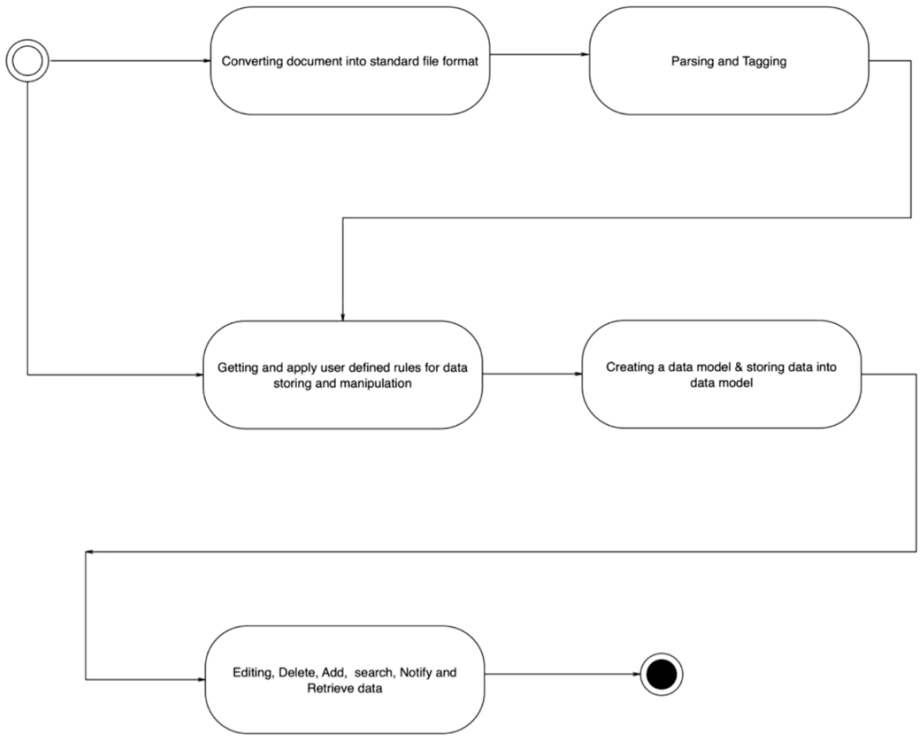
UML diagrams are supposed to demonstrate "how objects interact with each other and the order those interactions occur."[131] "The processes are represented vertically, and interactions are shown as arrows."[132]As you can see from the above flowchart of Berkheimer's application, the logical relation between components is unclear. As a result, understanding the sequence of creating the objects and performing functions is too difficult. By applying one of the UML diagrams, sequence diagram (Figure 20), the system and the sequence of events are more perceivable.

Figure 20. Suggested Sequence diagram for *Berkheimer* case



[131] Nishadha, *UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples*, Creately blog, https://creately.com/blog/diagrams/uml-diagram-types-examples/#SequenceDiagram. (last visited Mar. 27,2020).
[132] *Ibid* at https://creately.com/blog/diagrams/the-basics-the-purpose-of-sequence-diagrams-part-1/

Figure 21. Suggested Activity diagram for *Berkheimer*



The method used by the invention: Berkheimer's specific way for implementing his invention can be summarized as editing the elements once in an archive, and it affects multiple documents where the element appears by using an Object-Oriented data model. This helps the examiner understand the concept of Berkheimer's invention at a quick glance.

2.3.2 Hypothetical Disclosure: *McRo Case*

2.3.2.1 Hypothetical Pseudocode: *McRo Case*

In the following, a hypothetical Pseudocode (drafted by the author) of the McRo[133] case is illustrated.

Input: Time Aligned Phonetic Transcription: TAPT and Time Aligned Emotional Transcription: TAET (optional), and Time Aligned Data File (optional) (The system takes these files as inputs)

1- Determining Global variables

```
string   Time[];
string   Phoneme[];
string   MorphWeight[];
string   Prior-Interpolation-MorphWeight[];
string   Interpolation-MorphWeight[];
string   Morph-Weight-Set-Stream ;
```

2. Read TAPT's times and phonemes in two arrays: Time[] and Phoneme[]

```
for each tapt-element in TAPT      /*TAPT file can be a word or
                                     Excel or any other file*/
  Time[i] = tapt-element.time;
Phoneme[i] = tapt-element.phoneme;
```

Example:

Time[0] = 0.08
Time[1] = 0.1
Time[2] =0.16
….
Phoneme[0]= "h"
Phoneme[1]= "e"
Phoneme[2]= "l"

3. Generating Binary morph weights based on Secondary rules and default rules (If… then…. else) and save them into MorphWeight[] array. First, the secondary rules will be checked if they were not applicable for a specific phoneme, then in ELSE part the default rules will be applied:

[133] McRo, Inc. v. Bandai Namco Games America, case number 15-1080, from Appellate - Federal Circuit Court.

```
for each p[i] in Phoneme

    If p[i] == “oh” &&  Time[i]-Time[i-1] > 1.2 sec then
          MorphWeight[i] = 000010;

    If p[i] == “eh” && p[i+1] == “oh” && p[i-1] == “h” then
          MorphWeight[i] = 000001;
    ….
Else                          /*Default Rules*/

    If p[i] == “h”    then
          MorphWeight[i] = 100000;
    If  p[i] == “eh”   then
          MorphWeight[i] = 010000;
   …
```

4- Post process before interpolation by generating and adding random noise

```
for each mw[i]  in MorphWeight
        Prior-Interpolation-MorphWeight[i] = RandomNoise(mw[i]);   /*generating random
                                                                    noise is in prior art*/
```

5- Interpolation

```
for each pi[i] in Prior-Interpolation-MorphWeight
        Interpolation-MorphWeight[i] = Interpolation(pi[i]);   /*Interpolation
                                                                function exists in prior art*/
```

6- post-processing after interpolation: We should add an arbitrary morph weight from auxiliary (secondary) morph target to each keyframe.

7- Generate morph weight set stream:

```
for each t[i] in Time
            Time[i+2] = Time[i+2] + 0.01 sec
        /*By adding the arbitrary morph weight to each key frame , we  need to add the
        constant value to time except for the first and second time in the array*/
        for each imw[i] in Interpolation-MorphWeight
  Morph-Weight-Set-Stream = Time[i] + imw[i] + Time[i+1] + Arbitrary-Morph-Weight
```

8- Graphic part of the program will get the Morph-Weight-Set-Stream as an input to apply on animation characters.

2-Test case/ Best mode

In the McRo case, both test case and best mode disclosure are available. In his application, the patentee shows us how each one of these rule categories (default, secondary and post-process rules) works properly in a system through examples (test cases.) Simultaneously, the system behavior as a whole is demonstrated through an example (best mode.) This best mode shows how these rules (different parts of the system) are working together to emerge the final desired result of the system, which is generating the "morph weight set stream." In the patent application, the following test case for each of the rules (default, secondary and post-process rules) is demonstrated.

Test case

"Default Rules:
Criteria: Encounter a "h" as in "house"
Function: Use morph weight set (1,0,0,0,0,0) as transition target.

Criteria: Encounter an "eh" as in "bet"
Function: Use morph weight set (0,1,0,0,0,0) as transition target.

Criteria: Encounter a "1" as in "old"
Function: Use morph weight set (0,0,1,0,0,0) as transition target.

Criteria: Encounter an "oh" as in "old"
Function: Use morph weight set (0,0,0,1,0,0) as transition target.

Criteria: encounter a "silence"
Function: use morph weight set (0,0,0,0,0,0) as transition target.

Secondary Rules

Criteria: Encounter an "oh" with a duration greater than 1.2 seconds.
Function: Use morph weigh set (0,0,0,0,1,0)

Criteria: Encounter an "eh" followed by an "oh" and preceded by an "h".
Function: Use morph weigh set (0,0,0,0,0,1) as transition target.

Criteria: Encounter any phoneme preceded by silence
Function: Transition start time=(the silence's end time)-0.1*(the incoming phoneme's duration):Transition end time=the incoming phoneme's start time

Criteria: Encounter silence preceded by any phoneme.
Function: Transition start time=the silence's start time +0.1*(the outgoing phoneme's duration)

Post Processing Rules

Criteria: Encounter a phoneme duration under 0.22 seconds.
Function: Scale the transition target determined by the default and secondary rules by 0.8 before interpolation."[134]

Best mode of the invention:

Similarly, in the patent application for the present invention, the best mode is adequately explained. Note that the following hypothetical pseudocode (drafted by the author) extracted from a detailed explanation of the best mode resides in the patent application.

Default rule: **If (Criteria) _1:** Encounter an "oh" as in "old"
Secondary rule: **If (Criteria) _2**: Encounter an "oh" longer than 1 second long.
Then (Function) _2: Insert transitions between a defined group of morph weight sets at 0.5 second intervals, with transition duration's of 0.2 seconds until the next "normal" transition start time is encountered.(0100000)
Else (Otherwise) _2
Then (Function) _1: Use morph weight set (0,0,0,1,0,0) as transition target.
Post processing rule: **If (Criteria) _3:** Auxiliary morph target had the character's mouth moved to the left cyclically
Then (Function) _3: output animation would have the character's mouth cycling between centers to left as he spoke.

2.3.2.2 UML Diagram v. Flowchart: *McRo case*

In the following, I will compare the clarity of the flowcharts that patentee in McRo case discloses to his application (Figure.22) with two suggested UML diagrams (Sequence and Activity diagrams) (FIG.23 &24) as an example to show how these diagrams operate and improve the quality of the patent application.

[134] McRo, Inc. v. Bandai Namco Games America, *Ibid.*

Figure 22. Original Application Flowchart from McRo Case

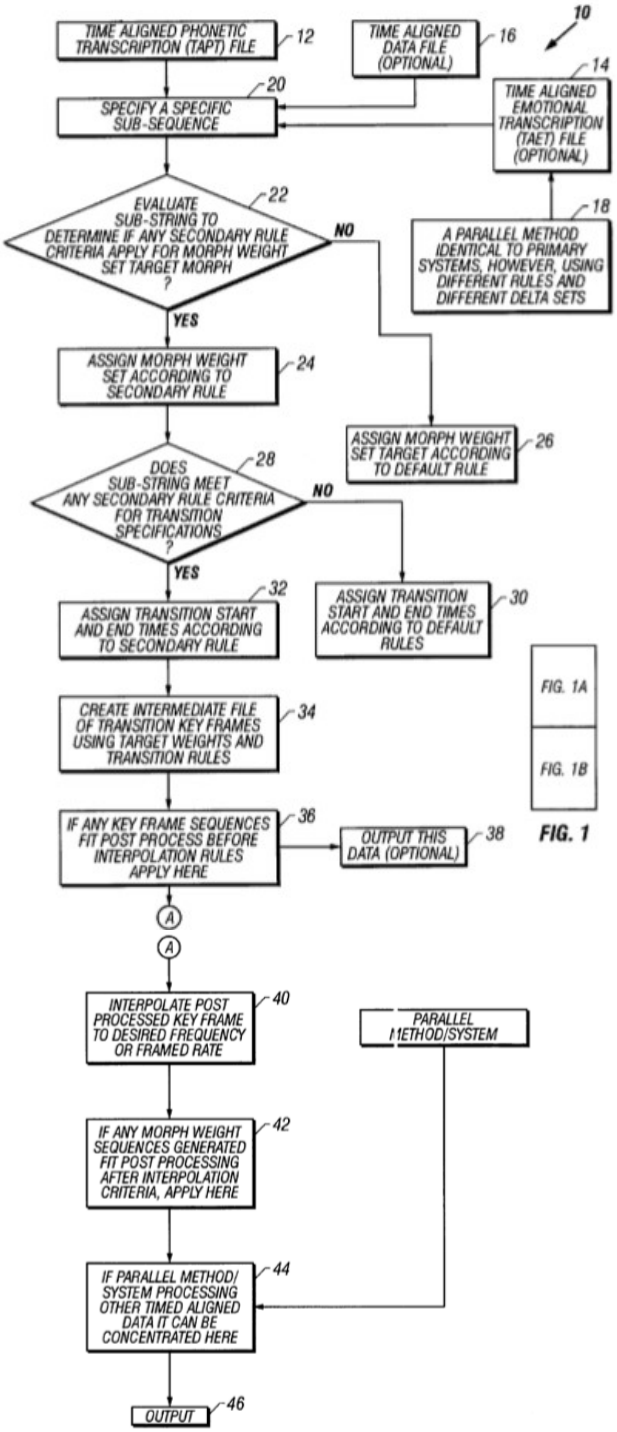


FIG. 1

Figure 23. Suggested Sequence Diagram for McRo Case

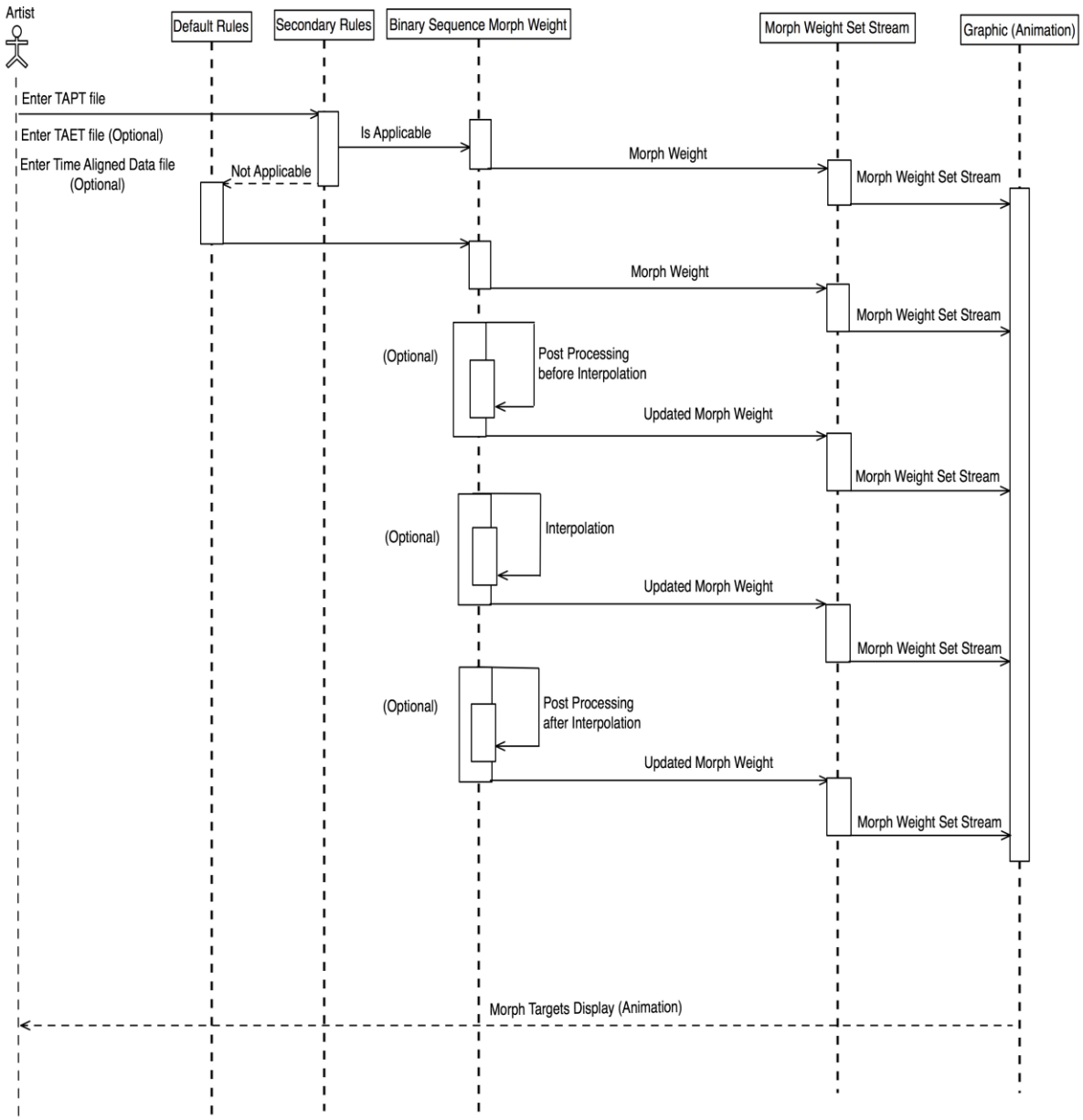
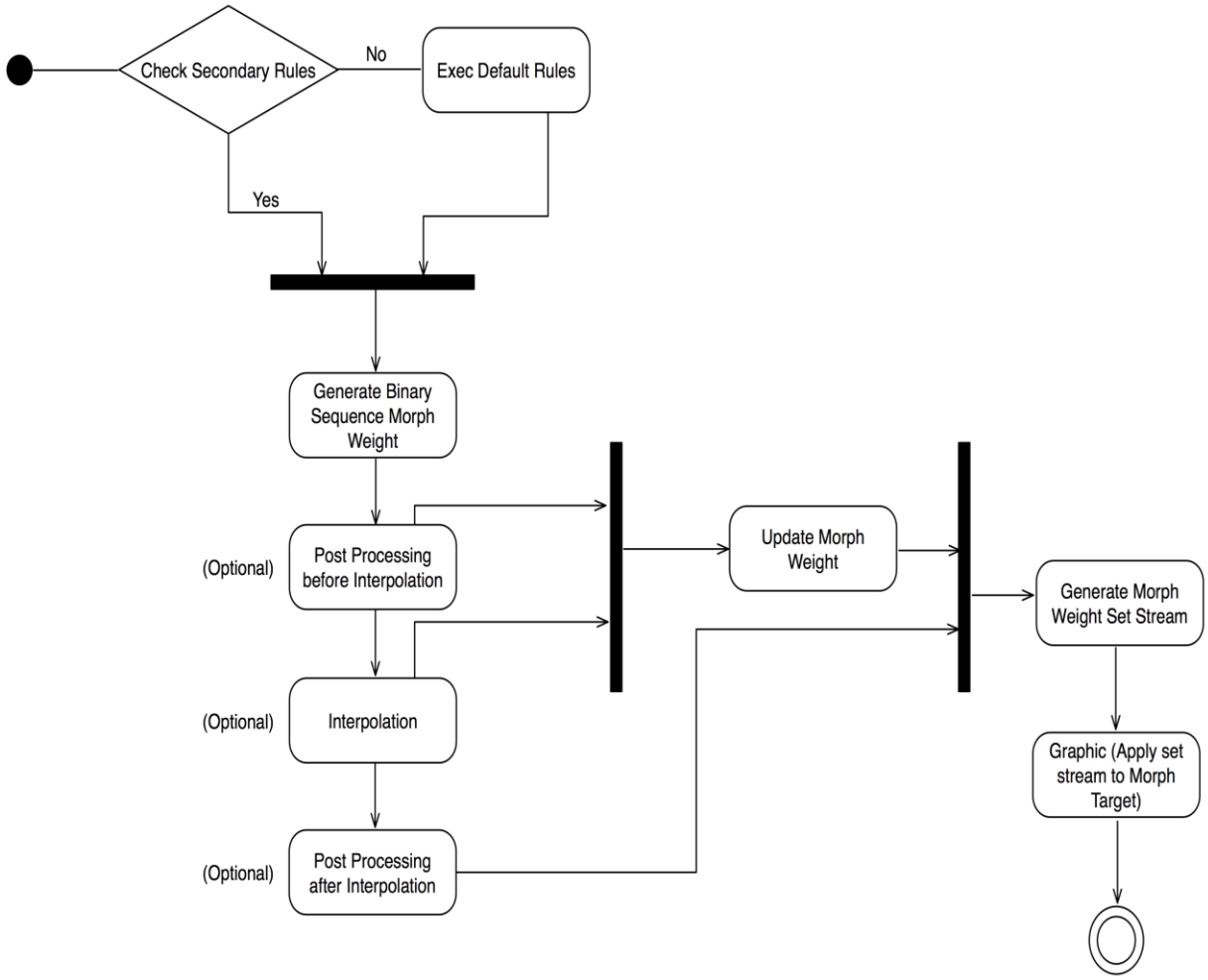


Figure 24. Suggested Activity Diagram for McRo Case



2.3.2.2 The method used by the invention: *McRO* case

In the McRO case, the "**binary sub-sequence rule-based**" method, is a specific way for automatically animating the lip synchronization and facial expressions of 3D characters and has an "inventive concept" sufficient to transform the abstract idea into a patent-eligible application.

CHAPTER THREE: AI RELATED INVENTIONS

3.1 AI invention: Deep Embedding Forest Model

The title of an AI patent publication (Publication #20180060728[135]) is "Deep Embedding Forest: Forest-based Serving with Deep Embedding Features," [136]and its abstract is as follows:
"A deep embedding forest-based (DEF) model for improving online serving time for classification learning methods and other tasks such as predicting user search selection results provided in response to a query or for image, speech or text recognition. Initially, a deep neural network (DNN) model is trained to determine parameters of an embedding layer, a stacking layer, deep layers, and a scoring layer thereby reducing high dimensional features. After training the DNN model, the parameters of the deep layers and the scoring layer of the DNN model are discarded, and the parameters of the embedding layer and the stacking layer are extracted. The extracted parameters from the DNN model then initialize parameters of an embedding layer and a stacking layer of the DEF model such that only a forest layer of the DEF model is then required to be trained. The output from the DEF model is stored in computer memory."[137]

As it appears from the abstract, this new AI invention creates the DEF model by discarding two layers while extracting two of the four layers of the DNN model with the forest-based model. The advantage of the DNN model is the ability to "extract high- level embedding vectors from low-level features."[138] Having a low serving cost is the benefit of the forest-based model.[139]
This AI invention is used in the prediction models of sponsored search. Sponsored search is one kind of machine learning application. The goal of the invention is "to show the user the advertisement that best matches the user's intent."[140] (When the user searches in a search engine, based on his or her query, the related ad is displayed.) For doing so, this invention is predicting the probability of click on a specific ad by the user. This prediction happens by "given input strings from a query, a keyword, an ad title, and a vector of dense features with several hundred dimensions."[141]

The prior art of the DEF model is a Deep crossing model which has been invented by almost the same inventors. In the prior art, the Deep Crossing model, we had to train four layers of the DNN model (embedding layer, stacking layer, deep layer, and the scoring layer), which has high computation cost (computation of several huge matrixes in multiple layers.) In DEF model (the new invention), the model extracted the embedding layer and stacking layer of the DNN model and discarded both the deep layer and the scoring layer of the same model. Then, the extracted parameters from the DNN model will be fed as initial parameters of an embedding layer and stacking layer of the DEF model. Consequently, only the forest layer of the DEF model needs to be trained. By doing so, the accuracy of the result is optimized in faster learning time. This is the reason that DEF can reduce prediction time significantly.[142]

---

[135] "U.S. Patent Application No. 15/253,669, Publication No. 20180060728 (published Mar.1, 2009) (Microsoft Technology Licensing, LLC, applicant)."http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20180060728.PGNR.&OS=DN/20180060728&RS=DN/20180060728 (last visited Mar.10, 2019)
[136] *Ibid*.
[137] *Ibid.*
[138] J. Zhu et al., *Deep Embedding Forest: Forest-based Serving with Deep Embedding Features (*Mar.16, 2017), https://arxiv.org/pdf/1703.05291.pdf
[139] *Ibid.*
[140] *Ibid.*
[141] *Ibid.*
[142] *Ibid.*

As you can see in (FIG. 25)[143], the DEF model consists of feature inputs, the embedding layers, the stacking layer, the forest layer, and the objective function. In (FIG.26)[144], "the embedding layers and the stacking layer of the Deep Crossing model are the same as in the DEF model in Figure 25. The difference is that the forest layer is replaced by the layers inside the dotted rectangle."[145]

Figure 25: The Deep Embedding Forest Model

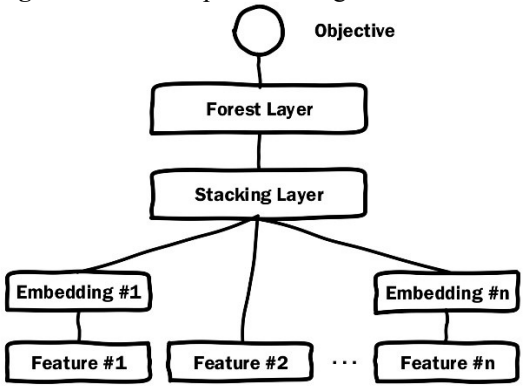
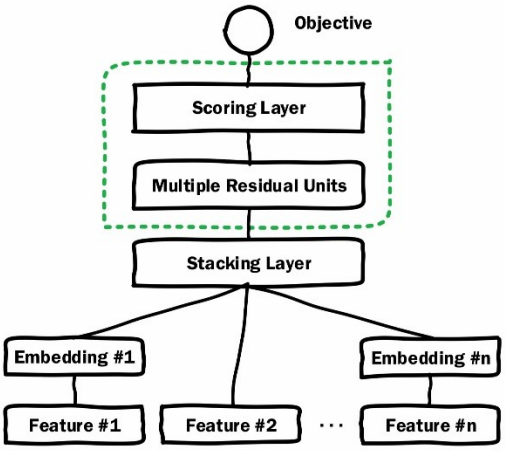


Figure 26: The Deep Crossing Model used to initialize the embedding layers of DEF



"Two-Step DEF V. Three-Step DEF

Training the DEF model involves three steps:

1- Initialize embedding layers with Deep Crossing
2- Initialize the forest layer with XGBoost and LightGBM
3- Joint optimization with partial fuzzification

[143] All figures in this section (AI related inventions) are extracted from the following article:
J. Zhu et al., *Deep Embedding Forest: Forest-based Serving with Deep Embedding Features (*Mar.16, 2017), https://arxiv.org/pdf/1703.05291.pdf
[144] *Ibid.*
[145] *Ibid.*

Two options are applying DEF. The first option, the Two-Step DEF, includes only step 1 and 2 of initialization the second option, the Three-Step DEF, comprises the full three-steps including partial fuzzification as mentioned above."[146]

Describing the functionality of each step is of less interest in this paper.

3.2 Applying Patent Eligibility Test: AI Invention

3.2.1 Routine Building Blocks determination: Deep Embedding Forest Model

As I mentioned before, the specific task in this AI invention is known as "Sponsored Search," that is, showing ads related to the intent of a given user. Thus, the routine building blocks are as follows:

1- Data Collection: Collecting the logged clicks (both absence and presence) given the user and the query and some other details such as the landing page, ad title, ad description, etc.

2- Training: Training an *accurate* Machine Learning model; predicting the click using the collected data.

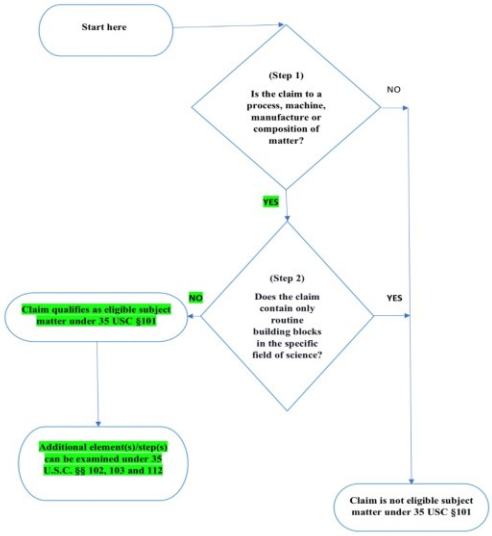
3- Inference: Using the trained model, predict the click for an online stream of data.

3.2.2 Determining "Anything more than Routine Building Block": AI Invention

The following statement is the specific improvement of the system: Improving online serving time and the accuracy of the result of predicting user search selection by discarding the deep layer and the scoring layer of the DNN model and replacing it with the forest layer with deep embedding features (DEF model.) In other words, the main contribution is to maintain the accuracy in step **"Training"** while improving the speed in step **"Inference."**

In general, in AI inventions, the more layers we have, the longer the serving time, but few layers hurt the accuracy. This invention reduces the serving time while keeping the accuracy high. In other words, the main contribution is in lowering the serving time, while not hurting the accuracy. The following flowchart (FIG.27) is a visual demonstration of my patent eligibility test as it applies to the AI invention, A deep embedding forest-based (DEF) model:

Figure 27: Application of Proposed Patent Eligibility Test: AI Invention



[146] Zhu et al. *supra* note 138.

3.3 HYPOTETICAL DISCLOSURE: AI RELATED INVENTIONS

In AI (Artificial Intelligence) inventions as other software patent inventions, the sufficiency of disclosure has a substantial role in validity/invalidity of the patent. The factors that we have discussed in the previous chapter could apply to another vast majority of software category, AI inventions as well. These factors are as follows:

1- Pseudocode Algorithm
2- Testing practice (Analytical report)
3- UML diagrams[147]
4- Method used by the invention

In the following, the factors above are applied to an AI patent publication (Publication #20180060728[148]) to examine the sufficiency of its disclosure. This hypothetical disclosure model can be used by patentees to test the patentability of their product and establish the boundaries of sufficient disclosure before applying for the patent. It can be used by courts to evaluate the validity of patent claims, and it may eventually become the basis for national and international standards in AI patenting.

3.3.1 Pseudocode: AI Invention

Any technical explanation of the software algorithm could be express by pseudocode. This is perhaps the best way of expressing the mechanism of the algorithms in AI. In a printed publication[149] (and not in a patent application), the patentee attached the following pseudocode (FIG.28)[150], which helps to understand the main functionality of the AI invention.

[147] *Unified Modeling Language*, Tutorialspoint, https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm
"Dynamic nature of a system is very difficult to capture. UML has provided features to capture the dynamics of a system from different angles. Sequence diagrams and collaboration diagrams are isomorphic, hence they can be converted from one another without losing any information. This is also true for Statechart and activity diagram.",
[148] "U.S. Patent Application No. 15/253,669, Publication No. 20180060728 (published Mar.1, 2009) (Microsoft Technology Licensing, LLC, applicant)."http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20180060728.PGNR.&OS=DN/20180060728&RS=DN/20180060728 (last visited Mar.10, 2019)
[149] Zhu et al. *supra* note 138.
[150] *Ibid.*

Figure 28: Pseudocode: Three-step Deep Embedding Forest

Algorithm 2: Three-Step Deep Embedding Forest

Input: Dataset D

Train a Deep Crossing model to initialize embedding layers with parameters $\{\mathbf{W}_j\}$ and $\{\mathbf{b}_j\}$ (Sec. 5.1)
▷ Map D to a low-dimension feature space (noted as D')
for $d \in D$ **do**
  compute stacking vector $\tilde{\mathbf{y}}$ according to Equ. 3
  pair $\tilde{\mathbf{y}}$ with target t
end
Train a forest/tree-based model with D' to initialize the forest layer with parameters Ψ, Θ and Π (Sec. 5.2)
▷ Joint optimization with partial fuzzification (Sec. 5.3)
for $m \in M$ **do**
  ▷ M is a set of mini-batches of D
  ▷ Forward Propagation
  for $d \in m$ **do**
    compute $\tilde{\mathbf{y}}$ according to Equ. 3
    for $l \in \mathcal{L}$ **do**
      ▷ $\mathcal{L}$ is a set of all leaf nodes in the forest.
      compute score on l according to Equ. 8 (App. A)
    end
    compute prediction score $\bar{t}$ according to Equ. 9 (App. A)
  end
  ▷ Backward Propagation
  compute $\{\frac{\partial O}{\partial \pi_l}\}, \{\frac{\partial O}{\partial c_r}\}, \{\frac{\partial O}{\partial a_r}\}, \{\frac{\partial O}{\partial \mathbf{W}_j}, \{\frac{\partial O}{\partial \mathbf{b}_j}\}$
  update $\{\pi_l\}, \{c_r\}, \{a_r\}, \{\mathbf{W}_j\}, \{\mathbf{b}_j\}$
end

3.3.2 Testing practice (Best mode) :AI Invention

In software engineering, the "test case" is used by a tester to determine whether an application or software system is working correctly or not. However, in almost all AI algorithms, there are lots of randomness and stochasticity. Therefore, using a few data points as test cases to evaluate these AI algorithms does not work. Instead, performing experiments with thousands or sometimes millions of data points (data set) is the way to show the validity and performance of these algorithms. A test case is used for testing of granular data (data points) rather than big data (data set); this is the reason why I changed the "test case" to "testing practice"[151] in this chapter. Using big data makes us use other mathematical theories such as the probability theory to evaluate the

[151] Krunal Patel, *Step Ahead In Competition With AI Test Automation*, Clarion Technologies, https://www.clariontech.com/blog/step-ahead-in-competition-with-ai-test-automation (last visited May 26, 2020)

results. The probability theory has a significant role in evaluating the results of the AI applications output. It gives us the idea of how to improve our software. Therefore, it makes the AI algorithm robust and powerful. In the following, some examples of testing practices are illustrated from the paper published by the patentee.[152] Including the following diagram[153](FIG.29) and tables[154](FIG.30&32&32) in AI-related patent applications will help to increase the quality of patent applications and the sufficiency of their disclosure.

Figure 29: Experiment with 3 Million Samples: Log Loss Comparison & Experiment with 3 Million Samples: Prediction Time Comparison: The comparison of scalability of the prediction runtime per sample on CPU processors among Deep Crossing, Two-step DEF with XGBoost and Two-step DEF with LightGBM. Note that the axis of prediction time for DEF is on the left side, while the one for Deep Crossing is on the right side.

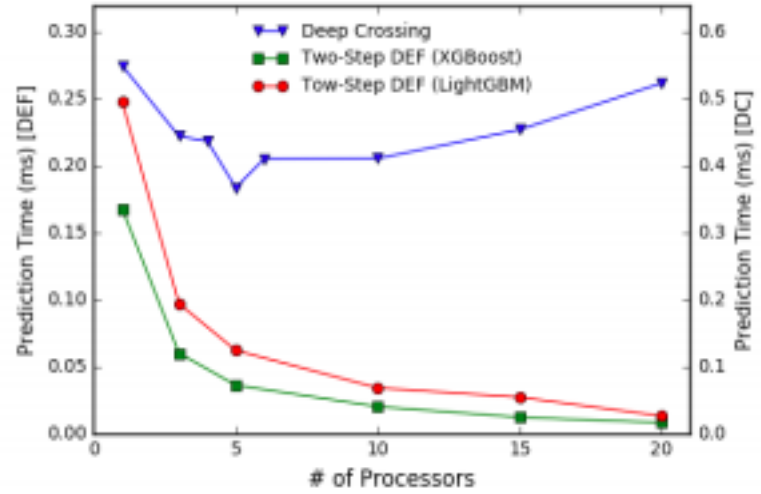


Figure 30: Comparison of performance and prediction time per sample between Deep Crossing and DEF. The models are trained with 3.59 M samples. Performance is measured by a relative log loss using Deep Crossing as the baseline. The prediction time per sample was measured using CPU processor.

	Deep Crossing	Two-Step DEF (XGBoost)	Two-Step DEF (LightGBM)	Three-Step DEF (LightGBM)
Relative log loss	100	99.96	99.94	99.81
Time (ms)	2.272	0.168	0.204	-

Figure 31: Comparison of performance and prediction time per sample between the Deep Crossing and Two-Step DEF using LightGBM that are trained with 60M samples. The predictions both were conducted on CPU processor. Performance is measured by relative log loss using Deep Crossing as the baseline.

	Deep Crossing	Two-Step DEF
Relative log loss	100	99.77
Time (ms)	2.272	0.162

[152] Zhu et al. *supra* note 138.
[153] *Ibid.*
[154] *Ibid.*

Figure 32: The same as Figure 21 but for the models trained by 1B samples

	Deep Crossing	Two-Step DEF
Relative log loss	100	100.55
Time (ms)	2.272	0.204

3.3.3 UML Diagram v. Flowchart: AI Invention

UML is a language that can be used to analyze, specify, construct and document software inventions. UML diagrams are the best tools in describing the "Analysis Pattern" [155]in AI inventions as well.

By UML diagrams, we can determine the different parts of the system and their relationships with each other and with the whole software. The most common tools between the designer and the programmer to communicate and to understand each other regarding software production are UML diagrams. Therefore, in software patent applications, AI inventions should use UML diagrams to help the patentee describe his invention properly and more understandably. For instance, UML diagrams in Neural Networks[156]are beneficial to depict the architecture of the networks of neurons in a diagram.

Depending on the nature of the invention, a patentee should disclose different diagrams as much as the functionality of the system fully described. For example, in Figure 33 although the patentee tried to explain his invention with diagrams which are similar to the UML diagrams (a combination of the Activity diagram and the Communication diagram), yet are far from the standard of the UML diagrams. The following suggested diagram will show the clarity (Figure 34) over the patentee's diagram (Figure 33). In this illustration as an example, by using the UML diagram (Sequence diagram), I will show how the UML diagrams operate and improve the quality of the patent application.

[155] Galia Shlezinger , Iris Reinhartz-Berger , Dov Dori, *Modeling Design Patterns for Semi-Automatic Reuse in System Design*, Journal of Database Management, v.21 n.1, p.29-57, (Jan. 2010)

[156] "Neural networks are just one of many tools and approaches used in machine learning algorithms. The neural network itself is not the algorithm, but a piece used in many different machine learning algorithms to process complex data inputs into a space that computers can understand." https://deepai.org/machine-learning-glossary-and-terms/neural-network

Figure 33: The original diagram from the patent application- the Deep Crossing Model, used to initialize the embedding layers of DEF

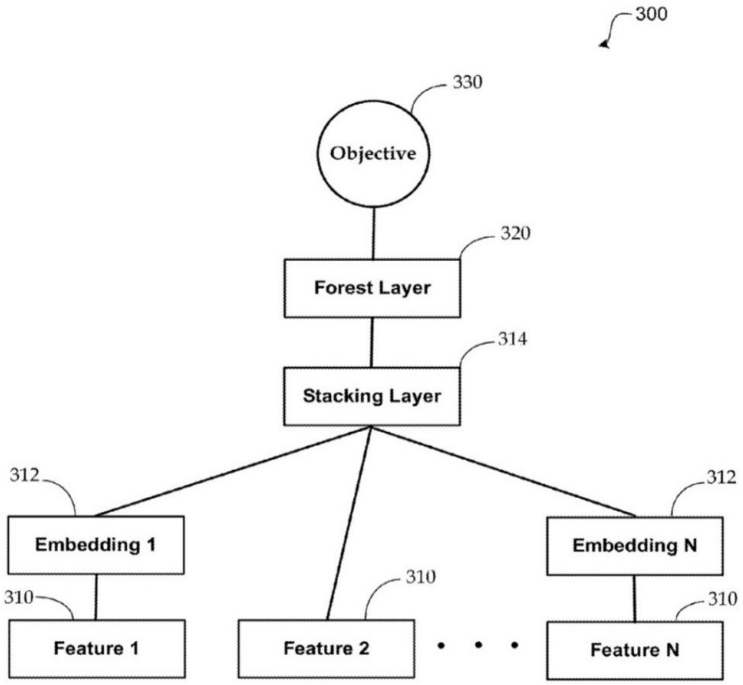


Figure 34: Suggested Sequence diagram for the invention

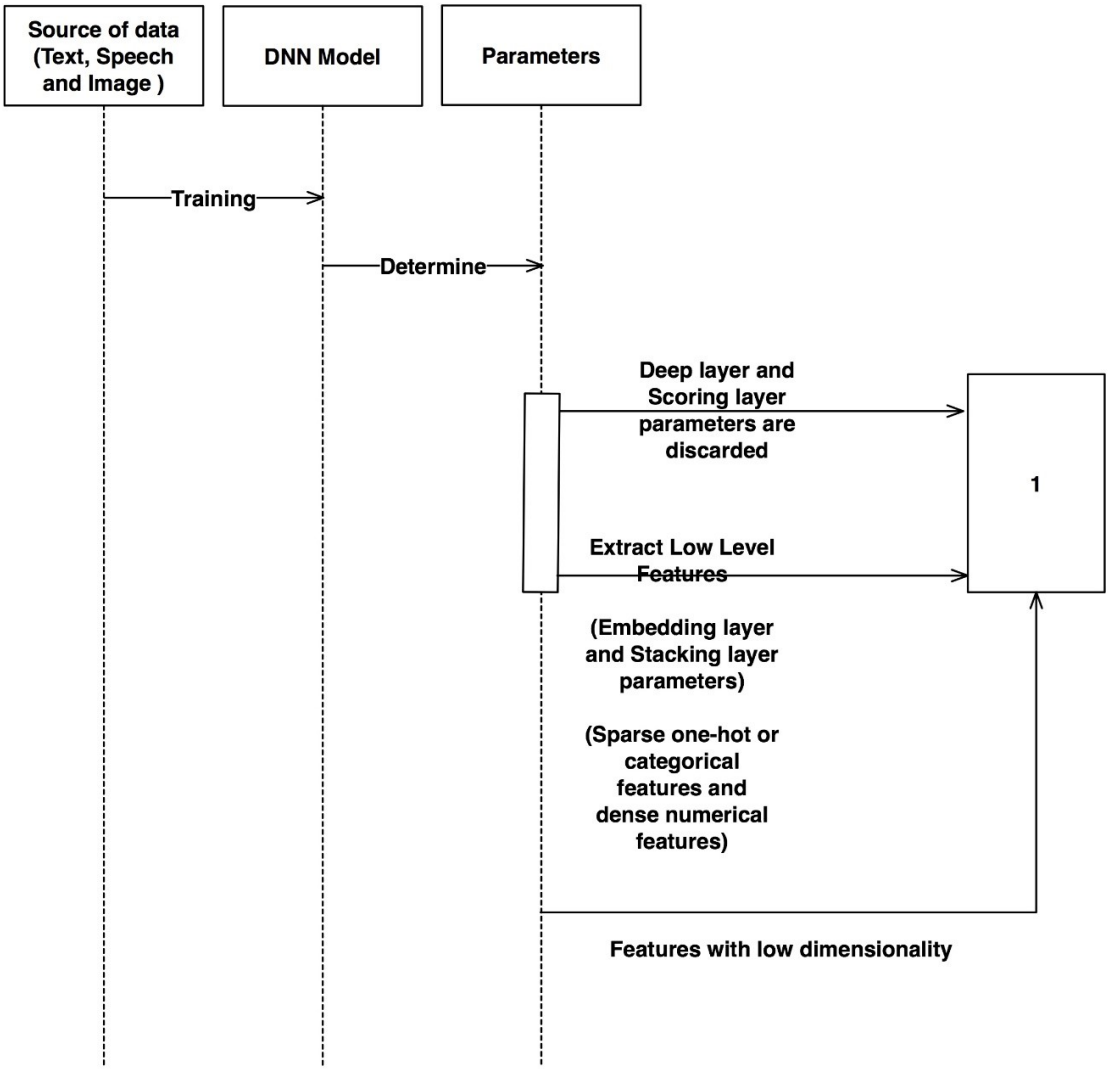


Figure 34: Continued

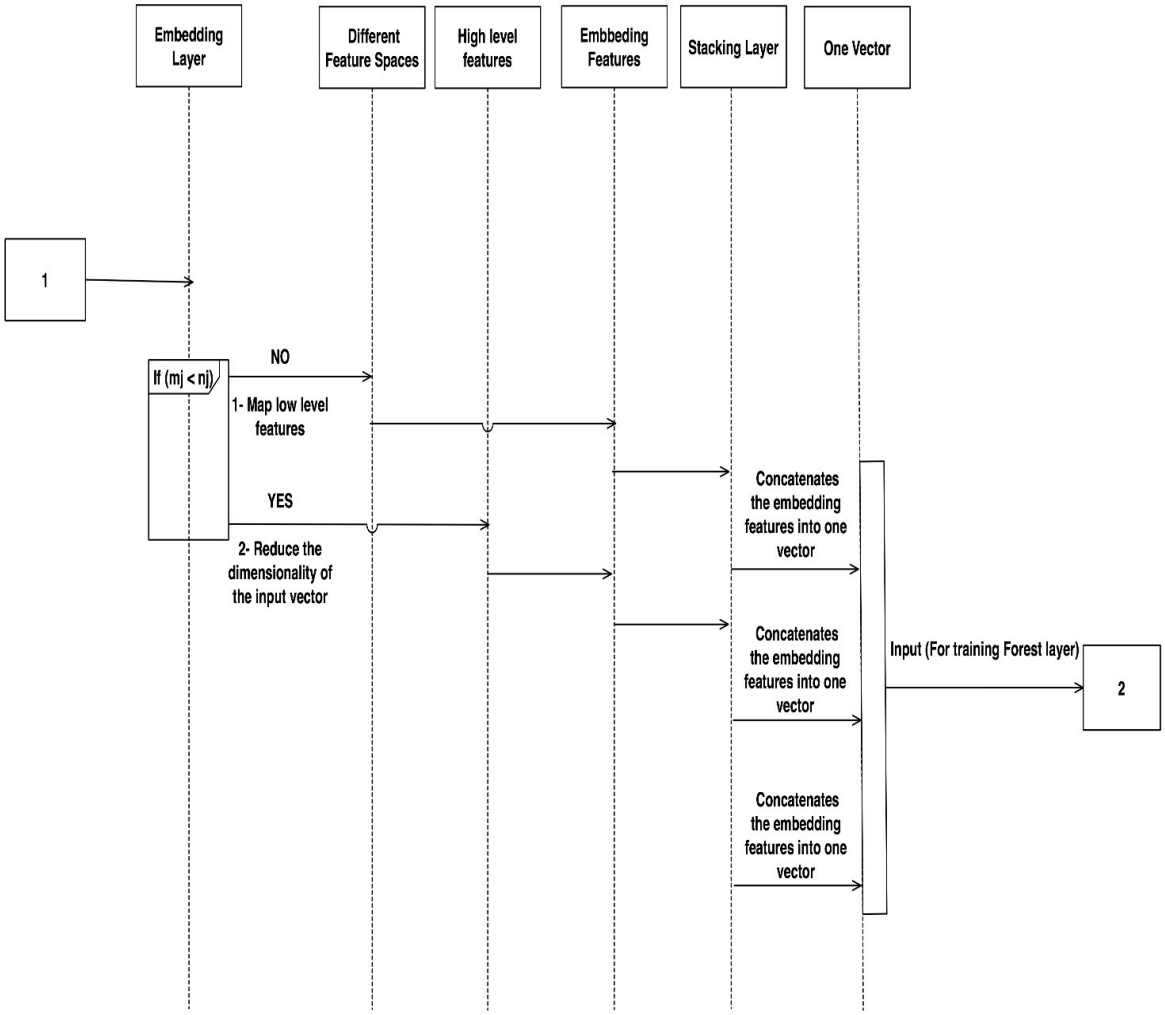
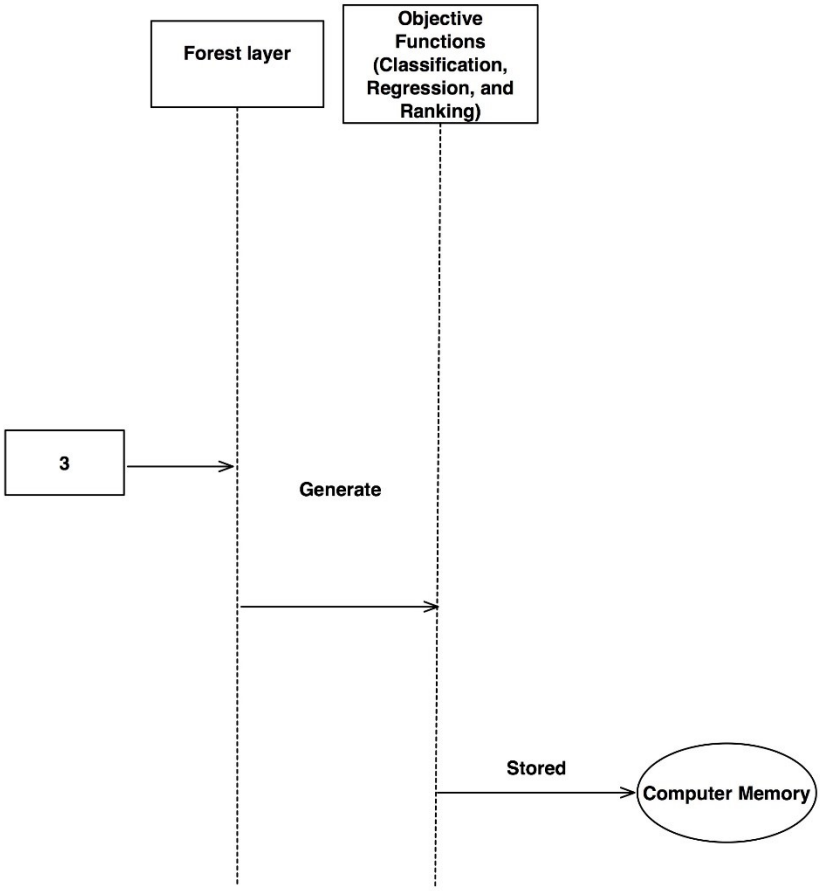


Figure 34: Continued



As you can see in the original diagram of the invention, the logical relationship between components is unclear. As a result, we cannot understand the relationship and the activity between the layers of the invention and the process of replacing two layers with the forest layer to achieve the final objective. By applying the UML diagram, sequence diagram (Figure 34), the system and the sequence of events are easier to comprehend.

3.3.4 Method used by the invention statement: AI Invention

A statement that mentions the specific way or approach that the inventor uses to implement or improve his AI invention, and further determining the difference between his invention and the prior art in a sentence or two in the patent application, will be helpful and essential.

For example, in the abstract of this invention, the functionality of the system is explained. For further clarification, the patentee could disclose the following statement to his application: Improving online serving time and the accuracy of the result of predicting user search selection by discarding the deep layer and the scoring layer of the DNN model and replacing it with the forest layer with deep embedding features (DEF model.)

CONCLUSION

Case name	Is it a real case?	Patent upheld?	Anything more than routine building blocks added?	Ideal disclosure? (Pseudocode, Testing practice /Test case/ Best mode, UML, Method used by the invention)	Explanation
Berkheimer v. HP Inc	Yes	No	No	None	*Berkheimer* did not use any of the ideal disclosures proposed in this study.
Enfish, LLC v. Microsoft Corporation, et al	Yes	Yes	Yes	Test case/ semi-best mode/ Method used by the invention	*Enfish* used the test case, the best mode and method used by the invention among ideal disclosures.
McRo, Inc. v. Bandai Namco Games America	Yes	Yes	Yes	Test case/ best mode	*McRo* used both the test case and the best mode among ideal disclosures.
AI invention: Deep Embedding Forest-based (DEF) model	No	N/A	Yes	Test case/ best mode/ semi- UML/ Method used by the invention	The patent is real but there is no record of litigation on this patent.

The *Alice* framework as a two-step[157] patent eligibility test asks: 1) whether the claimed invention falls into the § 101 judicial exception of an abstract idea; and if so, 2) is there any significant limitation in the components of the claimed invention or the invention as a whole to transform the abstract idea into the patent-eligible application? For the first step in the *Alice* test, in determining what an abstract idea is and what it is not, I defined the abstract idea in a practical way: as routine building blocks. This definition helps me to propose a particular measurement based on the nature of the underlying type of software to differentiate between the routine building blocks and non-routine building blocks (anything more). Routine building blocks, as I defined, are the ***basic, primary, or minimum steps*** for accomplishing the primary action of the specific system.

[157] Alice Corp. Pty. Ltd. v. CLS Bank Int' l, *supra* note 1. *See generally* Mayo Collaborative Services v. Prometheus Laboratories, Inc., 566 U.S. 66 (2012).

These Routine Building Blocks are so essential as to be incapable of removal without destroying the invention's primary functionality.

Alice's step two for determining patent eligibility requires measuring and assessing that "something more" than an abstract idea to pass §101. In other words, step two of the *Alice* test is looking for the "meaningful limitation"[158] or "inventive step" in the application to pass the §101 threshold. In contrast, my proposal suggests that "anything," trivial or not, beyond routine building blocks, makes the software claim pass the §101 threshold. With this approach, *Alice's* second step test is eliminated in my patent eligibility proposal. The reason for passing the §101 threshold is because there is something in the claim that is more than routine building blocks of the specific type of software. Then, I propose that the additional elements should be examined under §§ 102, 103, and 112 rather than the existing court method of using §101 as the eligibility test to see if they qualify for granting a patent.

Additionally, the proposal suggests proper disclosures through examples to improve the quality of patent applications by specifying some factors that might affect the validity of a software patent regarding the sufficiency of its disclosure. Lastly, in order to illustrate my approach, three patent cases were analyzed: *Berkheimer v. HP Inc, Enfish, LLC v. Microsoft Corporation et al., McRo, Inc. v. Bandai Namco Games America* and one Artificial Intelligence (AI) pending patent application, Deep Embedding Forest-based (DEF) model.

This new proposed paradigm for the patent eligibility test (§101) is based on a clear and practical definition of "abstract idea," the § 101 judicial exception, and based on the functional nature of computer software. With this new proposed paradigm, the inconsistency and inefficiency of the current *Alice* two-step framework will be resolved, and the certainty of predicting the outcome of the case will be increased. With a proper guideline for more detailed and efficient disclosures in software patent applications, this will in turn affect the validity rate of software innovations. Detailed and efficient disclosures are needed to justify detailed claims, and if a claim is not supported by sufficient disclosure, it might be invalidated under both §101 and §112. With the help of this legal paradigm, the unavoidably subjective assessments of computer programs in determining patent eligibility are formalized.

[158] *Ibid.*

BIBLIOGRAPHY

Books & Journals

1. Cohen *et al*., *Copyright in a Global Information Economy*, Wolters Kluwer, United States. (3nd ed,2010).
2. Cohen, Julie and Lemley, Mark A. , *Patent Scope and Innovation in the Software Industry*, (2001) 89 Cal. L. Rev. 1, 4.
3. Crouch, Dennis & Merges, Robert, *Operating Efficiently Post-Bilski by Ordering Patent Doctrine Decision-Making*, 25 Berkeley Tech.L.J.1673 (2010).
4. Everett, G.D., R. McLeod, Jr., *Software Testing: Testing across the Entire Software Development Life Cycle*, IEEE press, John Wiley & Sons, Inc., Hoboken, New Jersey, 2007.
5. Groves, Peter J, *Sourcebook on Intellectual Property Law*, p.304(1997) Cavendish Publishing limited, London.
6. Gupta, Maram Suresh, *Sufficiency of Disclosure in Patent Specification*, (2009) Vol 14, Journal of Intellectual Property Rights.
7. Holland. J., *Adaption in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press, 1975.
8. Kemeny. M., *Computer and non-patentable material: Rejections under Article I of the constitution*, Journal of the Patent and Trademark Office Society, 1992. 74:669-72.
9. Kenneth Nichols, *Inventing Software: The Rise of Computer –Related patents*, 1998.
10. Lefstin, Jeff and Menell, Peter, & Taylor, David, *Final Report of the Berkeley Center Workshop: Addressing Patent Eligibility Challenges*, 33 BERK. TECH. L.J. 551 (2018).
11. Lu, Bingbin, *Best Mode Disclosure for Patent Applications: An International and Comparative Perspective* (June 12, 2011). Journal of Intellectual Property Rights, Vol. 16, pp. 409-417, 2011.
12. Lemley, M.A., *Software Patents and the Return of Functional Claiming*, Stanford Public Law Working Paper No. 2117302 (2012).
13. Merges, Robert, *Go Ask Alice: What Can You Patent After Alice v. CLS Bank?, Berkeley* Technology Law Journal*,* SCOTUS blog. (6/2014).
14. Menell, Peter S. and Lemley, Mark A. and Merges, Robert P., *Intellectual Property in the New Technological Age*, 2019.
15. Merges Robert. P. & Duffy John Fitzgerald, *Patent Law and Policy: Cases and Materials*, 7th ed, 2017.
16. Merges, Robert P. & Duffy, John Fitzgerald, *Patent Law and Policy: Cases and Materials*, 6th ed, 2013.
17. Richman, J.H., *Legal Hybrid between the Patent and Copyright Paradigms Colum*, L.Rev. 2432. (1994).
18. Samuelson, Pamela, *Questioning Copyrights in Standards*, Boston College Law Review, Vol. 48:193, 215 (2007).
19. Samuelson, Pamela, '*Benson Revisited: The Case against Patent Protection for Algorithms and Other Computer Program-Related Inventions'*, 9 Emory L.J. 1025 (1990).
20. Samuelson Pamela et al., *A Manifesto Concerning the Legal Protection of Computer Programs*, Colum. L. Rev. 2308 (1994).
21. Seth, Rosenblatt, *Court sides with Oracle over Android in Java patent appeal, CNET*. (May 9, 2014).

22. Shlezinger, Galia and Reinhartz-Berger, Iris and Dori, Dov, *Modeling Design Patterns for Semi-Automatic Reuse in System Design,* Journal of Database Management, v.21 n.1, p.29-57, Jan 2010.
23. Strobos, J.S, *Talking the elusive patentable software: Are there still Diehr or was it just a Flook?* Harvard Journal of Law and Technology 6 (Spring 1993): 363-419.
24. Taylor, David O., *Patent Eligibility and Investment*, CARDOZO L. REV.

Case law

1. Accenture Global Services, GmbH v. Guidewire Software, Inc., 728 F.3d 1336 (Fed. Cir. 2013).
2. Alice Corp. Pty. Ltd. v. CLS Bank Int' l, No. 13-298 (June 19, 2014).
3. Baker v. Selden, 101 U.S. 99 (1879).
4. Berkheimer v. HP Inc., 881F.3d1360 (Fed. Cir. 2018).
5. Computer Associates Int'l, Inc. v. Altai, Inc, 982 F.2d 693 2d Cir. 1992.
6. Enfish, LLC v. Microsoft Corp., 822 F.3d 1327, 2016 U.S. App. LEXIS 8699, 118 U.S.P.Q.2D (BNA) 1684.
7. Interval Licensing, LLC v. AOL, Inc., 896 F.3d 1335, 1343 (Fed. Cir. 2018)
8. Lotus Dev't Corp. v. Borland Int'l516 U.S. 233 (1996).
9. Molecular Pathology v. Myriad Genetics, Inc., 569 U.S., 133 S. Ct. 2107, 2116, 106 USPQ2d 1972, 1979 (2013).
10. Mayo Collaborative Services v. Prometheus Laboratories, Inc., 566 U.S. 66 (2012).
11. McRO, Inc. v. Bandai Namco Games America Inc. (Fed. Cir. 2016).

Oracle v. Google, 847 F.Supp.2d 1178 (2012).

Reports

1. Reply brief of Petitioner McRO, Inc, McRO, Inc. v. Bandai Namco Games Am. Inc.(docket number 755) (2016)(Nos. 15-1080, 15-1099). Retrieved from https://www.law.berkeley.edu/wp-content/uploads/2016/05/McRO-v-Bandai-Namco-appellate-argument-FJC-Program.pdf.

Websites

1. https://www.bitlaw.com/source/cases/copyright/altai.html
2. http://www.scotusblog.com/case-files/cases/alice-corporation-pty-ltd-v-cls-bank-international/
3. https://www.uspto.gov/sites/default/files/documents/memo-berkheimer-20180419.PDF
4. Margaret Rouse, *database (DB),* TechTarget, https://searchsqlserver.techtarget.com/definition/database
5. Shailendra Chauhan, *Difference between Primary Key and Foreign Key*, DotNetTricks, https://www.dotnettricks.com/learn/sqlserver/difference-between-primary-key-and-foreign-key

6. Figure resource: https://medium.com/analytics-vidhya/programming-with-databases-in-python-using-sqlite-4cecbef51ab9
7. https://www.datacamp.com/community/tutorials/introduction-indexing-sql.
8. https://www.bitlaw.com/source/cases/patent/MCRO.html
9. Robert Saches, *Alice on Software Patents: Preemption and Abstract Ideas*, IPWatching, June. 29, 2014, https://www.ipwatchdog.com/2014/06/29/alice-on-software-patents-preemption-and-abstract-ideas/id=50210/
10. https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm.
11. *Definition of 'Pseudocode'*. The Economics Time, https://economictimes.indiatimes.com/definition/pseudocode
12. *What is Pseudocode? Software Engineering*, 2/22/12, https://softwareengineering.stackexchange.com/questions/136292/what-is-pseudocode/136354
13. *How to write a good test case*, Apache Open Office, 7/27/2012, https://wiki.openoffice.org/wiki/QA/Testcase/How_to_write_test_case
14. Bingbin Lu, *Best Mode Disclosure for Patent Applications: An International and Comparative Perspective*, Journal of Intellectual Property Rights Vol 16, September 2011, pp 409-417. Available at SSRN: https://ssrn.com/abstract=1938859
15. https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language
16. https://www.sparxsystems.eu/resources/project-development-with-uml-and-ea/introduction-to-uml/
17. https://stackoverflow.com/questions/29145545/is-uml-for-object-oriented-only
18. https://stackoverflow.com/questions/26212374/are-uml-diagrams-only-for-object-oriented-approach?noredirect=1&lq=1
19. https://softwareengineering.stackexchange.com/questions/78407/software-design-for-procedural-programming
20. *What is Unified Modeling Language?* Lucidchart, https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language.
21. Nishadha, *UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples*, Creately blog, https://creately.com/blog/diagrams/uml-diagram-types-examples/#SequenceDiagram.
22. https://creately.com/blog/diagrams/the-basics-the-purpose-of-sequence-diagrams-part-1/

23. "U.S. Patent Application No. 15/253,669, Publication No. 20180060728, http://appft1.uspto.gov/netacgi/nphParser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20180060728.PGNR.&OS=DN/20180060728&RS=DN/20180060728
24. J. Zhu et al., *Deep Embedding Forest: Forest-based Serving with Deep Embedding Features (*March 16, 2017), https://arxiv.org/pdf/1703.05291.pdf
25. *Unified Modeling Language*, Tutorialspoint, https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm
26. Krunal Patel, *Step Ahead In Competition With AI Test Automation*, Clarion Technologies, https://www.clariontech.com/blog/step-ahead-in-competition-with-ai-test-automation.
27. https://deepai.org/machine-learning-glossary-and-terms/neural-network
28. Margaret Rouse, extrapolation and interpolation, TechTarget, https://whatis.techtarget.com/definition/extrapolation-and-interpolation.

APPENDIX A

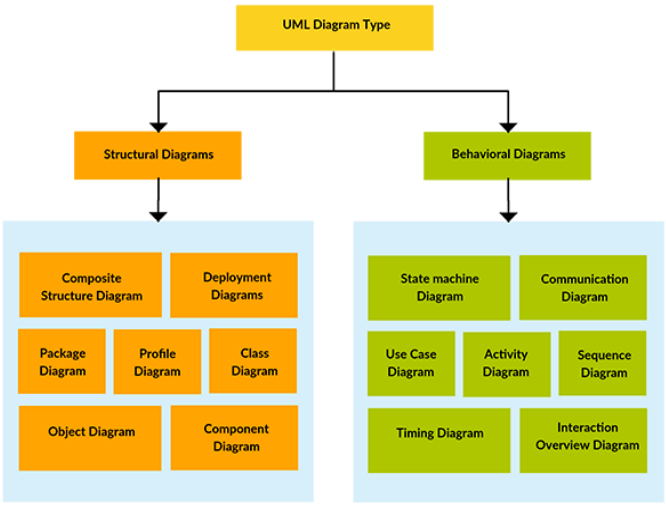
UML Diagram Types

These UML diagrams can be divided into two main categories; structural diagrams and behavioral diagrams.

Structural diagrams: Show the things in a system being modeled. In a more technical term, they show different objects in a system.

Behavioral diagrams: Shows what should happen in a system. They describe how the objects interact with each other to create a functioning system.

Figure A-1 UML Diagram Types



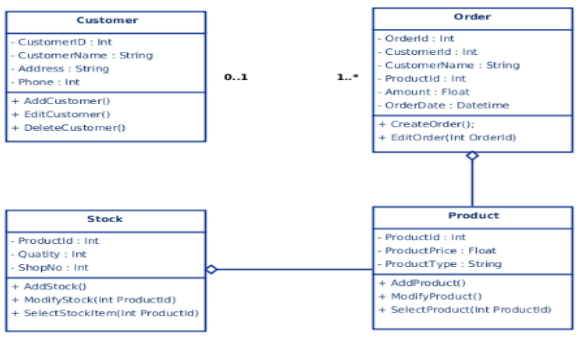
UML Diagram types

Some of the most important of these diagrams are introduced in the following:

Structural diagrams:

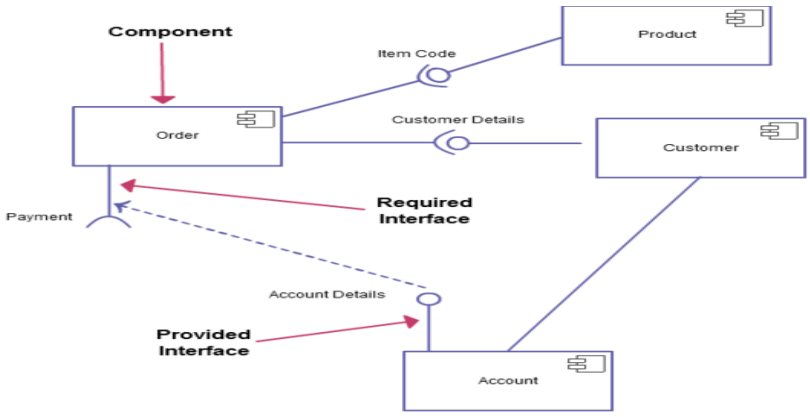
1- **Class diagram:** It is the most used UML diagram type. It shows the classes in a system, attributes, and operations of each class and the relationship between each class. A class is a template consisting of the attributes and methods or functions related to that class.

Figure A-2 Class Diagram



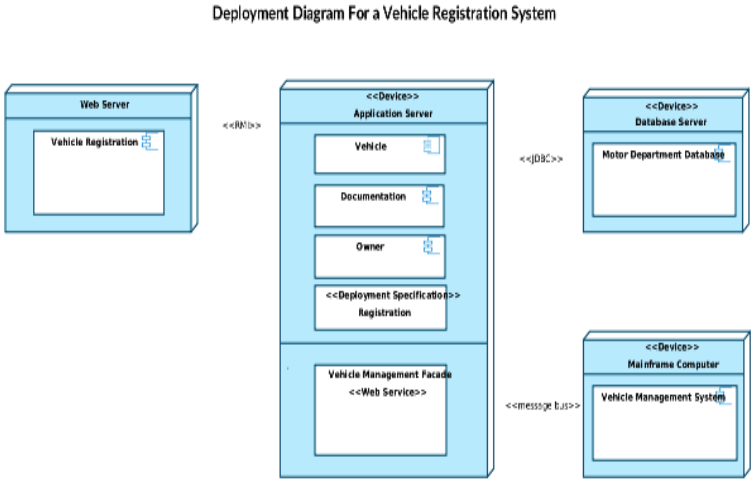
2- Component diagram: A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems that have many components.

Figure A-3 Component diagram



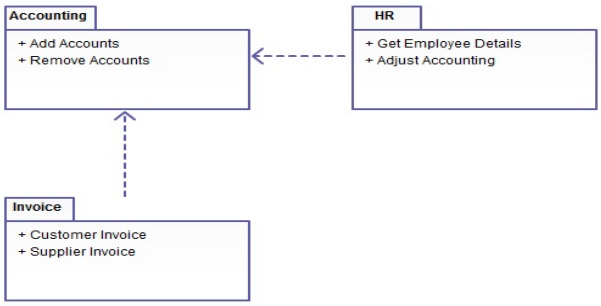
3- Deployment diagram: A deployment diagram shows the hardware of your system and the software in that hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

Figure A-4 Deployment Diagram



4- Package diagram: a package diagram shows the dependencies between different packages in a system.

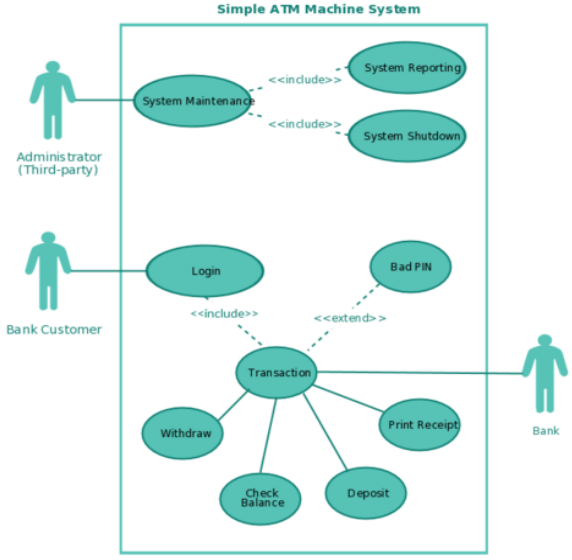
Figure A-5 Package diagram



Behavioral diagrams

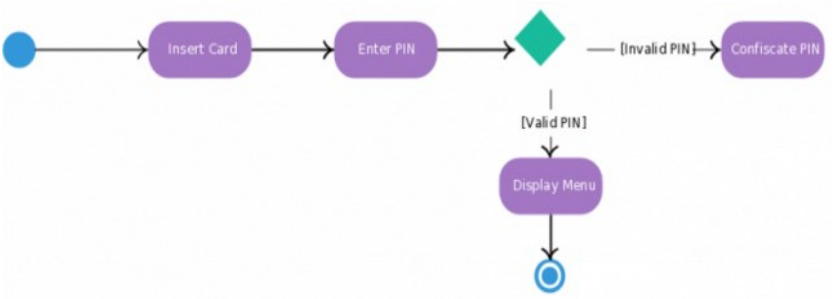
1- **Use case diagram:** Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

Figure A-6 Use case diagram



2- **Activity diagram:** Activity diagrams represent workflows in a graphical way. They can be used to describe business workflow or the operational workflow of any component in a system.

Figure A-7 Activity diagram



2- **Sequence diagram:** Sequence diagrams in UML show how objects interact with each other and the order those interactions occur.

Figure A-8 Sequence diagram