Análisis del Software Libre a través del concepto de Rizoma de Guilles Deleuze y Félix Guattari

Gustavo Adolfo Romero Pabón

Trabajo de Grado para Optar el título de filósofo

Director

Alonso Silva Rojas

Doctor en Ciencias Políticas

Universidad Industrial de Santander
Facultad de Ciencias Humanas
Escuela de Filosofía
2019

A mi madre y abuela que la vida me permita recompensarlas como ellas se lo merecen

A Tomás Pabón con profundo amor

Agradecimientos

Una gratitud infinita a mí madre quien ha sido mi apoyo y soporte.

A mi hermano quien me ha dado las lecciones más importantes en mi vida, infinitas gracias a él por compartir conmigo los caminos de la militancia política. Muchas gracias a Tomás Pabón quien desde que llego a mí vida me ha brindado un acariño incondicional, sin él este proyecto no habría sido posible.

Gracias a Sergio Andrés Rueda, Katheryn Picón, José Andrés Ortiz, quienes siempre han incentivado el pensamiento crítico desde el comienzo, gracias por las largas jornadas de trabajo, por las conversaciones y las tazas de café que hemos compartido, infinitas gracias por la amistad sincera que me han brindado.

Agradezco al profesor Alonso Silva Rojas quien muy amablemente aceptó dirigir este proyecto, además de permitirme desarrollar de la forma más libre posible las ideas que aquí se presentan, su apoyo y acompañamiento fue de vital importancia para la realización de este trabajo monográfico.

Gracias a toda la Escuela de Filosofía por contribuir a mí formación académica y personal, sus aportes a mi formación son invaluables. Quiero brindar un especial agradecimiento al profesor Christian Quintero y a la profesora Patricia Carreño.

Tabla de Contenido

Resumen

TÍTULO: Análisis del Software Libre a través del concepto de rizoma de Guilles Deleuze y Félix Guattari[*]

AUTOR: Gustavo Adolfo Romero Pabón[**]

PALABRAS CLAVE: Software Libre, Linux, Rizoma, Arbóreo, Deleuze y Guattari.

DESCRIPCIÓN:

Deleuze y Guattari despliegan el concepto de *Rizoma* en el segundo tomo de *Capitalismo y esquizofrenia* titulado *Mil mesetas.* Un rizoma es un modelo epistemológico en el cual los elementos no tienen una jerarquía; este modelo se contrapone al modelo arbóreo o jerárquico, en donde los elementos se encuentran subordinados unos hacia otros. Así las cosas, el propósito de la presente monografía en demostrar que el Software Libre se constituye como un modelo rizomático y en este sentido podría entenderse como una tecnología rizomática. Para ello en el primer capítulo recorreremos la historia del Software Libre, con el propósito de ubicar los elementos que dieron origen y que permitieron que el software libre se desarrollara tal y como lo conocemos hoy. En segundo lugar, presentamos el concepto de rizoma y con el analizamos los elementos del software libre que lo constituiría como un modelo rizomático y que permitirían establecerlo como una tecnología rizomática.

[*] Trabajo de grado
[**] Facultad de Ciencias Humanas. Escuela de Filosofía. Director: Doctor en Ciencias Políticas Alonso Silva Rojas

Abstract

TITLE: Free software analysis through the rhizome concept of Guilles Deleuze and Félix Guattari*

AUTOR: Gustavo Adolfo Romero Pabón**

KEYWORDS: Free Software, Linux, Rhizome, Arboreal, Deleuze and Guattari.

DESCRIPTION:

Deleuze and Guattari deploy the concept of rhizome in the second volume of Capitalism and Schizophrenia entitled A Thousand Plateaus. A rhizome is an epistemological model in which the elements do not have a hierarchy; This model is opposed to the arboreal or hierarchical model, where the elements are subordinated to each other. So, the purpose of this paper to demonstrate that Free Software is established as a rhizomatic model and in this sense could be understood as a rhizomatic technology. For this, in the first chapter we will cover the history of Free Software, with the purpose of locating the elements that gave rise to and that allowed free software to develop as we know it today. Secondly, we introduce the concept of rhizome and to analyze the elements of free software that constitute it as a rhizomatic model that would allow to establish it as a rhizomatic technology.

* Final undergraduate project
** Faculty of Humanities. School of Philosophy. Director: Doctor of Political Science.

Introducción

El Software Libre nace a principios de los años ochenta cuando Richard Stallman anuncia el proyecto GNU con el objetivo de otorgar libertades a los usuarios, ya que en ese momento el mundo del software se estaba viendo afectado por una modificación al licenciar el software de manera restrictiva, lo cual se traducía en que los desarrolladores ya no podían tener acceso al código fuente de los programas, algo que era común entre los programadores.

Posteriormente, Stallman fundará la Free Software Fundation (FSF por sus siglas en inglés) organización creada para defender las libertades del software y para difundir el movimiento del Software Libre. Según la FSF, un programa libre es todo aquel que respete la libertad de los usuarios de ejecutar, copiar, distribuir y modificar el software.

El concepto de Software Libre se formula en oposición al software privativo, que serían aquellos programas licenciados para reservar los derechos de uso, copia y modificación haciendo que el desarrollador siempre sea el propietario del programa, aun cuando éste venda el programa a distintos usuarios. Lo que implica que el usuario no es dueño del programa, así lo haya adquirido a través de una transacción monetaria y no es dueño en la medida en que éste no puede realizar modificaciones, no puede hacer copias y tampoco puede estudiarlo para saber cómo desarrolla sus funciones el programa o el paquete de software que haya adquirido.

En este punto es importante señalar que estas libertades promulgadas por la FSF para el software libre se expandieron y empezaron a ser utilizadas en otros ámbitos. De tal modo que tras veinte años de haberse iniciado el proyecto GNU se empieza a hablar de cultura libre como un

proceso de producción de individuos y colectivos en el cual se encuentran manifestaciones literarias, musicales, cinematográficas, científicas, educativas, etc.

Ahora bien, nuestra época se encuentra marcada por un protagonismo cada vez mayor de la tecnología y en este proceso el software ha empezado a jugar un rol fundamental en este desarrollo y por ende en la construcción de nuestras sociedades. Por lo que es importante tener en cuenta, como señala el profesor Lev Manovich en su libro *Software Takes Commands*, que desde la década del 90 existe un proceso de softwareización de la tecnología. Lev Manovich para demostrar esto señala que las marcas globales más destacadas a nivel mundial se encontraban en el negocio de producción de bienes materiales, mientas que en la actualidad son compañías de tecnologías de la información como Microsoft, Apple, Google, Facebook, entre otras.

En este sentido creemos que el proceso de softwareización que se vive desde la década de los 90 ha tenido gran influencia en el desarrollo del capitalismo contemporáneo, influyendo en la economía, la política y la cultura. Esto implica que pensemos el software como una parte constitutiva de nuestras sociedades.

Es importante entonces que nos preguntemos, ¿si en la actualidad el software juega un papel fundamental en la construcción de nuestras sociedades cómo lo debemos entender? Para resolver esta cuestión acudiremos al concepto de *Rizoma* postulado por los filósofos franceses Gilles Deleuze y Félix Guattari en cuanto que consideramos que con este concepto podemos entender el software. Específicamente nos permite comprender el modelo rizomático que se ha construido con el Software Libre que será nuestro objeto de estudio.

Anteriormente hemos señalado que el software libre funciona como un modelo rizomático, y para mostrar esto, en el primer capítulo, abordaremos la historia del software libre partiendo de

unos conceptos básicos como Software, código objeto y código fuente, los cuales nos permitirán ponernos en contexto para poder de esa forma abordar el concepto de software libre.

Posteriormente veremos a través de la FSF y de Richard Stallman la definición de Software Libre y de las libertades que con él se promulgan, esto porque la definición de software libre responde al cumplimiento de estas libertades. Con la figura de Stallman también veremos el nacimiento de la FSF y del proyecto GNU con el cual se intentará construir un sistema operativo completamente libre.

Este proyecto sería culminado, como veremos más adelante, en 1994 con la escritura del *kernel* conocido como Linux, por el estudiante de ciencias de la computación de la Universidad de Helsinki en Finlandia Linus Torvalds, llevando de esta forma a un estado de concretización el sistema operativo que, en 1983, y tras el abandono del laboratorio de IA del MIT se había propuesto construir Richard Stallman.

En el segundo capítulo, realizamos, en primer lugar, una breve presentación del concepto de *Rizoma*, posteriormente se realizará un análisis del Software Libre a través de este concepto. Este análisis se llevará a cabo por medio de la figura de la sinécdoque, pues presentaremos el rizoma Software Libre mediante algunos elementos que nos permitan abrir un mapa que muestre las múltiples entradas y salidas del rizoma. De esta manera se comprenderá mejor lo que es un rizoma y por qué el Software Libre funciona como un modelo rizomático.

1. Historia del Software Libre

Las tecnologías digitales de la información ayudan al mundo haciendo que sea más fácil copiar y modificar información. Los ordenadores prometen hacer esto de forma más sencilla para todos.

Richard Stallman

Aunque la historia de la computación y de las relacionadas con ella son breves debido al tiempo reciente de su desarrollo, la historia del software libre podría considerarse entre todas las historias una de las más largas. Ya que en los inicios del desarrollo de software todo cumplía con la definición de Software Libre, aunque el concepto aún no existía y no sería hasta los principios de la década de 1980 cuando el concepto de Software Libre hiciera su aparición formalmente.

El estado de libertad en el cual se desarrollaba el software en un principio sufrió un giro de 180° y el desarrollo de software a través de un modelo privativo de las libertades, que tanto los usuarios como los desarrolladores tenían, pasó a dominar la escena del desarrollo de software y se convirtió en la forma exclusiva de producir software.

Fue bajo este giro que Richard Stallman sentaría las bases del software libre como lo entendemos hoy en día. Con la formalización del concepto y el lanzamiento del proyecto GNU empezaron a aparecer programas libres y se terminaría de concretizar el proyecto de un sistema operativo completamente libre con la aparición en 1994 del núcleo del sistema de la mano de Linus Torvalds.

Con el paso del tiempo el Software Libre ha ido creciendo, mejorando y ganando más usuarios. El proceso de su génesis y desarrollo ha permitido que el Software Libre se convierta en una posibilidad que hay que considerar en casi todos los ámbitos.

Ahora bien, para presentar la historia del Software Libre en primer lugar partimos del concepto de software, esto para hacer más claro en qué consiste en general el software y nos permita a la vez vislumbrar en particular en que consiste el Software Libre, por ello analizamos tres conceptos fundamentales como lo son: código máquina, compilador y código fuente, este último tiene una gran relevancia, ya que, el código fuente abierto es una de las características distintivas del Software Libre.

En segundo lugar, examinaremos cómo se define el Software Libre y las libertades que este promulga tanto para programadores como para usuarios, pues esta definición y las libertades que la acompañan y la relación que estos dos elementos sostienen, nos permite comprender que tipos de programas pueden adquirir la denominación de Software Libre.

En tercer lugar, recorremos a groso modo la definición de licencia y los tipos de licencias que se pueden utilizar para licenciar el software, centrando especial atención en la GLP, porque esta es la licencia más importante y popular en el universo del software libre.

En cuarto lugar, y a través de la anécdota de la impresora del famoso programador norteamericano Richard Stallman conoceremos cómo nace el proyecto GNU y el movimiento por el Software libre.

Finalmente observaremos como se completa el proyecto GNU a través del desarrollo del *kernel* creado por el estudiante de ciencias de la computación de la Universidad de Helsinki Linus Torvalds y al que le dio el nombre de Linux, que permitirá que se concrete el proyecto de tener un sistema operativo completamente libre.

1.1 ¿Qué es el software?

Explicar qué es el Software Libre, sin antes explicar qué es el software resultaría en una tarea muy complicada e incluso imposible; por lo cual es necesario que señalemos algunos aspectos claves del software.

Según la Real Academia Española, el software es un conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora (Española, 2014, pág. 950). Una definición más canónica del software nos la brinda el Institute of Electrical and Electronics Engineers en su *Standard Glossary of Software Engineering Terminology* define al software como: Computer programs, procedures, and posibility associated documentation and data pertaining to the operation of a computer system (IEEE, 1990, pág. 66).

El profesor y académico británico Ian Sommerville, autor del libro *Ingeniería de Software* lo define como: Programas de cómputo y documentación asociada (Sommerville, 2011, pág. 6) y el ingeniero de software Roger Pressman en su texto *Ingeniería del Software. Un enfoque práctico* señala que el software es:

> El producto que construyen los programadores durante un largo tiempo. Incluyen programas que se ejecutan en una computadora de cualquier tamaño y arquitectura, contenido que se presenta a medida que se ejecutan los programas de cómputo e información descriptiva tanto en una copia dura como en formatos virtuales que engloban virtualmente a cualquier medio electrónico (Pressman, 2010, pág. 1).

Con las definiciones anteriormente presentadas podemos entender al software como el conjunto de programas que se pueden ejecutar en una computadora, así como toda la información y recursos necesarios para su diseño, instalación, operación, mantenimiento y refinamiento. Ahora que tenemos claridad en la definición de software podemos analizar algunos aspectos claves del

software, por ejemplo, qué es y cómo se produce el código fuente y el código objeto de los programas.

1.1.1 Código fuente, código objeto y compilador. Fernanda da Rosa y Federico Heinz señalan que por mucho que hablemos de que una computadora es un "dispositivo inteligente", no habría cosa más tonta que su procesador. Pues, si vamos a la esencia de un procesador éste sólo sabe manejar un alfabeto compuesto por dos letras "1" y "0" y dadas estas dos "letras" calcular su suma:

> Por lo general, las computadoras no operan sobre "letras" individuales (llamadas "bits"), sino sobre palabras de ocho letras, llamadas "bytes", por ejemplo "01100101". Hay un total de 256 palabras distintas que se pueden escribir con ocho bits. Estas 256 palabras que, si las interpretamos como números binarios, representan los números del cero al 255, constituyen el vocabulario completo de la computadora. (Rosa & Heinz, 2007, pág. 13)

Ahora bien ¿Cómo es posible que una computadora realice todas las actividades a las que estamos acostumbrados con ellas con un lenguaje tan restringido (numéricamente)? ¿Cómo es posible que solo sumando pueda realizar cálculos complejos?

Aunque pareciera que todas las tareas para las cuales utilizamos y en las cuales resultan muy útiles las computadoras fueran un milagro, éste tiene una sencilla explicación y es que las computadoras son programables, es decir, que las computadoras saben obedecer órdenes y además tienen la capacidad de ejecutar millones de éstas en solo segundos.

> Las computadoras son dispositivos de propósito general, a las que podemos configurar para cumplir tareas específicas: imprimir documentos, tocar música, mostrar vídeos, interconectar redes, por sí misma son incapaces de realizar ninguna. Para ello, necesitan que alguien les provea de instrucciones detalladas acerca de cómo interpretar la información y cómo comunicarse con el usuario. (Rosa & Heinz, 2007, pág. 14)

Para que una computadora pueda llevar a cabo una tarea específica necesita de un programa, que es un conjunto de instrucciones que debe seguir para realizar la tarea de manera satisfactoria. Es importante señalar que los programas se escriben en lenguaje de programación, que es un lenguaje formal[1] en el cual se especifican una serie de reglas, notaciones, símbolos y/o caracteres para que una computadora produzca diversas clases de datos.

Estos lenguajes formales con los cuales se escriben los programas y con los que también se desarrolla el código fuente, es traducido al lenguaje detallado que requiere la máquina para poder cumplir con las ordenes, es decir, que antes de que una computadora pueda ejecutar un programa, es necesario que este traduzca dichas instrucciones a su lenguaje, esto es a código máquina que no es más que una larga serie de unos y ceros.

> A fin de cuentas -escriben Jordi Adell y Iolanda Bernabé-, un ordenador no es más que una gran cantidad de interruptores electrónicos, que pueden estar únicamente en dos estados: dejar pasar la corriente o no dejarla pasar, uno o cero. Un programa le dice al ordenador qué interruptores poner a "cerro" y cuáles poner a "uno" en cada momento, dependiendo de las acciones que ejecutemos en el teclado o con el ratón. El proceso de convertir un programa escrito en un lenguaje de programación a instrucciones inteligibles para el ordenador, se denomina "compilación" y lo hacen otros programas de ordenador especializados: los compiladores. Una vez el programa está compilado ya es posible ejecutarlo, a cambio, una vez "traducido" a código máquina, es casi imposible que un ser humano entienda algo de la larga serie de unos y ceros que se ha convertido. (Adell & Bernabé, 2007, pág. 174)

Recapitulando tenemos que el programador escribe el código fuente con un lenguaje de programación, y, en un proceso de compilación el código fuente se convierte lenguaje máquina o

[1] Un lenguaje formal es un conjunto, finito o infinito de cadenas definidas sobre un alfabeto finito. Estas cadenas están formadas gracias a un alfabeto y a una gramática que están formalmente especificadas. (Balari, 2014)

también conocido como código objeto. De tal modo que podemos definir al código objeto como el archivo que resulta de compilar el código fuente.

El código objeto es el único "lenguaje" que entienden las computadoras, ya que estas trabajan intermitentemente con dos niveles de voltaje que se representan simbólicamente con dos dígitos: "1" que es el voltaje más alto y "0" el voltaje más bajo. Como podemos observar esto se basa en una lógica binaria implementada por mecanismos electrónicos, por lo tanto, este código binario permite que la máquina ejecute las instrucciones que le damos.

Ahora bien, construir un programa, modificarlo e incluso comprender cómo funciona sería una tarea extremadamente difícil si sólo tuviéramos a nuestra disposición el código objeto, pues, como señalamos anteriormente esto no es más que una larga cadena de unos y ceros.

Por ejemplo, si un programador quisiera hacer el clásico y conocido ejercicio de "Hello World" -este ejercicio de programación consiste en hacer que en la pantalla del ordenador aparezcan las palabras "Hello World"- y decidiera realizar esta tarea utilizando el lenguaje de programación de Java tendría al finalizar la escritura de este código algo similar a esto:

```
public class practica{
  public static void maid(String args[]){
   Systen.out.println("Hello World");
 }
}
```

Pero si sólo tuviéramos el código objeto de este mismo ejercicio, es decir, de la palabra "Hello World" obtendríamos algo como esto:

```
01001000011001010110110001101100011011110010000001010010010000110010101101100
11011000110111100100000001010
```

Como podemos observar, escribir programas, leerlos y modificarlos teniendo únicamente el lenguaje máquina resultaría una tarea titánica para los programadores por el alto grado de dificultad que esto comportaría, por lo cual, para disminuir la dificultad que implica hacer los programas con el lenguaje máquina, se inventaron los lenguajes de programación -de los cuales existen una gran variedad, entre los más famosos tenemos Java, C++, Python y C- cuya función es facilitar la escritura, la lectura y la modificación de programas.

Sí miramos el código de "Hello World" expresado en Java sería más fácil comprender lo que esté dice, incluso para personas que no conocen el lenguaje de programación, que el programa expresado en lenguaje máquina.

Este mismo programa puede escribirse en otros lenguajes de programación por ejemplo en C++:

```
#include <iostream>
 int main ()
{
  std: :cout << "Hello World"\n";
  return 0;
}
```

O Python que es uno de los lenguajes de programación más utilizados en la actualidad:

```
print "Hello World"
```

Salta fácilmente a la vista que entre un lenguaje de programación y otro hay una serie de diferencias, pero también podemos ver algunas coincidencias, esto es porque como habíamos señalado anteriormente el propósito de los lenguajes de programación es facilitar la comprensión y modificación de los programas.

Los lenguajes de programación no son comprensibles directamente a la computadora, pues éstas sólo pueden obedecer las órdenes que sean dadas en su propio lenguaje, por lo cual antes de que la computadora pueda ejecutar un programa escrito en lenguaje de programación debe ser traducido a lenguaje máquina.

> Para cada combinación de procesador, lenguaje y sistema operativo existen traductores automáticos. Llamados compiladores. Se trata de programas que leen un programa escrito en lenguaje de programación y, a partir de él, generan un escrito en el lenguaje de ejecución adecuado para una determinada combinación de procesador y sistema operativo. (Rosa & Heinz, 2007, pág. 23)

Los lenguajes de programación que utilizamos anteriormente para los ejemplos, se denominan lenguajes de alto nivel[2], esto es, un lenguaje que permite que los programadores escriban y entiendan los programas con mayor facilidad, esto por la cercanía de estos lenguajes con el lenguaje natural[3].

Pero este lenguaje de alto nivel tiene un problema y es que la máquina no lo entiende, es decir, que existe una disonancia entre el lenguaje con el que los humanos escribimos las instrucciones

[2] Los lenguajes de alto nivel son los más utilizados por los programadores, pues están diseñados para que los programadores escriban y entiendan los programas de un modo más fácil. Una característica de los lenguajes de programación de alto nivel es que se encuentran cercanos al idioma inglés y además son lenguajes básicamente simbólicos. Existe una gran variedad de lenguajes de programación de alto nivel, esto porque cada uno de ellos cumple con tareas específicas, por ejemplo, FORTRAN (Formula Translation Language) fue de los primeros lenguajes de alto nivel que se utilizó para expresar notaciones algebraicas. COBOL (Common Busines Oriented Language) creado en 1959 es un lenguaje utilizado para aplicaciones empresariales. El lector que desee conocer más de los lenguajes de alto nivel puede acudir a, *Abstraction Level Taxonomy of Programming Language Frameworks* en *International Journal of Programming Languages and Applications*, y a *A High-Level Programming and Commad Langueage* en *Communications of the ACM*.

[3] Un lenguaje natural -dice Gleen Brookshear- es aquel que ha evolucionado con el paso del tiempo para fines de comunicación humana, por ejemplo, el español, el inglés o el alemán (Brookshear, 1993, pág. 12). Entonces el lenguaje natural -en nuestro caso el español- es aquel que utilizamos para comunicarnos con las demás personas. Ahora bien, el lector que quiera saber más de cómo se relaciona el lenguaje natural con los lenguajes de programación puede acudir a, *Lenguaje natural e indización automatizada* en *Ciencias de la información*, y a *Computational Complexity of Natural Languages: A Reasoned Overview* en *Proceedings of the Workshop on Linguistic Complexity and Natural Language Processing*.

(programas) para los ordenadores y el lenguaje que ellos pueden entender (lenguaje máquina). Esta disonancia entre estos lenguajes tiene una solución que brinda un programa llamado compilador.

Como señala Alfred Aho en su libro *Compiladores. Principios, técnicas y herramientas*: Un compilador es un programa que puede leer un programa en un lenguaje (el lenguaje *fuente*) y traducirlo en un programa equivalente en otro lenguaje (el lenguaje *destino*) (Aho, 2008, pág. 1). Como podemos observar entonces la tarea del compilador es la de traducir el lenguaje fuente al lenguaje máquina que si es comprensible para una computadora y que de esta manera esta pueda ejecutar las instrucciones que se le han dado.

En este proceso de traducción del lenguaje fuente a lenguaje máquina el programador puede utilizar un *intérprete* que es otra herramienta que permite ejecutar esta acción. Un intérprete es otro tipo común de procesador de lenguaje. En vez de producir un programa destino como una traducción, el intérprete nos da la apariencia de ejecutar directamente las operaciones especificadas en el programa de origen (fuente) con las entradas proporcionadas por el usuario (Aho, 2008, pág. 2).

Ahora bien, si la tarea tanto de un compilador como del intérprete es traducir el lenguaje fuente a lenguaje máquina entonces cabe preguntarnos ¿qué es el lenguaje fuente? Para responder esta pregunta primero debemos señalar que el lenguaje escrito en lenguaje de programación se conoce como *Source code* en inglés y en español es traducido como código fuente. Entonces el código fuente es el conjunto de líneas de texto que son las directrices de las instrucciones que debe seguir la computadora para ejecutar el programa.

Ciertamente lo que le interesa a un usuario común de una computadora no es el código fuente de los programas, sino el nivel “binario” (código ejecutable), en tanto que este es el medio por el

cual una computadora cumple las ordenes que le damos; pero es conveniente tener clara esta distinción entre el código fuente y el código objeto para entender porque los partidarios del Software Libre se empeñan en disponer del código fuente.

Y es que el Software Libre es más que el derecho a disponer de la fuente del código, es también la libertad de estudiarlo, modificarlo y redistribuirlo. Y como señala Miquel Vidal esos derechos, o su ausencia, condicionan a cualquiera que use un ordenador y han configurado la industria del software y de la informática tal y como la conocemos hoy en día (Vidal, 2004, pág. 47).

1.2 Definición de Software Libre y las cuatro libertades

El Software Libre es un asunto de libertad, no de precio. Para entender este concepto, debemos pensar en la acepción de libre como en <<libertad de expresión>> y no como en <<barra libre de cerveza>> (Stallman, Software libre para una sociedad libre, 2004, pág. 59).

Esta es tal vez la definición más conocida y extendida del Software Libre. Y claramente hace una referencia a las libertades que tienen los usuarios de ejecutar, copiar distribuir, estudiar, y mejorar el software, que puede ser resumida en las siguientes libertades:

> **Libertad 0**: la libertad para ejecutar el programa sea cual sea nuestro propósito.
>
> **Libertad 1**: la libertad para estudiar el funcionamiento del programa y adaptarlo a tus necesidades -el acceso al código fuente es indispensable para esto.
>
> **Libertad 2**: la libertad para redistribuir copias y ayudar así a tu vecino.
>
> **Libertad 3**: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad -el acceso al código fuente es indispensable para esto. (Stallman, Software libre para una sociedad libre, 2004, págs. 59-60)

Las libertades anteriormente enunciadas comportan unas ventajas para los usuarios de Software Libre. Por ejemplo, las libertades 1 y 3 requieren de acceso al código fuente porque para realizar las actividades de estudiar y modificar el software es necesario que el usuario tenga acceso al código fuente, pues sin él es poco probable que estas libertades sean efectivas.

En este sentido la primera ventaja que comporta el Software Libre para sus usuarios frente a los usuarios de software propietario, es que los primeros pueden modificarlo según las necesidades que ellos tengan o que una comunidad requiera, mientras que para los segundos las modificaciones están prohibidas.

La segunda ventaja se encuentra en la seguridad del uso del software, ya que al disponer los usuarios del código fuente del programa pueden realizar una revisión del mismo, de este modo se pueden detectar los posibles fallos de seguridad. Por ejemplo, Linux hace posible que sus usuarios no necesiten de un antivirus, pues las actualizaciones constantes que se realizan se encargan de proteger el sistema, esto no significa que el sistema sea completamente inmune de ser infectado por un virus, pero sí que los problemas sean resueltos más rápidamente lo que permite que el sistema tenga mayor seguridad.

La tercera ventaja que presenta el Software Libre para sus usuarios frente al software de propietario, es que a los primeros al disponer libremente del código fuente, éste es revisado y modificado por múltiples usuarios, lo que permite que se encuentre en un proceso constante de mejoramiento de su calidad. Esto, sumado a la fácil portabilidad que presenta el Software Libre (GNU/Linux), permite que sea más sencillo adaptar los programas para su funcionamiento en diferentes arquitecturas de ordenadores. Al mejoramiento constante debemos agregar que uno de los grandes objetivos del Software Libre es compartir la información trabajando de manera cooperativa, ya que este es el principal modelo sobre el cual la humanidad ha innovado y avanzado.

Lo anterior parte de la premisa de que el conocimiento le pertenece a la humanidad, sin distinción de clase, raza, sexo, etc., razón por la cual sus usuarios cumplen un papel fundamental a la hora de decidir hacia dónde deben evolucionar los programas que se desarrollan, esto lo hacen a través de corregir o señalar los errores que tienen los programas, proporcionando nueva funcionalidad a dichos programas o desarrollando software.

La cuarta ventaja es que, al no tener restricciones en la distribución del software, el costo es muy bajo, e incluso puede llegar a cero. Además, es necesario señalar que el costo de un software propietario implica no solo la adquisición de la licencia, sino también su mantenimiento, la operación y los ajustes que se le deban realizar. Por el contrario, cualquier persona con una computadora y conexión a internet puede utilizar Software Libre. Esto para la mayoría de usuarios individuales es una gran ventaja, pues estas libertades garantizan que no se vean agobiados por los costos del software.

Es importante aclarar en este punto que generalmente esta ventaja del Software Libre se confunde con el freeware, pues con ello muchos piensan que son lo mismo, es decir, que el Software Libre siempre es gratis o que el software gratuito es de código abierto. Para comprender las diferencias que existen entre estos dos tipos de software debemos tener en cuenta que según la Free Software Fundation, el Software Libre es aquel que los usuarios pueden ejecutar, copiar, distribuir, estudiar, modificar y mejorar; de manera más precisa el Software Libre se refiere a las cuatro libertades que enunciamos anteriormente.

Por el contrario, el freeware o software gratuito define un tipo de software no libre que se distribuye sin costo para su uso y por tiempo ilimitado, es decir, que se puede utilizar sin pagar ninguna cantidad de dinero, pero que tiene restricciones en cuanto a su uso y a las libertades que el usuario puede tener respecto al código fuente.

Con esto podemos decir que la denominación de Software Libre es aplicable a cualquier programa cuyos usuarios gocen de las libertades anteriormente enunciadas, es decir, las libertades de redistribuir copias, de forma gratuita a cualquier persona y en cualquier lugar del mundo.

Gozar efectivamente de estas libertades significa no tener que pedir permiso para modificar, distribuir, estudiar -estas libertades depende de la legalidad vigente, es decir, mediante una licencia determinada, como veremos más adelante- ni pagar una contraprestación económica por ello.

Sin embargo, son aceptables ciertas reglas sobre la distribución del Software Libre, siempre y cuando estas no entren en conflicto con las libertades centrales. Por ejemplo, si observamos una regla como el Copyleft que implica que cuando se redistribuya un programa éste no puede tener restricciones para denegar a otros las libertades centrales, y, como veremos más adelante, esta regla no entra en conflicto con las libertades, por el contrario, las protege.

En este punto es necesario aclarar qué entendemos por software propietario, pues siempre utilizamos esta referencia para contraponerlo al Software Libre. En efecto, el software propietario tiene como característica principal el no permitir que sus usuarios tengan acceso al código fuente de sus programas, además de limitar o prohibir modificaciones y su redistribución.

En este sentido Culebro Juárez, Gómez Herrera y Torres Sánchez han señalado en su libro *Software Libre vs. Software Propietario: ventajas y desventajas* que el software no libre:

> Se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o que su código fuente no está disponible o el acceso se encuentra restringido. (Juárez, Gómez Herrera, & Torres Sánchez, 2006, pág. 4)

Por lo anterior, podríamos concluir señalando que el software propietario es cualquier programa informático con condiciones de uso y distribución que se encuentren en contravía de las

libertades centrales -anteriormente señaladas- que brinda el Software Libre tanto para sus usuarios como para sus programadores.

1.3 Licencias y tipos de licencias

En esta sección analizaremos *grosso modo* los principales aspectos legales relacionados con el Software Libre, pues en la actualidad la creación de conocimiento, la información y sus derivados se han convertido en uno de los ejes centrales del desarrollo económico y social, como bien señala el sociólogo Manuel Castells: "information generation, processing, and transmission become the fundamental sources of productivity and power because of new technological conditions emerging in this historical period" (Castells, 2010, pág. 21). El economista italiano Andrea Fumagalli, señala al respecto lo siguiente:

> Si en el capitalismo industrial, el control de las máquinas era una condición propedéutica para la acumulación, que tendía a incorporar el saber técnico, en el capitalismo cognitivo la acumulación se funda en la apropiabilidad y en el control del saber y el conocimiento social. En otras palabras, el conocimiento social -es decir, el *general intellect*- constituye hoy el eje del proceso de creación de riqueza. (Fumagalli, 2010, pág. 103)

En este contexto, conocer el marco legal que regula las expresiones intelectuales se hace necesario para poder utilizar las diversas herramientas legales que están a nuestra disposición para colectivizar el conocimiento y garantizar los derechos -que vimos en la sección anterior- de los usuarios de Software Libre. Además, al conocer este marco jurídico podemos entender de una mejor manera los derechos y las obligaciones que se adquieren al utilizar una determinada aplicación o paquete de Software Libre.

Contrario al Software Libre, el software propietario utiliza el marco legal y jurídico para garantizar que el fabricante pueda impedir a los usuarios el acceso al código fuente de sus programas y con ello coartar los derechos de modificar y copiar.

Ahora bien, en este punto y por mor de nuestro propósito, es necesario que miremos rápidamente el concepto de derechos de autor[4], ya que juega un papel importante en la protección de los programas informáticos y en las creaciones intelectuales.

Según Delia Lipszyc:

> Se entiende por derecho de autor, en su sentido subjetivo, al conjunto de facultades del que goza un autor en relación con la obra que tiene originalidad o individualidad suficiente y que se encuentra comprendida en el ámbito de protección dispensada. En su sentido objetivo, se refiere a la denominación que recibe la materia que trata estos asuntos (Lipszyc, 2003)

Como podemos observar, los derechos de autor se encargan de regular los derechos que se reconocen -de manera exclusiva- al creador de una obra. Las obras protegidas por los derechos de autor pueden ser libros u obras de arte y modernamente se han incluido los programas de computadoras. Bajo esta protección se prohíbe reproducir sin permiso estas obras de manera total o parcial, con o sin modificaciones. Y esta protección entra en vigor cuando la obra es publicada.

Es importante señalar que estos derechos que protegen los derechos de autor se derivan de los derechos morales que tienen que ver con el reconocimiento del autor y la divulgación de su obra, y de los derechos patrimoniales que se relacionan con la reproducción, distribución, comunicación y modificación de la obra.

[4] Para ampliar el concepto de derechos de autor el lector se puede remitir al libro *Derechos de ¿autor? El debate de hoy* de Lillian Álvarez Navarrete.

1.3.1 Licencias robustas, permisivas y dual. Una licencia de software es la autorización o permiso que un autor, en cualquier forma contractual, otorga a los usuarios de un programa informático, para que estos lo puedan utilizar bajo una forma determinada y en conformidad a unas condiciones establecidas. Es decir, que una licencia:

> Determina cómo el autor cede normalmente de manera no-exclusiva- parte de sus derechos al usuario (copia, modificación, distribución, etc.), determinado en qué condiciones el usuario puede utilizar el programa informático y detallando el ámbito de los derechos y obligaciones asociados. A este contrato se le denomina licencia de software (Hernández, 2005, pág. 69)

A través de la licencia se autorizan las condiciones de uso, modificación o redistribución para los usuarios y los límites de estos. Es importante señalar que las licencias pueden o no tener algún valor monetario. Además, en la licencia se puede estipular algún plazo de duración y las demás cláusulas que el titular del derecho considere pertinentes.

Si ponemos la discusión del software privativo y el Software Libre en el ámbito de la normatividad jurídica que rige los derechos de autor encontraríamos que sus situaciones no son muy diferentes, pues ambos tipos de software se distribuyen bajo licencia. En este claroscuro que podría existir entre un tipo de software y otro, lo que nos ayudaría a hacer distinciones es el tipo de licencia bajo el cual se encuentra licenciado el software, pues en el caso del Software Libre la licencia no restringe el uso, la redistribución y la modificación.

Por el contrario, y como señala el ingeniero Jordi Mas Hernández: “En el mundo del software propietario prácticamente cada fabricante de software ha creado su propia licencia adecuada al software en cuestión y el modelo de negocio del fabricante” (Hernández, 2005, pág. 69). En este sentido, las licencias que usa el software propietario tienen como propósito impedir que el usuario pueda estudiar, copiar, modificar o redistribuir el software.

Ahora bien, en el universo del Software Libre existe una gran variedad de licencias. A nivel general se pueden distinguir tres modelos principales de licenciamiento de software; estos modelos se diferencian entre sí a través de cómo los propietarios de los derechos los ceden a los usuarios:

1. *Licencias Robustas*: en este grupo la licencia por antonomasia es la General Public License (GLP por sus siglas en inglés). Su autoría corresponde a la Free Software Fundation y es la más conocida del Software Libre, incluso uno de sus proyectos bandera Linux se encuentra licenciado bajo la GLP. La FSF creó esta licencia para todo el software que generara, ya que con ella podían cubrir con gran detalle las libertades del Software Libre. Es decir, la GLP permite la redistribución binaria y del código fuente. También está permitido que se realicen modificaciones en los programas sin restricciones. Además, cuando un programa es publicado bajo esta licencia nunca puede ser cambiado.
2. *Licencias permisivas*: La licencia más conocida de este tipo es la BDS. Las licencias permisivas son aquellas que permiten ceder a los usuarios el uso de los programas a través de las condiciones que definen el Software Libre. En este sentido las licencias permisivas no restringen el uso, la modificación y la redistribución. El problema de estas licencias es que alguien puede utilizar un programa libre para hacer aplicaciones u otro tipo de productos propietario, sin encontrarse obligado a compartir las modificaciones y las mejoras realizadas.
3. *Licenciamiento Dual*: Este tipo de licencias se basan en que el autor cede los derechos de su obra bajo dos licencias diferentes. Es decir, dependiendo el uso que el usuario vaya a realizar de su producto se otorgan diferentes tipos de libertades y establecen obligaciones. Por ejemplo, una empresa puede ofrecer una versión libre

de un programa y a la vez una versión del mismo programa con condiciones más óptimas para usuarios fuera del modelo de Software Libre.

1.4 Richard Stallman, el proyecto GNU y el movimiento del Software Libre

Richard Stallman es un hacker estadounidense y fundador del movimiento por el Software Libre en el mundo. Nació el 16 de marzo de 1953 en Manhattan, New York. Estudió física en Harvard y a medidos de los años 80 ingresó a la plana mayor de los programadores de software del Laboratorio de Inteligencia Artificial del MIT, puesto al que renunciaría posteriormente para dedicarse de tiempo completo al desarrollo de Software Libre.

Algunos de sus logros más importantes como programador de Software Libre incluyen la realización del editor de texto GNU Emacs. Este editor es altamente extensible y configurable, es distribuido bajo licencia libre y actualmente lo mantiene la Free Software Fundation. También creó el compilador GCC para lenguajes de programación como C, C++, Objetive C y Fortran; este compilador puede recibir un programa fuente escrito en cualquiera de estos lenguajes y con ello genera un código ejecutable binario en el lenguaje de la máquina que se requiera. Estos y algunos otros proyectos fueron desarrollados por Richard Stallman y se encuentran licenciados en GLP y son parte del proyecto GNU.

Ahora bien, a mediados de los años 80 y mientras Richard Stallman trabajaba en el IA del MIT -como señalamos anteriormente- decidió abandonar su trabajo para dedicarse por completo al desarrollo de Software Libre a través del Proyecto GNU. Esta decisión de abandonar el IA del MIT se produce, porque a su ingreso a la comunidad de desarrollo de software él se había acostumbrado a colaborar con otros desarrolladores intercambiando el código fuente de los programas, ya que cuando se compartía con otros, se lograba mejorar y ampliar el espectro de uso de los programas, obteniendo mejores resultados y optimizando el código.

Esta práctica de compartir el código fuente era muy común en los años 70 como el propio Stallman señala:

> Cuando entré a trabajar en el Laboratorio de Inteligencia Artificial (AI Lab) del MIT en 1971, pasé a formar parte de una comunidad que compartía software y llevaba haciéndolo durante años. El acto de compartir software no se circunscribe a nuestra comunidad en particular: es tan antiguo como los propios ordenadores, lo mismo que compartir recetas es tan viejo como la cocina. Simplemente, nosotros lo hacíamos en mayor medida. (Stallman, Software libre para una sociedad libre, 2004, pág. 19)

Con el paso del tiempo esta cultura de compartir que había forjado las comunidades de desarrolladores de software fue cambiando, en gran medida porque el desarrollo de software se convirtió en un gran negocio, lo que produjo que el código fuente de los programas pasara de ser un elemento que libremente los programadores compartían e intercambiaban a ser considerado un secreto estratégico para empresas.

¿Cómo se dio cuenta Stallman que la comunidad del software estaba cambiando? La historia empieza cuando la compañía Xerox le regala una impresora láser al departamento en el cual Stallman trabajaba. Esta impresora se suponía era un prototipo de vanguardia que dependía de los datos distribuidos en una red para convertirlos en documentos.

Pero esta impresora se atascaba con mucha frecuencia impidiendo que las personas que trabajaban en el laboratorio pudieran recoger los documentos que enviaban a imprimir. Así narra este evento Sam Williams:

> An hour after sending off a 50-page file to the office laser printer, Stallman, 27, broke off a productive work session to retrieve his documents. Upon arrival he found only four pages in the printer's tray. To make matters even more frustrating, the four pages belonged to another user, meaning that Stallman's print job and the

> unfinished portion of somebody else's print job were still trapped somewhere within the electrical plumbing of the lab's computer network. (Williams, 2010, pág. 1)

La frustración que producía que la impresora nueva se atascara cuando se enviaban los documentos hizo que Stallman intentara solucionar el problema. Al ser un prominente programador buscó el software de la nueva impresora láser de Xerox y ahí se encontró con un problema mayor a los atascamientos de la máquina, pues ésta no tenía ningún software que él o sus compañeros en el laboratorio pudieran leer, ya que, como hemos señalado anteriormente, las compañías tenían el hábito de publicar el código fuente de los programas en archivos legibles y en esta ocasión Xerox los había provisto archivos de software de forma binaria.

Al intentar Stallman obtener el código fuente del programa a través de Robert Sproull en una visita al campus de la Universidad Carnegie Mellon éste se tiene que negar a compartir el código, ya que ha firmado un acuerdo para no revelar el código. En otras palabras, existía un acuerdo contractual entre la Compañía Xerox y Sproull para que éste o cualquier otro que tuviera el código no pudiera compartirlo. Williams señala al respecto:

> In talking to audiences, Stallman has made repeated reference to the incident, noting that the man's unwillingness to hand over the source code stemmed from a nondisclosure agreement, a contractual agreement between him and the Xerox Corporation giving the signatory access to the software source code in exchange for a promise of secrecy. Now a standard item of business in the software industry, the nondisclosure agreement, or NDA, was a novel development at the time, a re ection of both the commercial value of the laser printer to Xerox and the information needed to run it. […] For Stallman, the realization that Xerox had compelled a fellow programmer to participate in this newfangled system of compeled secrecy took a while to sink in. In the first moment, he could only see the refusal in a personal context. (Williams, 2010, pág. 8)

El caso de la impresora sirvió para que Stallman se diera cuenta de lo que estaba pasando en la industria del software y los cambios a los que rápidamente se estaba viendo avocada. Por ello

decide renunciar a su puesto en el MIT. El 27 de septiembre de 1983 Stallman anuncia el comienzo del proyecto GNU en el grupo de noticias net.unix-wizards.

Al abandonar el MIT Stallman tiene el propósito de construir un sistema de software completo totalmente libre. Este proyecto lo llamó GNU -un acrónimo sarcástico para señalar que GNU no era Unix-. Desde el inicio este proyecto contó con algunos elementos de software ya disponibles. Razón por la cual Stallman se dedicó a escribir el compilador de C y el editor (Emacs), estos aún se encuentran en uso y son muy populares entre los usuarios de software libre.

Con la construcción de este proyecto Stallman se preocupó en gran medida por las libertades que tendrían tanto los programadores que hicieran parte del proyecto, como por las libertades que tendrían los usuarios de los programas. Para defender que tanto los programadores como los usuarios tuvieran los derechos de redistribuir, copiar, estudiar y modificar el código fuente Stallman escribe la licencia GLP -de la que ya se ha hablado- con el propósito de garantizar que los programas fueran libre en el sentido en que hemos señalado anteriormente.

Pero Stallman no solamente inició el proyecto GNU, sino que fundo también la Free Software Fundation, con el fin de obtener fondos para el desarrollo y la protección del software libre. Los fundamentos éticos los sentó en sus textos: *El manifiesto GNU* y *Por qué el software libre no debe tener propietarios*.

Seis años después de haber sido lanzado públicamente el proyecto GNU se había desarrollado gran parte del sistema, pero aún no se tenía una de las piezas principales, a saber, el núcleo del sistema o *kernel* que es la parte del sistema operativo que se relaciona con el hardware, además de permitir que las aplicaciones compartan los recursos.

Aún con la falta de este elemento primordial para tener un sistema operativo completo el software desarrollado en GNU era muy popular en las comunidades de usuarios de las distintas variantes de Unix.

1.5 GNU/Linux

1.5.1 Los antepasados directos de Linux: Multics, Unix y Minix. En los años 60 la Oficina Técnica de Procesamiento de la Información (IPTO - Information Processing Techniques Office) de la Agencia de Proyectos de Investigación Avanzados (ARPA- Advanced Research Projects Agency), junto a *Bell Telephone Laboratories* de AT&T y *General Electrics* se unieron para desarrollar un sistema operativo de tiempo compartido, el desarrollo de este proyecto derivo en el sistema operativo conocido como MULTICS (Multiplexed Information and Computing Service). Multics presento cuatro características relevantes que constituyeron un gran avance para su tiempo, además de marcar el desarrollo de futuros sistemas operativos.

La primera es que tenía una estructura modular, es decir, que el software, el hardware y el sistema pueden crecer añadiendo más recursos como memoria principal, almacenamiento de disco, etc. La segunda, es la lista de control de archivo, que es un mecanismo que implementa control de acceso a un recurso del sistema enumerando las entidades del sistema que tienen permiso para acceder al recurso. La tercera es que el sistema contaba con varios mecanismos que permitían analizar el rendimiento del sistema operativo y con ello poder optimizar su rendimiento. Finalmente, Multics contaba con una pantalla de rayos catódicos lo cual permitió por primera vez que los usuarios vieran lo que escribían en una pantalla.

Pese a estos grandes avances realizados por los desarrolladores de Multics, este contaba con un par de inconvenientes los cuales harían que en 1969 los Laboratorios Bell decidieran dejar el proyecto -pero sin abandonar la idea de crear un sistema operativo de tiempo compartido-, el primer inconveniente es que no lograba ser un sistema totalmente funcional y el segundo inconveniente es que resultaba muy costoso para poder ser utilizado.

Ese mismo año Kenneth Thompson quien era investigador en los Laboratorios Bell de AT&T junto con Dennis Ritchie desarrollan un sistema operativo multitarea que podía soportar dos usuarios simultáneamente, este sistema operativo incluía un sistema de archivos y un intérprete de comandos, sus creadores lo nombraron UNICS (Uniplexed Information and Computing Service), el cual era una versión simplificada de Multics que conservaba un gran número de los avances y desarrollos que este sistema operativo había logrado. A diferencia de su antecesor Unics empieza a rendir frutos rápidamente y en 1970 pasó a denominarse UNIX y el 3 de noviembre 1971 la Versión 1 es ejecutada por primera vez en un DEC PDP-7.

En 1973 Thompson y Ritchie reescriben el núcleo del sistema -que en principio había sido escrito en lenguaje ensamblador, el cual es un lenguaje de programación de bajo nivel- utilizando lenguaje C de alto nivel creado por el propio Ritchie, este cambio hizo más fácil el mantenimiento del sistema y además le brindo portabilidad al sistema a un amplio rango de arquitecturas.

Lo anterior aunado a la decisión de los Laboratorios Bell de mantener público el código fuente y de permitir que las universidades adquirieran el sistema únicamente pagando el costo del medio magnético, los manuales y el transporte. Estas ventajas tenían una condición y es que los avances que produjeran las universidades relacionados con el sistema fuesen reportados a los Laboratorios Bell.

Para 1975 muchas universidades utilizaban Unix para investigación y la enseñanza práctica. La universidad de California, Berkeley fue la más destacada, pues en esta se desarrolló gran parte de las innovaciones más importante de este sistema como fue la inclusión del C shell (intérprete de comandos), el Editor Vi (editor de texto) y el compilador Pascal.

Esto producto en buena medida a que Kenneth Thompson había decidido tomarse un año sabático de los Laboratorios Bell y enseñar en dicha universidad y junto a los estudiantes recién graduados William Joy y Chuck Haley empezaron a escribir software para el sistema bajo el nombre de Berkeley Software Distribution (BDS).

En 1979 los trabajos que se estaban desarrollando en la Universidad de California convencieron a la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA) para financiar el desarrollo de un sistema Unix estándar para uso del gobierno; 4BSD fue el resultado de ese proyecto.

Uno de los objetivos principales de este proyecto era proporcionar soporte para los protocolos de las redes de Internet de DARPA (TCP/IP) y en la versión 4.2BSD hizo posible la comunicación entre diversas instalaciones de red, incluidas las redes de área local (como Ethernet y Token Ring) y redes de área amplia (como NSFNET). Estos protocolos permitieron que Internet creciera de 60 redes conectadas en 1984 a más de 8000 redes y aproximadamente 10 millones de usuarios en 1993.

Ahora bien, en 1981 AT&T decide poner fin a la distribución de fuentes con Unix y distribuir el sistema bajo una licencia comercial, esta decisión genero una gran molestia en la comunidad académica que utilizaba Unix para la enseñanza práctica de sistemas operativos.

De tal modo que para 1983 AT&T lanza Unix System V que es la primera versión que se distribuye de manera comercial y que además estaba diseñada para ser compatible con las siguientes versiones; ya que uno de los grandes problemas que tenía este sistema operativo es que cada nueva versión era incompatible con la versión anterior, es decir, que en cada nueva versión se realizaban muchos cambios. Por ende, los programas escritos para una versión particular no podían ser ejecutados en la siguiente sin tener que ser compilados e incluso reescritos nuevamente.

Ese mismo año y como habíamos señalado anteriormente la Universidad de California lanza su Unix BSD versión 4.2 que ya no era compatible con la versión de System V, aunque desde 1974 ya se empezaban a desarrollar nuevos sistemas operativos derivados de la versión original de Unix. Es importante señalar que después de esto tanto BSD y System V se vieron como las mejores alternativas para el desarrollo de un nuevo Unix, así que los desarrolladores elegían entre alguno de estos para este propósito.

Cabe destacar además que cada uno de estos sistemas tuvo varios tipos de sistemas operativos derivados de ellos y algunos siguen vigentes hasta la fecha, por ejemplo, el sistema SunOS -que en la actualidad se conoce como Solaris- desarrollado por Bill Joy utilizando BSD, así como los sistemas libres NetBSD, OpenBSD y FreeBSD; mientras que sistemas como Xenix, HP-UX e incluso Linux han seguido la variante System V.

Hoy en día la mayoría de las variantes de Unix no son 100% BSD o System V, pues éstas a través del tiempo han tendido a unificarse para generar mejores sistemas. Sin embargo, esas variantes tienen en mayor o menor medida preponderancia de una de estas dos variantes.

Como hemos podido observar hasta el momento, en la década de los 80 Unix era un sistema muy popular por su utilidad para desarrollar distintas tareas, sin embargo su mayor dificultad era

la incompatibilidad que existía entre las diferentes versiones, por ende como señalamos anteriormente los programas eran escritos para cada versión en específico, esto impedía que se desarrollaran programas de propósito general para Unix, así es que a mediados de la década el grupo /usr/grup -una organización de usuarios de Unix- que tenía como propósito la compatibilidad de las aplicaciones de los diferentes sistemas.

Así que en 1986 se publica un estándar para los sistemas denominado Posix (Interfaz de Sistema Operativo Portable) con el que se pretendía poner fin a la portabilidad de los programas entre los sistemas Unix, aunque este tuvo muy buena acogida Posix no pudo cumplir a cabalidad su propósito. Pero Posix no fue el único y en 1984 se fundó la compañía X/Open Ltd., una compañía formada por varios vendedores de Unix, en la cual también se desarrolló un estándar Unix.

Los esfuerzos de unificación de Unix fueron posteriormente recogidos por la IEEE (Instituto de Ingenierías Eléctrica y Electrónica) que no representa un sólo estándar, sino un grupo de estándares que tienen como objetivo garantizar la portabilidad de aplicaciones a los sistemas abiertos.

Aunque Unix era el sistema más popular hasta finales de la década de los 80, las dificultades que había tenido respecto a la compatibilidad de las versiones y la decisión de AT&T de poner fin a las distribuciones, generó una pérdida en el entusiasmo que en un principio habían tenido sus usuarios y que lo habían convertido en un gran éxito. De este modo el tiempo estaba maduro para que surgiera un nuevo sistema.

Como señalamos anteriormente, muchos profesores universitarios lamentaron que ya no fuera posible utilizar Unix en la enseñanza práctica y teórica de los sistemas operativos, uno de ellos fue el profesor Andrew Tanenbaum de la Vrije Universiteit de Ámsterdam en Países Bajos.

El profesor Tanenbaum, al igual que Thompson, se inspiró en Multics con el propósito de escribir un nuevo sistema operativo, aunque Tanenbaum terminó escribiendo un nuevo sistema más pequeño inspirado en Unix que decidió llamar Minix, que significa Mini-Unix. Aquí es importante señalar que este sistema es originalmente desarrollado como una herramienta de enseñanza para su libro *Operating Systems Desing and Implementation.*

Como Minix no contenía ningún código de AT&T, se encuentra fuera de las restricciones que esta empresa había puesto al uso de su software, por lo cual el código fuente de Minix fue puesto a disposición de las universidades para que lo utilizaran en la enseñanza se sistemas operativos tanto de modo práctico como teórico. Además, estaba disponible para cualquier persona que lo quisiera utilizar. De este modo Minix gano rápidamente un gran número de usuarios, pues, era un sistema operativo pequeño el cual tenía a disposición de los usuarios todo el código fuente para que estos lo estudiaran y lo modificaran como quisieran. Al respecto Tanenbaum señala:

> By not using even one line of AT&T code, this system avoided the licensing restrictions, so it could be used for class or individual study. In this manner, readers could dissect a real operating system t osee what is inside, just as biology students dissect frogs. (Tanenbaum & Woodhull, 2006, pág. 16)

Minix es lanzado en 1987 y unos pocos meses después de su lanzamiento había generado un gran interés en todo el mundo -especialmente en las universidades-. Es tal el interés que este nuevo sistema operativo suscitó que al poco tiempo se creó un grupo de noticias en USENET,

comp.os.minix, grupo que llegó a tener unos 40000 usuarios que leían lo que estaba sucediendo con Minix y que además aportaban código para el crecimiento de este sistema:

> Withim a few months of its appearance, Minix become a bit of a cult ítem, with its own USENET (now Google) newsgroup, *comp.os.minix*, and over 40,000 users. Numerous users contributed commands and other user programs, so MINIX quickly became a collective undertaking by large numbers of users over the Internet. (Tanenbaum, 2008, págs. 719-720)

Aquí es importante señalar que uno de los propósitos principales del profesor Tanenbaum con su sistema operativo era que éste se utilizara con fines académicos y educativos, por lo cual, aunque muchas personas hicieron aportes a través del grupo en USENET de lo que creían debía ser mejorado u agregado al sistema, la mayoría de estos aportes eran rechazados puesto que para que este sistema operativo pudiera cumplir su objetivo pedagógico era necesario que se mantuviera lo más pequeño posible, de tal modo que pudiera ser comprendido por una sola persona.

Además, hacer más grande el software también implicaba intrínsecamente que para su estudio se tuviera un hardware lo suficientemente potente para ejecutar el sistema, lo cual la mayoría de los estudiantes no podían costear.

Por lo anterior, Minix se escribió al igual que Unix en lenguaje de programación C y en 1987 en su lanzamiento este sistema consistía en 11800 líneas de C y 800 líneas de código ensamblador y fue solo para la nueva PC de IBM como lo señala el propio Tanenbaum en un artículo que escribió para la revista *Communications of the ACM*: “My idea was to write the system, called Mini-uNIX, or MINIX, for the new IBM PC, which was cheap (starting at $1565) a student could own one” (Tanenbaum, 2016, pág. 71), pero no tardaron mucho tiempo en que el sistema fuese adaptado para poder utilizarse en otro tipo de computadoras.

Finalmente, respecto a Minix es importante señalar que también fue diseñado para que fuera compatible con la versión VII de Unix, la cual Tanenbaum consideraba sencilla y elegante. Además de ser la versión de sistema operativo que más se utilizaba para la enseñanza en los departamentos de ciencias de la computación en las universidades. Como pudimos observar a través del desarrollo de Minix este siempre estuvo muy ligado con la educación, ya que su creador quería suplir una necesidad que existía en las aulas de enseñanza de sistemas operativos.

Nuestro interés en presentar las características y la forma como fueron desarrollados estos sistemas es que estos van a definir la filosofía básica del diseño de Linux, ya que éste será creado siguiendo las premisas que empezaban a establecerse a través de Multics, Unix y Minix y que nos ponen en el momento de la génesis de Linux.

1.5.2 El núcleo GNU/Linux. La historia de Linux se remonta a los orígenes del sistema operativo Unix, del cual ha evolucionado. Linux fue creado originalmente por Linus Torvalds en la Universidad de Helsinki en Finlandia. Inicialmente sólo fue un proyecto de aficionado, el cual se inspiraba en Minix, un pequeño Unix desarrollado por Tanenbaum.

Linus Torvalds creó Linux en 1991, cuando aún era estudiante de la facultad de Ciencias de la Computación de la Universidad de Helsinki en Finlandia, donde había estado utilizando Minix, un sistema similar a Unix que para esa época ya no era libre y comenzó a escribir su propio núcleo.

Este fue desarrollado mediante controladores de dispositivos y acceso de disco duro, en el mes de septiembre Linus Torvalds ya tenía un diseño básico del *kernel* que llamó versión 0.01. Este *kernel* recibo el nombre de Linux y se convino con el sistema GNU para producir un sistema operativo completamente libre.

En octubre de 1991, Torvalds envió un mensaje al grupo de noticias conocido como comp.os.minix para anunciar el lanzamiento de la versión 0.02, una versión básica que aún se necesita para operar Minix, pero que, sin embargo, suscitó bastante interés en la comunidad de programadores. Razón por la cual el núcleo fue rápidamente mejorado por Torvalds y un número creciente de voluntarios que se comunicaban a través de internet.

Para diciembre de ese mismo año tenían un sistema Linux funcional, independiente de Unix que fue lanzado como la versión 0.11. Para el mes de enero de 1992, Linux ya tenía una nueva versión llamada 0.12. Torvalds liberó la versión 0.11 bajo una licencia gratuita de su propia invención, pero la versión 0.12 fue liberada bajo la Licencia Pública General de GNU.

La siguiente versión se conocería como 0.95, este nombre le fue dado con la intención de reflejar que Linux se estaba convirtiendo en un sistema completo (Alfaro, 2000, pág. 7). Linux continuó mejorando a través de los años y comenzó a ser utilizado en aplicaciones a gran escala, como en algoritmos web, creación de redes, y la producción de base de datos lo que demostraba que Linux estaba listo para su uso en masa.

En diciembre de 1993 el núcleo se encontraba en la versión 0.99, en una aproximación asintótica del 1.0, número que generalmente se utiliza para dar cuenta de una versión estable. La versión 1.0 fue lanzada en 1994, ya que Torvalds consideraba que todos los componentes del núcleo estaban totalmente maduros, presentándose la versión 1.0. Esta versión fue la primera en estar disponible para descargar en internet.

En la versión 2.2, que se realizó una importante actualización del núcleo de Linux, fue lanzada oficialmente en enero de 1999. Para el año 2000, la mayoría de las compañías de computadoras se hicieron compatibles de una u otra forma con Linux, reconociendo la importancia

del trabajo realizado en la década de los 90. Además, veían en Linux la posibilidad de unificar el mundo fracturado de Unix.

La próxima versión sería la 2.4 lanzada en enero de 2001. Esta versión proporcionaba compatibilidad (entre otras mejoras) con la próxima generación de procesadores Itanium de 64 bits de Intel.

La versión del núcleo 4.0 fue lanzada en abril del 2015. Su código fuente creció varios cientos de miles de líneas –en comparación con la versión 3.19-, gracias a las contribuciones de miles de desarrolladores de software que trabajaron en él, además de empresas comerciales que incluyeron programadores pagos de las cuales se pueden destacar empresas como Red Hat, Intel, Samsung, Broadcom, Texas Instruments, IBM, Novell, Qualcomm, Nokia, Oracle, Google, AMD, e incluso Microsoft.

Actualmente, el núcleo se encuentra en la versión 4.18. Hoy Linux es un clónico de Unix completo. Mucho software libre ha sido ya portado a Linux, y están empezando a aparecer aplicaciones comerciales.

1.5.3 ¿Qué es Linux? Linux es un Sistema Operativo, una versión de Unix, independiente y de libre distribución para computadoras personales (PC), servidores y estaciones de trabajo. Linux es uno de los paradigmas más prominentes del software libre y del desarrollo del código abierto, pues su código fuente se encuentra disponible y es de acceso público, para cualquier persona. Puede ser usado, estudiado, redistribuido y modificado.

Como sistema operativo, Linux es un conjunto de programas que le permiten interactuar con el computador y ejecutar programas. Un sistema operativo consiste en varios programas fundamentales que necesita el computador para que se pueda comunicar y recibir instrucciones de

los usuarios; por ejemplo, leer y escribir datos en el disco duro, controlar el uso de la memoria y ejecutar programas.

La parte más importante de un sistema operativo es el núcleo. En el sistema GNU/Linux, Linux es el núcleo. El resto del sistema puede consistir en otros programas, muchos de los cuales fueron escritos por o para el proyecto GNU. Dado que el núcleo de Linux en sí mismo no forma un sistema operativo de trabajo, preferimos utilizar el término GNU/Linux para referirnos al sistema que muchas personas llaman de manera informal “Linux”.

Desde el principio, Linux se diseñó para que fuera un sistema multi-usuario multi-tarea. Estos hechos son suficientes para diferenciar a Linux de otros sistemas operativos más conocidos. Sin embargo, Linux es muy diferente de los otros sistemas operativos, pues, a diferencia de ellos, nadie es dueño de Linux. Gran parte de su desarrollo lo realizan voluntarios no remunerados. Desarrollo de lo que más tarde sería GNU/Linux el cual inició en 1984, cuando la Free Software Fundation comenzó el desarrollo de un sistema operativo libre de tipo Unix, llamado GNU.

El proyecto GNU ha desarrollado un conjunto de herramientas de software libre para su uso, estas herramientas permiten a los usuarios realizar tareas que van desde copiar o eliminar ficheros del sistema hasta escribir y compilar programas de edición sofisticada en una variedad de formatos de documentos.

Aunque hay muchos grupos e individuos que han contribuido al desarrollo de Linux, el mayor contribuyente es la Free Software Fundation, la cual creó no sólo la mayor parte de las herramientas que se utilizan en GNU/Linux sino su filosofía y la comunidad que hizo posible Linux.

2. Rizoma y software libre

No se tiene un sistema, tan sólo líneas
y movimientos
Deleuze &Guattari

En al capítulo anterior hicimos un recorrido por la génesis y el desarrollo del software libre, ya que la creación de GNU/Linux, su perfeccionamiento y popularización en el mundo de la informática en particular y de la tecnología en general es un acontecimiento relativamente reciente en la historia de la humanidad, si lo comparamos retrospectivamente con otros desarrollos tecnológicos e informáticos. De tal manera que para este trabajo se hacía necesario revisar su historia.

Lo fundamental de este cuestionamiento por su historia radica en que las ideas básicas que dieron origen a Linux y permitieron su desarrollo se encuentran relacionadas intrínsecamente con el origen del sistema, así como con el entorno en el cual se ha desarrollado.

Por lo anterior, podemos considerar entonces a Linux como el resultado de la evolución permanente de esas ideas básicas. Y en este sentido, el conocimiento de estas ideas y el proceso de evolución que han tenido para dar como resultado a GNU/Linux y sus distintas versiones resultan fundamentales para la compresión de las particularidades que presenta este sistema operativo, así como para conocer sus ventajas y desventajas respecto de otros sistemas.

Pero, para ello es importante que recordemos que Linux sigue evolucionando día a día y que el seguimiento de este proceso de evolución constante se beneficiará del conocimiento de los pasos que hasta ahora ha dado el sistema.

Teniendo claro cómo se ha desarrollado el proyecto del software libre en este capítulo veremos cómo éste podría ser explicado -de manera filosófica- a través del concepto desarrollado por los filósofos franceses Guilles Deleuze y Félix Guattari, a saber, el concepto de rizoma. Para ello, en un primer momento presentaremos el concepto y sus principios y la crítica que presentan los filósofos franceses a lo que denominan el pensamiento arborescente.

En segundo lugar, presentamos un análisis del Software Libre a través del concepto de rizoma. Este análisis lo realizamos por medio de la figura de una sinécdoque, esto porque intentar presentar todas las relaciones y conexiones que se posibilitan exceden la capacidad de este trabajo. Con esta figura de la sinécdoque pretendemos presentar el rizoma Software Libre mediante algunos de sus elementos que nos permitan abrir un mapa que muestre los múltiples puntos de entrada y de salida del rizoma. Entendiendo de mejor manera con esto que es un rizoma y por qué el software libre funciona como un modelo rizomático.

2.1 Rizoma: de la botánica a la filosofía

En botánica un rizoma es un tipo de tallo que crece de manera subterránea, razón por la cual da lugar al surgimiento de brotes y raíces a través de nudos. Así lo señala por ejemplo el botánico español Pío Font Quer en su *Diccionario Botánica*: Metamorfosis caulinar debido a la adaptación a la vida subterránea, o, dicho de manera más simple, tallo subterráneo (Quer, 2000, pág. 950).

Al ser un tallo subterráneo el rizoma puede crecer indefinidamente, es decir, puede avanzar en cualquier dirección y cubrir una gran superficie, por lo cual es común que, con el paso del tiempo, ciertos sectores del rizoma vayan muriendo, pero sin que dejen de producirse nuevos brotes en otras áreas. Como señala Eduard Adolf Strasburger:

> Los rizomas presentan raíces que nacen de ellos y se ramifican de vez en cuando. Como los segmentos más viejos del rizoma mueren con el paso de los años, se produce una multiplicación vegetativa: a partir de una planta rizomatosa se puede formar un **policormo** ampliamente ramificado, que acaba ocupando una gran superficie y, en ciertas circunstancias, llega a ser muy viejo, aunque las partes aéreas de los tallos mueran cada año. (Strasburger, 2004, pág. 160)

Las plantas que cuentan con rizoma pertenecen al grupo de las plantas perennes, es decir, plantas que viven durante cortos periodos. Este adjetivo califica a las plantas que viven entre tres o más años. Si una planta de este tipo vive entre dos o más años se le conoce como bienal; pero si vive un año o menos se le conoce como anual.

Todas las matas, arbustos y árboles son perennes; lo son también las plantas herbáceas con órganos subterráneos persistentes como la grama (que tiene rizoma), la aguaturma y la patata (que tienen tubérculos), la cebolla y la azucena (que tienen bulbo) (Quer, 2000, pág. 819). Al vivir por poco tiempo estas plantas es normal que pierdan sus superficies en el invierno, pero el rizoma se mantiene y permite el almacenamiento de nutrientes a nivel subterráneo.

Ahora bien, es posible dividir un rizoma en distintos fragmentos, asegurándose que en cada uno haya al menos una yema, y luego pueden ser plantados por separado para que se sigan desarrollando. Esto se debe a que el rizoma posibilita la reproducción asexual[5] de la planta. Es decir, que a partir de un fragmento del cuerpo o de una única célula, se puede desarrollar un individuo completo mediante la mitosis.

[5] La reproducción asexual es una forma de reproducción de un ser vivo ya desarrollado en el cual, a partir de una sola célula o grupo de células, se desarrolla por procesos mitóticos un individuo completo, genéticamente idéntico al primero. Este proceso se lleva acabo con un solo progenitor y sin la intervención de los núcleos de las células sexuales o gametos.

Según su crecimiento y ramificación, los rizomas se pueden clasificar en dos clases: *Simpodiales* o *Monopodiales*. Los rizomas simpodiales se caracterizan porque son rizomas en los cuales cada uno de sus porciones surge a través de una yema axilar[6] sucesiva, es decir que cada porción corresponde al desarrollo de yemas axilares sucesivas.

Los rizomas monopodiales, a diferencia de los simpodiales en los cuales la yema terminal impulsa de forma indefinida el crecimiento del rizoma, estos se caracterizan por ser especies invasivas o malezas, un claro ejemplo es el sorgo de Alepo (Sorghum halepense).

Por otro lado, en filosofía el concepto de rizoma fue desarrollado por Félix Guattari y Gilles Deleuze con el fin de denominar un modelo epistemológico en el cual todos los elementos no se encuentren condicionados de influir en los demás. El concepto de rizoma fue desarrollado en *Capitalismo y Esquizofrenia*, obra publicada en dos volúmenes: en 1972 apareció *El Anti-Edipo* y ocho años más tarde en 1980 *Mil Mesetas.*

Así pues, el rizoma como concepto filosófico, desarrollado por Deleuze y Guattari en *Mil Mesetas* nos proporciona un modelo fluido para pensar cómo las ideas y la información emergen, fluyen y evolucionan, desarrollando de este modo una metáfora basada en el desarrollo botánico de una masa de raíces. En tal modo el rizoma como modelo de pensamiento abarca multiplicidades y rechaza la noción de un comienzo y de final claros.

Entonces, el modelo rizomático describe elementos que no se someten a una subordinación jerárquica; por el contrario, en este modelo no importa la posición de los elementos, pues cualquier predicado que se afirme de alguno de los elementos, puede incurrir en la concepción de los demás

[6] Una yema axilar es un brote embrionario localizado en la axila de una hoja. Estos brotes tienen el potencial de formar brotes, los cuales pueden ser vegetativos tallos y ramas o reproductivas flores.

elementos. De esta forma el modelo rizomático tampoco es un modelo genitivo, en otras palabras, no posee un centro.

Por tanto, el modelo rizomático fomenta la noción de mapa, en el cual los conceptos, las creaciones y las interpretaciones cambian desarrollando una progresión orgánica de ideas. Al igual que el rizoma -botánico que vimos anteriormente- este modelo se extiende y se expande de manera imperceptible, no sigue caminos predeterminados ni llega a destinos específicos. En otras palabras, podríamos decir que este modelo no es predecible. El modelo rizomático entonces nos ofrece un enfoque nuevo para entender los cambios que forman y reformulan el mundo que nos rodea, especialmente cuando en el siglo XXI estos cambios suceden a una gran velocidad.

2.1.1 Los principios del Rizoma. En el primer capítulo de *Mil Mesetas: Capitalismo y Esquizofrenia*, titulado "*Rizoma",* los filósofos franceses Deleuze y Guattari presentan una serie de principios que denominan rizomáticos. El primer y segundo principio es el de *Conexión y Heterogeneidad* el cual dice que cualquier punto del rizoma puede ser conectado a cualquier otro, y debe serlo (Deleuze & Guattari, 2002, pág. 13).

El tercer principio es el de la *Multiplicidad*, este principio señala que un sistema rizomático, se compone de múltiples líneas y conexiones, por lo cual en un rizoma no hay puntos o posiciones, como ocurre en una estructura, un árbol, una raíz. En un rizoma sólo hay líneas (Deleuze & Guattari, 2002, pág. 14).

En este sentido, para los autores, la multiplicidad es celebrada por los muchos y la pluralidad, en contraposición con la unidad, lo binario y la totalización, figuras que representan los modelos de pensamiento occidental. Por lo tanto, la función del rizoma es:

> Extirpate roots and foundations, to thwart unites and break dichotomies, and to spread out roots and branches, thereby pluralizing and disseminating, producing differences and multiplicites, making new connections. Rizomatics affirms the principles excluded from Western thought and reintterprets reality as dynamic, heterogenous, and non-dichotomous (Best & Kellner, 1991, pág. 99)

El cuarto principio es el de *Ruptura asignificante*. Con este principio se establece que un rizoma puede ser roto, interrumpido en cualquier parte, pero siempre recomienza según esta o aquella de sus líneas y según otras (Deleuze & Guattari, 2002, pág. 15). Los movimientos del rizoma pueden volver a colocarse alrededor de interrupciones, pues la sección cortada se regenera a sí misma y además sigue creciendo, creando nuevas líneas.

Dicho de otro modo, el rizoma puede ser interrumpido, cortado, segmentado en cualquiera de sus puntos, porque de estas interrupciones se pueden generar nuevas conexiones, las cuales pueden ser rizomáticas o arborescentes.

Por último, tenemos el principio de *Cartografía y Calcomanía*: Un rizoma no responde a ningún modelo estructural o genitivo (Deleuze & Guattari, 2002, pág. 17). Con este principio los autores ponen en contraposición las ideas de mapa y calco:

Los calcos son del orden de la copia, reproducen al infinito, no generan, no estimulan el movimiento, son inertes. Calcos son todas las hojas de un árbol […] El calco siempre debe ser colocado sobre el mapa para realizarse. Por el contrario, el mapa es una interpretación del territorio y, a la vez, sirve para recorrerlo en varios sentidos posibles (Díaz, 2007)

Un mapa a diferencia de un calco tiene múltiples entradas, las cuales se producen y se reproducen a través de la relación de las viejas y las nuevas formas que convergen en diferentes

puntos del rizoma. Por consiguiente, Deleuze y Guattari presentan un ejemplo, el cual citamos a continuación:

> La orquídea desterritorializa al formar una imagen, un calco de avispa; pero la avispa se reterritorializa en esa imagen. No obstante, también la avispa se desterritorializa, deviene una pieza del aparato de reproducción de la orquídea; pero reterritorializa a la orquídea al trasportar el polen. La orquídea y la avispa hacen rizoma, en tanto que heterogéneos. Diríase que la orquídea imita a la avispa cuya imagen reproduce de forma significante (mímesis, mimetismo, señuelo, etc.) pero eso sólo es válido al nivel de los estratos –paralelismo entre dos estratos de tal forma que la organización vegetal de uno imita la organización animal del otro- (Deleuze & Guattari, 2002, pág. 15)

Un rizoma es un mapa y no un calco, por lo tanto, la orquídea hace mapa y no calco de la avispa y esta misma relación también se sostiene a la inversa, con lo cual se produce un movimiento desterritorializante, ya que la relación que sostiene la avispa y la orquídea está en constante movimiento. De este modo, la relación nunca es la misma y cuando pretende serlo, siempre es de forma distinta. Por el hecho de que, la orquídea no produce un calco de la avispa, sino que crea un mapa.

En síntesis, observamos como los significantes son codificados para representar lo dado. Ya que, en esa pretendida imitación de la sociedad, o de la naturaleza, el devenir captura los diferentes códigos: en este caso la orquídea, por ejemplo, adquiere forma de avispa hembra atrayendo a la avispa macho, la cual es seducida por el disfraz de la flor, se posa en la superficie de ella y se impregna de polen, luego esparcirá el polen en otras orquídeas, fecundándolas.

De cierto modo, pareciera que la orquídea imito a la avispa, pero lo que en realidad sucede es que la orquídea deviene momentáneamente avispa, puesto que, entre la avispa y la flor circulan intensidades. No se produce ni imitación ni semejanza, sino el surgimiento de series heterogéneas a partir de un rizoma común, a saber, el rizoma de fecundación.

Ahora bien, los principios del rizoma son resumidos por Deleuze y Guattari del siguiente modo:

> Resumamos los caracteres principales de un rizoma: a diferencia de los árboles o de sus raíces, el rizoma conecta cualquier punto con otro punto cualquiera, cada uno de sus rasgos no remite necesariamente a rasgos de la misma naturaleza; el rizoma pone en juego regímenes de signos muy distintos e incluso estados de no-signos. El rizoma no se deja reducir ni a lo Uno ni a lo Múltiple. No es lo Uno que deviene dos, ni tampoco que devendría directamente tres, cuatro o cinco, etc. No es un múltiple que deriva de lo Uno, o al que lo Uno se añadiría (n+1). No está hecho de unidades, sino de dimensiones, o más bien de direcciones cambiantes. No tiene ni principio ni fin, siempre tiene un medio por el que crece y desborda. Constituye multiplicidades lineales de n dimensiones, sin sujeto ni objeto, distribuibles en un plan de consistencia del que siempre se sustrae lo Uno (n-1). Una multiplicidad de este tipo no vería sus dimensiones sin cambiar su propia naturaleza y metamorfosearse. Contrariamente a una estructura, que se define por un conjunto de puntos y de posiciones, de relaciones binarias entre estos putos y de relaciones biunívocas entre esas posiciones, el rizoma sólo está hecho de líneas: líneas de segmentaridad, de estratificación, como dimensiones, pero también línea de fuga o de desterritorialización como dimensión máxima según la cual, siguiéndola, la multiplicidad se metamorfosea al cambiar de naturaleza. Pero no hay que confundir tales líneas, o lineamientos, con las filiaciones de tipo arborescente, que tan sólo son uniones localizables entre puntos y posiciones. Contrariamente al árbol, el rizoma no es objeto de reproducción: ni reproducción extrema como el árbol-imagen, ni reproducción interna como la estructura-árbol. El rizoma es una antigenealogía, una memoria corta o antimemoria. El rizoma procede por variación, expansión, conquista, captura, inyección. Contrariamente al grafismo, al dibujo o a la fotografía, contrariamente a los calcos, el rizoma está relacionado con un mapa que debe ser producido, construido, siempre desmontable, conectable, alterable, modificable, con múltiples entradas y salidas, con sus líneas de fuga. Lo que hay que volver a colocar sobre los mapas son los calcos, y no a la inversa. Contrariamente a los sistemas centrados (incluso policentrados), de comunicación jerárquica y de uniones preestablecidas, el rizoma es un sistema acentrado, no jerárquico y no significante, sin General, sin memoria organizadora o autómata central, definido únicamente por una circulación de estados. Lo que está en juego en el rizoma es una relación con la sexualidad, pero también con el animal, con el vegetal, con el mundo, con la política, con el libro, con todo lo natural y lo artificial, muy distinta de la relación arborescente: todo tipo de "devenires" (Deleuze & Guattari, 2002, págs. 25-26)

2.1.2 Crítica al pensamiento arborescente. Por otra parte, Deleuze y Guattari realizan una crítica al pensamiento que denominan arborescente, el cual definen de la siguiente manera: la lógica del árbol es una lógica del calco y de la reproducción (Deleuze & Guattari, 2002, pág. 17). El pensamiento arborescente se encuentra en contraposición con el rizomático, porque este último funciona como un mapa, es decir como un sistema abierto:

> el mapa es abierto, conectable en todas sus dimensiones, desmontable, alterable, susceptible de recibir constantes modificaciones. Puede ser roto, alterado, adaptarse a distintos montajes iniciando por un individuo, un grupo o formación social (Deleuze & Guattari, 2002, pág. 18)

Por ende, el rizoma puede conectar un punto con otro cualquiera. El árbol por el contrario se identifica con la lógica binaria y la ramificación dicotómica. Por ello, Deleuze y Guattari intentan ir más allá de la lógica binaria, la cual ha tenido prisionero al pensamiento occidental desde hace siglos.

El árbol se ha convertido en una metáfora que muestra cómo piensa Occidente. El tronco representa la base o, mejor dicho, la unidad en la que todo descansa. Las formas arborescentes se caracterizan por ser troncales, es decir, que tienen un sentido unidireccional, un principio de homogenización. Lo rizomático, por otra parte, es un conjunto de ramificaciones, es un campo en el cual se unen y se reúnen múltiples elementos; además convergen multiplicidades con diversas entradas y salidas.

Lo rizomático se anida en un plan de consistencia que actúa bajo un agenciamiento, que amplía las dimensiones, en una multiplicidad que transforma la naturaleza a medida que aumenta las conexiones en diferentes direcciones el rizoma, avanza con múltiples brotes, se conecta a partir de cualquier punto con otro, está hecho de direcciones cambiantes (Aladino, 2015).

El árbol en cambio, no aumenta sus conexiones ni las direcciones, ya que este sólo ocupa una posición fija. El árbol pone sus raíces profundamente en la tierra, por lo que se configura como un punto estático y la única forma que tiene de crecer es hacia arriba.

Como señalamos anteriormente el calco funciona replicando las estructuras existentes, por el contrario, el mapa está orientado a la experimentación y la adaptación. Este fenómeno lo podemos observar en los sistemas de red. Las redes tienen un proceso de constante invención, se expanden y contraen, emergen y se alejan. Los mapas tienen múltiples vías de entrada como el ciberespacio tiene múltiples puertos de entrada.

Siguiendo con la metáfora de la red podríamos decir que el rizoma al formar una red compleja su forma no puede determinarse y en este sentido para Deleuze y Guattari, el modelo rizomático es la naturaleza de todas las cosas. Incluso si lo miramos desde un árbol, cuando se piensa desde esta perspectiva, siempre hay un afuera en el que hacen rizoma con algo: con el viento, con un animal, con el hombre (Deleuze & Guattari, 2002, págs. 16-17).

2.2 Software Libre modelo rizomático

En el capítulo anterior vimos los elementos que permitieron construir al software libre tal cual lo conocemos hoy pasando por su definición y sus libertades hasta la construcción del *kernel* (Linux) que permitiría concretizar el proyecto de un sistema operativo completamente libre.

En el anterior apartado presentamos lo que podríamos considerar un resumen del concepto de rizoma presentado por Deleuze y Guattari en *Mil Mesetas*, construido a través de tres elementos el primero es la presentación del concepto, el segundo los principios que constituyen el modelo rizomático y finalmente la crítica al pensamiento arborescente. De cierto modo, podríamos decir

que estos tres elementos constituyen también -aunque *grosso modo*- los puntos principales del texto de Deleuze y Guattari.

En este punto debemos hacer la salvedad y como señala el profesor Jorge Francisco Maldonado es bastante arriesgado hacer un *resumen* de un texto como el de *Mil Mesetas* por la extensión de las temáticas que se trabajan, los niveles de explicación que se manejan y las conexiones (que no son pocas) que se posibilitan (Maldonado, 2008, pág. 106).

Lo señalado por el profesor Maldonado nos pone en contexto de dos problemáticas -una a nivel explícito y otro a nivel implícito- importantes respecto al texto de Deleuze y Guattari. El primero sería el que ya señala propiamente la cita del pasaje de la tesis doctoral del profesor Maldonado y es que la presentación de un texto como el de *Mil Mesetas* comporta una dificultad por los niveles de explicación y de conexiones que se posibilitan con él, y en este sentido también resumir lo que se enuncia en el texto de los filósofos franceses sería de cierto modo una tarea muy difícil por lo anteriormente expresado.

Ahora bien, aunque el resumen que hemos realizado teniendo en cuenta lo que hemos señalado anteriormente, sería incompleto precisamente por la propia dificultad interna que comportaría el texto, en el sentido de las posibilidades de explicación y de conexiones internas del propio texto; pero también por las conexiones que se pueden realizar con las demás mesetas presentadas en el conjunto del libro. Además, podríamos ir más allá y ver las conexiones que se pueden establecer en el ámbito interno del conjunto de las mesetas, y también con el conjunto de los conceptos presentado por Deleuze y Guattari en sus obras individuales, como en sus obras escritas a dúo.

Esta dificultad de las conexiones y de las temáticas nos sitúa en una dificultad metódica de cómo abordar el propio conjunto de conceptos presentados en *Rizoma*. Por ello hemos decidido abordar el concepto a través de la figura retórica de la sinécdoque, que consiste en designar una cosa con el nombre de otra con la que existe una relación de inclusión, por lo cual podría utilizarse el nombre del todo por la parte y viceversa; esto con el fin de poder abordar la dificultad que comporta las conexiones y las temáticas del *Rizoma*.

Así las cosas, a continuación, presentaremos un par de sinécdoques que nos permitan, a partir de las partes que abordarán hablar del "todo", sabiendo de cierta forma que abarcar todas las posibilidades que abre el rizoma sería una tarea titánica.

Lo que pretendemos entonces con estos elementos es abrir un mapa, con múltiples posibilidades de entrada y de salida, esto porque si intentamos concentrar el rizoma a través de un solo punto dejaríamos de lado las posibilidades de conexiones. Por el contrario, si presentamos fragmentariamente estos elementos abrimos la posibilidad de múltiples conexiones con otros puntos y que se constituya a través de estas direcciones cambiantes.

2.2.1 GLP: o cómo crear autores rizomorfos. En el apartado 1.3.1 vimos las licencias robustas de la GLP, pero recordemos que esta licencia es la más ampliamente usada en el universo del Software Libre, ya que garantiza la libertad de usar, estudiar, compartir y modificar el software. En este sentido, su propósito es declarar como software libre al software que se encuentre licenciado bajo ella y proteger el software libre de intentos de apropiación que restrinjan las libertades anteriormente señaladas.

También cabe recalcar que esta licencia fue creada por Richard Stallman para el proyecto GNU. La GLP también es la primera licencia copyleft para uso general, en otras palabras, los trabajos derivados sólo pueden ser distribuidos bajo los términos de la misma licencia.

Gracias a la filosofía promulgada por la Free Software Fundation esta licencia garantiza que los usuarios de un programa o de un paquete de software reunidos en la definición de software libre (véase apartado 1.2) se aseguré que el software cada vez que es distribuido, modificado y ampliado se encuentre protegido para que otros usuarios puedan seguir gozando de estos derechos.

Ahora bien, una de las principales diferencias entre GNU/Linux respecto a otros sistemas operativos privativos como Windows o IOS, es que su núcleo y otros componentes del sistema son Software Libre y de código abierto. Su código está basado en el sistema Copyleft[7]. Este sistema funciona gracias a la reciprocidad. Es decir, que todas las obras derivadas de una pieza de Copyleft también debe ser Copyleft.

Aunque el Software Libre es desarrollado a través de la colaboración, frecuentemente se producen de forma independiente el uno del otro, es decir, que muchos de los programadores de software realizan sus proyectos de manera individual. Estos aportes individuales en la construcción de un proyecto colectivo, han permitido que personas muy distintas y diversas realicen aportes al desarrollo del proyecto (Software Libre), a través de perspectivas y de intereses distintos.

Esta independencia es producto de la libertad que tienen los programadores y usuarios de modificar el código fuente de los programas y del núcleo mismo del sistema para construir y reconstruir los distintos elementos que los configuran; así, explorar y experimentar con ellos es

[7] El término surge de las comunidades que desarrollan software libre y hace referencia a una práctica que consiste en el ejercicio del derecho de autor, con el objetivo de permitir la libre distribución de copias y versiones modificadas de la obra. Exigiendo que los derechos de autor sean preservados en las versiones modificadas. Se aplica principalmente a programas informáticos, obras de arte, cultura, ciencia, o cualquier tipo de obra o trabajo.

posible en la medida en que el código se encuentra abierto y a disposición de los programadores y usuarios.

De tal forma que podríamos decir que la independencia de los usuarios y programadores funciona como un agenciamiento:

> Hay, pues, agenciamientos muy diferentes, mapas-calcos, rizomas-raíces, con coeficientes de desterritorialización variables. En los rizomas existen estructuras de árbol o de raíces, y a la inversa, la forma de un árbol o la división de una raíz, pueden ponerse a brotar en forma de rizoma. La localización no depende aquí de análisis teóricos que implican universales, sino de una pragmática que compone las multiplicidades o los conjuntos de intensidades (Deleuze & Guattari, 2002, pág. 20)

Por lo anterior podríamos decir que en la GLP y en el núcleo a través del agenciamiento que generan producen dos ejes. El primero es de relación, ya que, tanto los usuarios como los programadores, se ven conectados a través del uso, las modificaciones y la distribución del sistema y el segundo es el del proceso, puesto que, usuarios y programadores, al desarrollar conjuntamente el sistema, expanden sus límites, incluyendo nuevos elementos y formando nuevos procesos de acumulación de intensidades.

De tal modo que relación y proceso ensamblan elementos heterogéneos, a una red, a una multiplicidad rizomática en la cual la configuración, conexión y desarrollo de los elementos depende de su cofuncionamieto. Por consiguiente, las licencias y el núcleo del Software Libre al permitir la redistribución y la modificación del código fuente (núcleo) producen una innovación que podríamos llamar ontológica; aunque esta ontología no es tanto del ser como del devenir de los elementos implicados (Lama, 2009).

La sentencia del software libre es que todos podrán *agenciarse* en su devenir, pues tendrán permiso de modificar y redistribuir GNU, pero ningún distribuidor se le permitirá restringir su

redistribución posterior. Está claro que la idea de la libertad de acceso que tiene el software libre, se asemeja significativamente a la idea del autor desarrollado por Deleuze y Guattari en *Rizoma:*

> El Anti-Edipo lo escribimos a dúo. Como cada uno de nosotros era varios, en total ya éramos muchos. Aquí hemos utilizado todo lo que nos unía, desde lo más próximo a lo más lejano. Hemos distribuido hábiles seudónimos para que nadie sea reconocible. ¿Por qué hemos conservado nuestros nombres? Por rutina, únicamente por rutina. Para hacernos nosotros también irreconocibles. Para hacer imperceptible, no a nosotros, sino a todo lo que nos hace actuar, experimentar, pensar. Y además porque es agradable hablar con todo el mundo y decir el sol sale cuando todos sabemos que es una manera de habar. *No llegar al punto de ya no decir yo, sino a ese punto en el que ya no tiene importancia decirlo o no decirlo*[8]. Ya no somos nosotros mismo. Cada uno reconocerá los suyos. Nos han ayudado, aspirado, multiplicado (Deleuze & Guattari, 2002, pág. 9)

Si ponemos el pasaje anteriormente citado en el marco del universo del software podríamos decir que, en éste, a grandes rasgos, existen dos tipos de autor -léase desarrollador-.

El primero sería el "autor" de software-arbóreo el cual se reserva los derechos de su obra, pues sostiene la idea que hay una razón analógica entre el software y los objetos materiales. Además, creen que los derechos naturales de autor son una tradición aceptada e indiscutida de nuestra sociedad. Lo que no saben es que desde un punto de vista histórico sucede todo lo contrario, como lo señala Stallman:

[8] Énfasis agregado.

> La idea de los derechos naturales de autor fue propuesta y decididamente rechazada cuando se concibió la Constitución de EE.UU. Esa es la razón por la que la Constitución sólo permite un sistema de copyright y no requiere uno; por esa razón dice que el copyright debe ser temporal. Establece asimismo que el propósito del copyright es promocionar el progreso –no recompensar a los autores. El copyright recompensa a los autores en cierta medida, y a los editores más, pero se concibe como un medio de modificar su comportamiento[9]. (Stallman, 2004, pág. 73)

El otro gran tipo de autor sería el "autor" de software-rizomático, el cual considera que su obra puede ser utilizada por cualquiera, con la única condición de que nadie pueda negarle a otro esos derechos de libre uso, pues, en el momento de que alguien suprima esa condición, está limitando su disponibilidad (por ejemplo, distribuir un código modificado, en el cual no exista la posibilidad de acceder a las fuentes modificadas) este principio rige el desarrollo de software libre. Los programadores de software que actúan de esta manera son rizomorfos, pues hacen que sus desarrollos sirvan para usos extraños.

Una de las tareas del Software Libre es la eliminación del autor como figura rígida, del mismo modo como lo señalan Deleuze y Guattari: Cuando se atribuye el libro a un sujeto, se está descuidando este trabajo de las materias, y la exterioridad de sus relaciones. Se está fabricando un Dios para movimientos geológicos (Deleuze & Guattari, 2002, pág. 9) al eliminar este Dios los programadores de Software Libre están generando líneas de articulación con nuevos programas, nuevas relaciones, nuevos códigos.

[9] Tomo esta cita que hace referencia al proceso Norte Americano, pues ahí se encuentra gran parte de los desarrolladores de Software libre.

En suma, los programadores rizomorfos generan líneas de fuga, con su trabajo surgen movimientos de desterritorialización y de desestatificación. Pues permiten que éste sea re-configurado cuantas veces ellos y los usuarios así lo deseen.

Los programadores rizomorfos están desconfiando la autoridad, por lo tanto, promueven la descentralización. Su trabajo se contrapone claramente al de los programadores arbóreos, los cuales restringen el uso de su software, impiden su *agenciamiento* y su rizomatización. De cualquier modo, hay rizomorfos, creadores de mapas no de calcos y aunque pretendan serlo, nunca se dará algo igual. Los mapas siempre serán diferentes, siempre otros, siempre estarán cambiando su naturaleza, metamorfoseándola.

En conclusión, podemos observar cómo los programadores de Software Libre han construido un rizoma, este rizoma software no tiene principio ni final, se encuentra en constante construcción de una diferencia. Así los programadores se encuentran siempre prestos a generar conjunciones, uniones y descentralizaciones, siempre rizomáticos, nunca arbóreos, nada plantado todo rizoma y como señalan Deleuze y Guattari: Un rizoma no empieza ni acaba, siempre está en el medio, entre las cosas, inter-ser, *intermezzo*. EL árbol es filiación, pero el rizoma tiene la conjunción "y…y…y…" (Deleuze & Guattari, 2002, pág. 29)

Así también el Software Libre, sus programadores y usuarios se encuentran siempre conjugando nuevas posibilidades, nuevas conexiones y nuevas relaciones; con ellas se expande el rizoma, crece y se multiplica.

2.2.2 Software Libre y la cultura libre. Antes de ver la relación que sostiene el software libre y la cultura libre es necesario que nos refiramos brevemente al concepto de cultura.

Si nos referimos a la etimología de la palabra cultura, encontraremos que es un concepto procedente de la noción de naturaleza. Uno de sus significados originales es “producción”, o sea, un control del desarrollo natural (Eagleton, 2001, pág. 11). Es por esto que, en principio, el concepto de cultura designó un proceso esencialmente material y relacionado con todo lo que el hombre producía.

Luego éste se vio transformado en un asunto del espíritu. Este giro representa un desarrollo semántico del concepto dentro del transcurrir histórico de la humanidad y marca el tránsito de lo rural a lo urbano. Pero esta inversión semántica resulta contradictoria, pues las personas cultivadas son aquellas que viven en un medio urbano y quienes en verdad “cultivan”, o sea, quienes viven de la labranza de la tierra no son cultivados.

Es importante que tengamos en cuenta que cultura en su sentido original de “producción”, alude a un control. Por consiguiente, lo cultural es lo que podemos transformar, pero el elemento que hay que alterar tiene su propia existencia autónoma y eso es lo que hace que el concepto de cultura participe del carácter de naturaleza. Igualmente, la naturaleza siempre posee algo de cultural, porque las culturas proceden de una relación continua con la naturaleza.

La relación entre “naturaleza” y “cultura” no es arbitraria, ni tan libre como se puede pensar, puesto que la cultura también es un asunto de generar y seguir reglas, por ende, implica una relación entre lo reglamentado y lo no reglamentado. En este sentido, la idea de cultura, implica una doble negativa: contra el determinismo orgánico, por un lado, y contra la autonomía del espíritu, por otro. Generando así un rechazo tanto del naturalismo como del idealismo. Así las cosas, incluso la producción humana de condición más elevada echa sus más humildes raíces en nuestro entorno biológico y natural (Eagleton, 2001, pág. 16).

En este sentido la cultura funciona como un término descriptivo y valorativo, ya que puede designar lo que ha acontecido, pero también lo que ha de suceder. Esta idea resulta trascendental al momento de oponerse entonces al naturalismo y al idealismo. Así mismo el concepto se contrapone a las ideas de determinismo y voluntarismo.

La palabra "cultura" contiene en sí misma una tensión entre producir y ser producido, entre racionalidad y espontaneidad que se opone a la idea ilustrada de un intelecto inmaterial y desencarnado, pero que también desafía al reduccionismo cultural imperante en gran parte del pensamiento contemporáneo. Como hemos podido observar, el concepto de cultura nos lleva desde lo natural a lo espiritual, por lo que podríamos decir que hay una relación entre estos dos ámbitos.

Teniendo esta definición de cultura, veamos ahora el origen de la cultura libre. La cultura libre se origina con base en los principios del Software Libre, ya que a través de este movimiento se volvió posible copiar y distribuir en principio paquetes de software y posteriormente obras culturales a nivel general, por ejemplo, libros, películas, música, etc., de forma gratuita y sin intermediarios.

Compartir y distribuir se encuentran en contra vía de los negocios de algunas industrias, por ejemplo, la industria cultural y de desarrollo de software, las cuales basan su modelo de negocio en la venta y distribución de estos productos.

Ahora bien, el concepto de cultura libre se popularizó en 2004 tras la publicación del libro del abogado norteamericano Lawrence Lessig *Cultura Libre*. Como señalamos anteriormente este libro se encuentra de cierto modo inspirado en el Software Libre y en el papel que desempeñó Richard Stallman por ser quien desarrolló la idea y formuló el copyleft. En la cultura libre el papel

del copyleft es una pieza clave, ya que permitió que se aplicara la protección de las libertades a otros campos de la cultura.

Otros elementos que ayudaron a moldear la idea de cultura libre fue el surgimiento de dispositivos digitales de copia y almacenamiento de información y el desarrollo a gran escala de internet que jugará un papel determinante en la creación de movimientos en defensa de la cultura libre, lo cual, anudado a la facilidad técnica para el acceso y la reutilización de la cultura, en contraposición con las barreras legales cada vez mayores estimuladas por algunas industrias estadounidenses, pusieron en evidencia una contradicción que estimuló la crítica y el activismo (Fossatti, 2019).

No obstante, en estos distintos movimientos de defensa de la cultura libre existieron diferencias en cuanto a qué se quiere decir libre cuando se habla de la cultura libre. Para ello, en el 2006 fue redactado por la Free Software Fundation y Creative Commons la definición de los trabajos culturales libres, así que para ellos se define libre como:

- La libertad de usar el trabajo y disfrutar de los beneficios de su uso.
- La libertad de estudiar el trabajo y aplicar el conocimiento adquirido de él.
- La libertad de hacer y redistribuir copias, totales o parciales, de la información o expresión.
- La libertad de hacer cambios y mejoras, y distribuir los trabajos derivados.

Como podemos observar, estas libertades son muy parecidas a las promulgadas por la FSF para definir del Software Libre. Aunque con algunas variaciones, vemos que, en esencia, estas libertades, al igual que las del Software Libre, protegen su uso, su modificación y su distribución.

Anteriormente, habíamos señalado que el desarrollo de tecnologías digitales de copia y almacenamiento de información produjeron un desarrollo acelerado de diferentes expresiones culturales; sin embargo, acceder a estas obras se encuentra restringido por los derechos de autor. Lawrence Lessig suele hablar de “leyes que limitan la creatividad”, al referirse a esta situación contradictoria en la cual la participación cultural es cada vez más posible y a la vez más restringida por las leyes (Fossatti, 2019).

Y es que en la actualidad gracias a distintos medios tecnológicos digitales es más fácil acceder y usar productos culturales, pero el uso de estos se encuentra precarizado por las distintas condiciones legales que limitan el uso de los productos culturales. Con esta normatividad se limita también el derecho a la cultura. Derecho que se encuentra consagrado en la *Declaración Universal de los Derechos Humanos* en el artículo 27: Toda persona tiene derecho a tomar parte libremente en la vida cultural de la comunidad, a gozar de las artes y a participar en el progreso científico y en los beneficios que de él resulten.

En este punto debemos aclarar que la cultura libre no es una “cultura gratuita”, sino que es libre en el sentido de la libertad de expresión, del libre albedrío y de las elecciones libres. Como señala Lessig:

> Pero lo hace también indirectamente limitando el alcance de estos derechos, para garantizar que los creadores e innovadores que vengan más tarde sean tan libres como sea posible del control del pasado. Una cultura libre no es una cultura sin propiedad, del mismo modo que el libre mercado no es un mercado en el que todo es libre y gratuito. Lo opuesto a una cultura libre es una "cultura del permiso"--una cultura en la cual los creadores logran crear solamente con el permiso de los poderosos, o de los creadores del pasado (Lessig, 2005, pág. 18)

Ahora debemos preguntarnos ¿cómo se relaciona el modelo rizomático del software libre con la cultura libre? En primer lugar, debemos señalar que las raicillas que han permitido que el software libre se desarrollara, es decir, que las libertades que permitieron distinguir y definir al software de otros modelos de desarrollo de software, se expandieran y se conectaron con otros elementos. De este modo podríamos ver representado el primer y el segundo principio del rizoma que señala que cualquier punto del rizoma puede ser conectado con cualquier otro punto.

También ha tenido lugar una conexión con otros puntos, en este caso de la cultura, representada en manifestaciones literarias, musicales, cinematográficas, científicas, educativas, etc., que constituyen la cultura libre. De este modo se hace lo múltiple:

> Lo múltiple *hay que hacerlo*, pero no añadiendo constantemente una dimensión superior, sino, al contrario, de la forma más simple, a fuerza de sobriedad, al nivel de las dimensiones de que se dispone, siempre n-1 (sólo así, sustrayéndolo, lo Uno forma parte de lo múltiple) Sustraer lo único de la multiplicidad a constituir: escribir a n-1- Este tipo de sistema podría denominarse rizoma (Deleuze & Guattari, 2002, pág. 12)

Con lo anterior podemos observar cómo el rizoma que ha constituido el Software Libre tiene una gran diversidad de extensiones y de conexiones con las artes, las ciencias y las luchas sociales, además, de ver cómo se hace rizoma con el afuera. Pues, al ser rizomatoso, el software libre produce y conecta nuevas raíces que servirán para nuevos usos extraños.

3. Conclusión: Software libre una tecnología rizomática

La tecnología ha jugado a lo largo del tiempo un papel importante en el proceso de profundización de la modernidad, a través de la transformación de la experiencia humana y ejecutando cambios sociales, dando como resultado una época en la que se desarrollan a gran velocidad nuevas y mejores tecnologías, nuevas máquinas que suplantan a las viejas y obsoletas.

De tal modo que podemos señalar como una característica de nuestra época moderna el desarrollo vertiginoso de lo nuevo y además como una época cambiante en la cual las soluciones tecnológicas se hacen cada vez más necesarias para poder afrontar los retos u las necesidades de nuestra sociedad. Por ejemplo, uno de los grandes cambios que ha producido el internet es que ha cambiado por completo la forma en que nos comunicamos, permitiendo que recibamos cada vez más y más rápido la información, no sólo de nuestro entorno cercano, sino también de los lugares más remotos del planeta tierra.

En este sentido, las nuevas tecnologías han hecho posible alcanzar funcionamientos que nos permiten acercarnos al concepto de *Rizoma* de Deleuze y Guattari. Un ejemplo claro de ello sería el Internet, del cual se han realizado varios estudios para entenderlo como un rizoma. Pero más allá del Internet creemos que el Software Libre, a través de su desarrollo, ha tomado cuerpo y forma de rizoma, convirtiéndose de esa manera en una tecnología que podríamos nombrar como rizomática.

En el primer capítulo de este trabajo pudimos ver la génesis y el desarrollo del Software Libre, el cual definimos por medio de las cuatro libertades planteadas por la FSF y podríamos

señalar que el software libre sería todo aquel programa que cumpla con estas libertades y en este sentido permita a los usuarios y programadores estudiar, copiar, distribuir y modificar.

Estas libertades han permitido que el Software Libre se desarrolle a través de una red en la que todos los puntos se unen, se expanden, crecen y permiten desarrollar nuevos programas. En esta red la información se encuentra distribuida, en tanto que se debe tener acceso al código fuente para poder hacer efectivas, las libertades que señalamos en el apartado 1.2 del primer capítulo. En este sentido el Software Libre trabaja como un rizoma, o, en otras palabras, funciona como un modelo rizomático.

En efecto, el Software Libre, permite, en primer lugar, que cualquier persona sin distinción de raza, género, etc., pueda desarrollar paquetes de software y estos desarrollos pueden producirse desde cualquier punto del mundo, sin necesidad de una central y sin la necesidad de pedir permiso para estos desarrollos. También podemos agregar que estos desarrollos y modificaciones que se realicen al núcleo del sistema pueden servir a cualquier propósito. Más allá de tener distribuciones de un sistema operativo, el Software Libre ha permitido desarrollar programas para multimedia, ofimática, desarrollo web, videojuegos, edición de imágenes, entre otros.

El Software Libre, al tener un desarrollo en el cual no hay un punto central, es multi-centro. Ahora bien, podría pensarse que el *kernel* del sistema funciona como un centro, en tanto que es necesario para el desarrollo de los demás elementos, pero en realidad el núcleo aquí no funciona como un centro, sino más bien como una raicilla que ha permitido extender los horizontes y los desarrollos a otras áreas.

La multi-centralidad no es la única característica que nos permitiría ver el modelo rizomático del Software Libre, pues otros elementos que lo han constituido como la General Public

License ha permitido que los principios del Software Libre sean desterritorializados de su sector de origen para aplicarse a otras materias, por ejemplo, a la música, el cine, los libros y las artes en general, produciendo una cultura libre y movimientos sociales en defensa de esa cultura libre, de tal modo que el modelo rizomático del Software Libre hace un rizoma con el mundo:

> Hace rizoma con el mundo, hay una evolución aparalela del libro y del mundo, el libro asegura la desterritorialización del mundo, pero el mundo efectúa una reterritorialización del libro, que a su vez se desterritorializa en sí mismo en el mundo, (si puede y es capaz) (Deleuze & Guattari, 2002, pág. 16)

Por lo anterior, el Software Libre también sería rizomático en tanto que puede ser des-territorializado de su entorno inicial para aplicarse en sectores distintos, permitiendo que las raíces se expandan y crezcan. Incluso, si algunas son cortadas, se conecta con nuevos elementos en un largo proceso de territorialización, desterritorialización y reterritorialización con alcance en todos los ámbitos de la vida, construyendo múltiples agenciamientos y múltiples conexiones.

En conclusión, tenemos que en un mundo en el que los cambios tecnológicos están sucediendo de forma tan rápida y vertiginosa el concepto de *Rizoma* de Gilles Deleuze y Félix Guattari nos permite comprender las tecnologías de funcionamiento múltiple (Software Libre, Internet, etc.). De esta forma se puede ver las conexiones y agenciamientos que realizan y observar cómo crecen y se expanden las raicillas abriendo un mapa de nuevas posibilidades.

Referencias bibliográficas

Adell, J., & Bernabé, I. (2007). Software libre en educación. En J. C. Almanera, *Tecnología Educativa* (págs. 173-193). Madird: McGraw-Hill.

Aho, A. (2008). *Compiladores. Principios, técnicas y herramientas.* México : Pearson Educación.

Aladino, E. (2015). *Cuatro Patios*. Obtenido de Cuatro patios: http://revista4patios.com/pensamiento-rizomaacutetico.html

Alfaro, F. M. (2000). *Introducción a Linux.* Lima: Instituto Nacional de Estadística e Informática.

Balari, S. (2014). *Teoría de lenguajes formales.* Barcelona: Universidad Autonoma de Barcelona.

Best, S., & Kellner, D. (1991). *Postmodern Theory. Critical Interrogations.* New York: Guilford Press.

Brookshear, G. (1993). *Teoría de la computación. Lenguajes formales, autómatas y complejidad* (1 ed.). Wilmington: Addidon Wesley Iberoamericana.

Castells, M. (2010). *The Information Age: The Rise of the Network Society* (second ed., Vol. I). Singapore: Blackwell Publishing Ltd.

Deleuze, G., & Guattari, F. (2002). *Mil Mesetas: Capitalismo y esquizofrenia.* Valencia: Pre-Textos.

Díaz, E. (2007). Para leer "Rizoma". En E. Díaz, *Entre la tecnociencia y el deseo* (págs. 89-108). Buenos Aires: Biblos.

Eagleton, T. (2001). *La idea de cultura.* Barcelona: Paidós.

Española, R. A. (2014). *Diccionario de la Real Academia Española* . Madrid: RAE.

Fossatti, M. (03 de 05 de 2019). *Ártica*. Obtenido de Ártica: https://www.articaonline.com/2014/08/que-es-la-cultura-libre-tema-1-encirc/

Fumagalli, A. (2010). *Bioeconomía y capitalismo cognitivo.* Madrid: Traficantes de Sueños.

Hernández, J. M. (2005). *Software Libre: técnicamente viable,económicamente justo y socialmente viable.* Barcelona: Infonomia.

IEEE. (1990). *Standard Glossary of Software Engineering Terminology.* New York: IEEE.

Juárez, M. C., Gómez Herrera, W. G., & Torres Sánchez, S. (2006). *Software Libre vs. Software Propietario: ventajas y desventajas.* México.

Lama, J. P. (2009). La orquídea y la avispa hacen mapa en el medio de un rizoma: Cartografía y máquinas, releyendo a Deleuze y Guattari. *Pro-Posições*, 121-149.

Lessig, L. (2005). *Por una cultura Libre.* Madrid: Traficante de sueños.

Lipszyc, D. (2003). *Derechos de autor y derechos conexos* . Unesco.

Maldonado, J. F. (2008). *Música y Creación: Un sentido en el pensamiento de Gilles Deleuze.* Madrid: Universidad Autónoma de Madrid.

Pressman, R. (2010). *Ingeniería del Software. Un enfoque práctico.* (7 ed.). (V. C. Olguín, & J. Enríquez Brito, Trads.) México : Mc Graw-Hill.

Quer, P. F. (2000). *Diccionario de Botánica.* Barcelona: Península.

Rosa, F. d., & Heinz, F. (2007). *Guía práctica sobre Software Libre. Su selección y aplicación local en América Latina y el Caribe.* Montevideo: UNESCO.

Sommerville, I. (2011). *Ingeniería de Software* (9 ed.). (V. C. Olguín, Trad.) México: Pearson Educación.

Stallman, R. (2004). Por que el software no debe tener propietarios. En R. Stalliman, *Internet, Hackers y Software Libre* (págs. 71-78). Buenos Aires: Editora Fantasma.

Stallman, R. (2004). *Software libre para una sociedad libre.* Madrid: Traficantes de Sueños.

Strasburger, E. A. (2004). *Tratado de Botánica* . Barcelona: Ediciones Omega.

Tanenbaum, A. (2008). *Modern Operating Systems* (Fourth ed.). New Jersey: Pearson Prentice Hall.

Tanenbaum, A. (March de 2016). Lessons Learned from 30 Years of MINIX. *Communications of the ACM, 59*(3), 70-78.

Tanenbaum, A., & Woodhull, A. (2006). *Operating Systems Desing and Implementation* (Third ed.). United States of America: Prentice Hall.

Vidal, M. (2004). Cooperación sin mando: Una introducción al software libre. En C. Gradin, *Internet, hackers y software libre* (págs. 45-68). Argentina: Editora Fantasma.

Williams, S. (2010). *Free as in Freedom (2.0): Richard Stallman and the Free Software Revolution.* Boston: Free Software Fundation .