UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo e implementación de un sistema de monitoreo de señales de T.V. y radio para el control de la publicidad de partidos políticos y candidatos independientes

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Eugenio Kuri Sainz

ASESOR DE INFORME

Ing. Jesús Javier Cortés Rosas

Ciudad Universitaria, Cd. Mx., 2016

ÍNDICE

CAPÍTULO 1 DESCRIPCIÓN DE LA EMPRESA 6

CAPÍTULO 2 DESCRIPCIÓN DEL PUESTO DE TRABAJO 7

CAPÍTULO 3 MARCO TEÓRICO 8

3.1 ANÁLISIS DE HUELLA ACÚSTICA 8

3.1.2 APLICACIONES DEL ANÁLISIS DE HUELLAS ACÚSTICAS 10

3.1.3 EL PROCESO DE GENERACIÓN Y ANÁLISIS DE UNA HUELLA ACÚSTICA 11

3.1.4 ECHOPRINT 14

3.1.5 PREPROCESAMIENTO Y TRANSFORMACIÓN 14

3.1.6 CREACIÓN DEL HASH 15

3.1.7 ALMACENAMIENTO 16

3.1.8 BÚSQUEDAS 17

3.2 BASES DE DATOS NOSQL 18

3.3 PROGRAMACIÓN ORIENTADA A OBJETOS 22

3.4 SCRUM 26

3.5 SPRING 31

CAPÍTULO 4 Antecedentes del tema 34

CAPÍTULO 5 Descripción del sistema 38

5.1 CARACTERÍSTICAS GENERALES 38

5.2 ARQUITECTURA DE SOFTWARE 42

5.3 ARQUITECTURA DE HARDWARE 48

CAPÍTULO 6 Contexto de la participación profesional 52

CAPÍTULO 7 Análisis y metodología empleada 55

CAPÍTULO 8 Resultados 57

CAPÍTULO 9 Conclusiones 58

Bibliografía 61

Figuras

Figura 1: Metodología de creación de huella acústica 12

Figura 2: Proceso de detección de huella acústica 13

Figura 3: Ejemplo de onsets en bandas diferentes 15

Figura 4: Diferencia de tiempo entre pares de onsets 16

Figura 5: Ejemplo de jerarquías 24

Figura 6: Actividades dentro de la metodología SCRUM 29

Figura 7: Proceso de digitalización 39

Figura 8: Proceso de Ingesta 40

Figura 9: Proceso de detección 41

Figura 10: Proceso de generación de testigos 42

Figura 11: Arquitectura general de software del sistema 43

Figura 12: Arquitectura de hardware del sistema 48

INTRODUCCIÓN

El presente informe describe de forma detallada cómo participé en el desarrollo e implementación de Fingercrab®, un sistema de monitoreo de señales de televisión y radio que utiliza actualmente el Instituto Nacional Electoral (INE) para detectar los anuncios publicitarios de los partidos políticos, candidatos independientes y demás entidades políticas; además de grabar todas las señales durante los tiempos autorizados para transmitir anuncios políticos (de 06:00 a 24:00 horas) para tener evidencia confiable y poder verificar que las detecciones hechas por el sistema son correctas.

En este informe demostraré que durante el desarrollo de este proyecto apliqué los conocimientos adquiridos durante mi formación como ingeniero, ya que es un proyecto de una gran complejidad, con un alcance enorme y que impacta directamente el curso político del país, pues la gran mayoría de los ciudadanos deciden por qué candidato o partido votar influidos en gran medida por la publicidad que ven y escuchan en los medios monitoreados por este sistema, por lo que un buen control sobre éstos es crucial para tener unas elecciones limpias y equitativas.

Para que el INE decidiera utilizar el sistema creado por mi equipo de trabajo, fue necesario participar en una licitación pública e internacional, en la que competimos contra quince empresas y finalmente en las pruebas de concepto demostramos tener la mejor solución.

OBJETIVO

El presente informe tiene como objetivo demostrar los conocimientos adquiridos durante mis estudios en la carrera de ingeniería en computación, al aplicarlos para desarrollar un sistema de la complejidad y envergadura como es el sistema de monitoreo de señales de T.V. y radio que utiliza actualmente el Instituto Nacional Electoral para garantizar que los actores políticos cumplan con las normas que rigen el uso de la radio y la televisión para emplear publicidad electoral durante y fuera de las campañas electorales.

CAPÍTULO 1
DESCRIPCIÓN DE LA EMPRESA

Grupo de Tecnología Cibernética S.A. de C.V. (Grupo Tecno) es una empresa mexicana con 31 años de experiencia y más de 250 colaboradores.

Esta empresa es distribuidora de los principales fabricantes de soluciones de infraestructura y ha logrado ser el distribuidor más grande de Oracle en Latinoamérica.

Recientemente, Grupo Tecno incursionó en el mercado de software desarrollando sus propias soluciones, continuando con la calidad y excelencia que lo han caracterizado a lo largo de su historia, y así ha logrado crear sistemas de software de la más alta calidad que han dejado completamente satisfechos a los clientes más exigentes.

La misión de Grupo Tecno es generar valor, confianza y seguridad a organizaciones gubernamentales y privadas, a colaboradores, proveedores, asociados y accionistas, a través del diseño, implementación y soporte de integraciones de infraestructura tecnológica y soluciones de negocio.

Su visión es ser una empresa eficiente y eficaz en la integración de soluciones y servicios de tecnologías de información y comunicaciones.

Grupo Tecno es una empresa incluyente, ya que entre sus colaboradores se encuentra gente con capacidades diferentes, a quienes se les proporciona todo lo necesario dentro de las instalaciones para que puedan laborar y desarrollarse exactamente de la misma forma que cualquier otro colaborador.

CAPÍTULO 2
DESCRIPCIÓN DEL PUESTO DE TRABAJO

Yo soy desarrollador de software Jr. en Grupo Tecno desde hace más de un año y medio, sin embargo mis responsabilidades van más allá de la programación.
Durante el desarrollo del proyecto descrito en este informe, cumplí con la función de apoyar en la programación, ya que fui responsable de la administración de la base de datos y del desarrollo de tres módulos en el sistema, además de trabajar en conjunto con mis compañeros para desarrollar otros módulos, por lo que tuve participación en el desarrollo de prácticamente todos los módulos que integran a Fingercrab®. Sin embargo, una vez que el sistema estuvo listo para salir a producción, también participé en la implementación, configuración de los servidores, instalación y configuración del sistema operativo, configuración de la red que comunica los distintos componentes del sistema, presentación del sistema al cliente y soporte técnico. Además, actualmente también tengo asignadas las responsabilidades de venta y preventa de bases de datos Oracle y soluciones en la nube.

Todas estas funciones realizadas dentro de la empresa me han permitido aprender sobre casi cualquier área relacionada con mis estudios de ingeniería en computación, además de que me ha permitido aplicar y reafirmar los conocimientos que adquirí en la universidad.

CAPÍTULO 3
MARCO TEÓRICO

3.1 ANÁLISIS DE HUELLA ACÚSTICA

El análisis de huella acústica es un proceso que utiliza computadoras para analizar pequeños fragmentos de grabaciones de audio para identificarlo (puede ser una canción o cualquier otra grabación). Los sistemas de análisis de huella acústica identifican el contenido del audio y buscan dentro de una base de datos de referencia para encontrar grabaciones con características similares. Estos sistemas pueden encontrar coincidencias entre el audio analizado y las referencias en su base de datos, aun cuando el audio a analizar haya sido grabado en espacios públicos y contenga ruido. Existen diferentes algoritmos de audio, y cada uno de ellos puede ser más útil para un tipo de análisis en específico. Por ejemplo, hay algoritmos más eficientes para analizar fragmentos muy cortos, y a su vez existen otros algoritmos que están diseñados para obtener un resultado más certero en ambientes con ruido. En el presente trabajo explicaré el funcionamiento general de los algoritmos de generación y análisis de huella acústica, analizando a fondo un algoritmo libre llamado *echoprint*, que fue el utilizado en el proyecto descrito en este informe. Para un análisis detallado de los distintos algoritmos existentes y su comparación, favor de consultar la tesis desarrollada por Alatair Porter, que se titula *Evaluating musical fingerprinting systems* [3].

La generación de una huella acústica se logra obteniendo una muestra corta de una grabación de audio desconocida y generando metadatos de dicha grabación. Estos metadatos se generan convirtiendo la señal de audio en una serie de valores numéricos cortos llamados *hashes*, que sirven para tener un identificador único de la grabación. Los sistemas de análisis de huellas acústicas tienen grandes bases de datos con huellas acústicas para todas las grabaciones que se deseen detectar. Para identificar una grabación de audio, la huella acústica de la grabación es generada y es comparada con las huellas acústicas de referencia almacenadas en

la base de datos para encontrar huellas idénticas o muy parecidas. Las huellas acústicas utilizan información del espectro acústico de la señal a bajo nivel para generar un identificador único de audio.

Idealmente, un sistema de análisis de huella acústica debe reconocer una grabación de la misma forma en la que un humano lo hace. Por ejemplo, los humanos podemos identificar una canción que conocemos después de escuchar sólo un fragmento de la misma, aun cuando exista un poco de ruido en el lugar en el que nos encontramos y haya ligeras variaciones en la velocidad de la canción. Esto significa que el algoritmo generador del *hash* debe utilizar las mismas características que nosotros percibimos en una canción para generar su *hash*, sin importar en qué formato se encuentra dicha grabación o cómo fue codificada. Este tipo de sistemas se conocen como sistemas de identificación basados en contenido (*CBIDs* por sus siglas en inglés), y reciben este nombre porque identifican las grabaciones con base en su contenido acústico en lugar de hacerlo por la forma en la que el archivo está almacenado en la computadora.

Para que un algoritmo de análisis de huella acústica coincida con una huella almacenada en la base de datos, un algoritmo de generación de huella acústica debe generar *hashes* idénticos para la huella de referencia y la grabación que se está analizando. Los *hashes* deben ser idénticos, incluso cuando la grabación es corta o si la grabación suena similar para el ser humano aunque la señal sea distinta. Por ejemplo, si la grabación se hizo con un micrófono en un cuarto ruidoso, entonces el algoritmo de generación de huella acústica debe ser capaz de separar el contenido de la grabación (generalmente es la música la que se desea analizar) del ruido externo grabado por el micrófono.

Las bases de datos de referencia deben contener las huellas acústicas para todas las grabaciones que se desean comparar, cada una de ellas generada para el total de la duración de la grabación. Esto sirve para que en caso de que sólo se tenga una porción de la grabación para analizar, ésta pueda ser identificada sin la necesidad de tener la grabación completa. De esta forma, el segmento puede ser grabado en cualquier punto de la grabación, lo que facilita la identificación de un audio en lugares públicos.

Por otro lado, los sistemas de análisis de huella acústica no deben almacenar una copia del audio que se utilizó para generar la huella, pues basta con que conserven la huella generada para hacer el análisis.

3.1.2 APLICACIONES DEL ANÁLISIS DE HUELLAS ACÚSTICAS

El mercado más grande para las aplicaciones de análisis de huellas acústicas se encuentra entre los consumidores de música. Además de permitir identificar canciones desconocidas, los sistemas de análisis de huellas acústicas pueden ser utilizados para determinar si se poseen los derechos para subir o reproducir determinada canción, para rastrear canciones mientras son distribuidas a los consumidores a través de la radio o cualquier otro medio, o para darle valor a los consumidores.

Los dispositivos de reproducción de música pueden verificar la huella de una señal de audio para determinar si está autorizado para reproducir dicha señal. Por ejemplo, las compañías que se dedican a hacer copias de discos compactos, se pueden asegurar de que no estén duplicando audio para el cual el cliente no tenga una licencia de copia.

El análisis de huellas acústicas también es utilizado por servicios en línea como YouTube o Soundcloud. En estos servicios, el sistema de verificación de derechos automático identifica la música reproducida en los videos o audios y verifica que el usuario que subió el material tenga los derechos necesarios, verificándolo en una base de datos. En el caso de YouTube, al identificar que un usuario publicó material sin tener los derechos, le permite al poseedor de los derechos elegir entre las siguientes opciones: retirar el material, permitir que siga publicado o permitir que siga publicado con publicidad que le generará un ingreso.

Este tipo de sistemas también puede ser utilizado por las empresas de monitoreo de medios para identificar el material que se está transmitiendo para distintos fines. Lo más común es que se utilicen para crear la lista de las canciones que fueron transmitidas en una estación de radio. Sin embargo, esta señal también puede ser monitoreada con otros fines como es el caso del proyecto presentado en este informe. En este proyecto, el monitoreo se hace con la finalidad de identificar cuántas veces se transmitió determinado anuncio, en qué horario y cuál fue su duración, para poder definir si el anunciante y el medio de comunicación cumplieron o no con la legislación correspondiente.

3.1.3 EL PROCESO DE GENERACIÓN Y ANÁLISIS DE UNA HUELLA ACÚSTICA

La mayoría de los algoritmos de generación de huella acústica siguen una serie de pasos para transformar una señal de audio a una huella acústica: pre-procesamiento, encuadrado y traslape, transformación y análisis, extracción de características y generación de huella acústica. La figura 1 contiene un diagrama de flujo con la metodología para la generación de huellas acústicas.

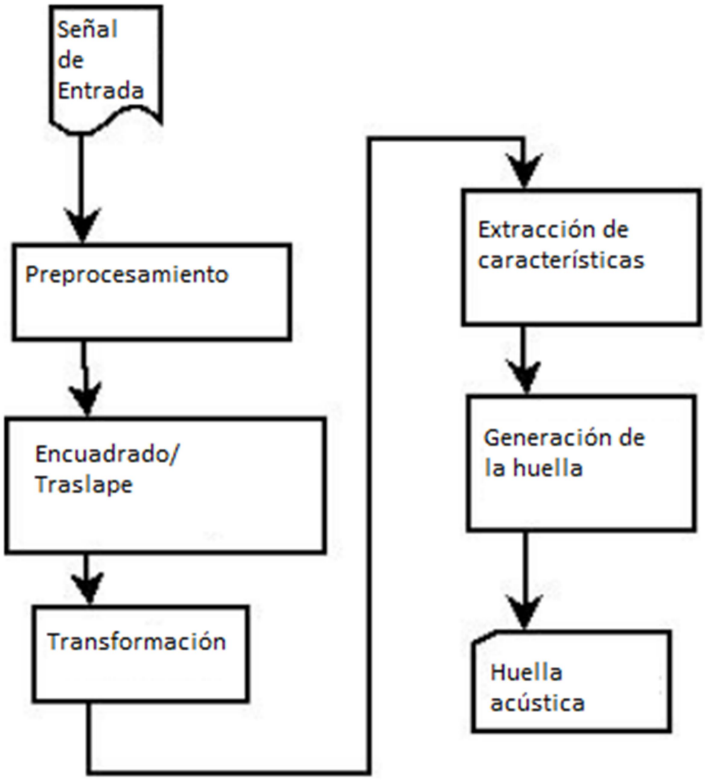

Figura 1: Metodología de creación de huella acústica

En el primer paso (preprocesamiento), todas las señales de entrada son convertidas a un formato común para poder ser analizado. Frecuentemente, este paso implica convertir la señal de entrada a una señal monoaural y disminuir la frecuencia de muestreo a la frecuencia estándar de los discos compactos, que es 44,1000 Hz. Posteriormente, un algoritmo toma la señal de audio (que se encuentra en el dominio del tiempo) y la convierte en una señal en el dominio de la frecuencia, de forma que se pueda extraer más información. Durante el encuadrado y traslape se determina cuántas muestras deben considerarse cuando se realice la transformación de una señal en el dominio del tiempo. Cada *frame* tiene una ventana para ayudar en los cálculos, y los *frames* son procesados traslapando pedazos de la señal en el dominio del tiempo. El proceso de extracción de características toma la señal que ha sido convertida al dominio de la frecuencia y selecciona características que identifiquen el audio. Finalmente, una vez que las características han sido elegidas y extraídas de la señal, deben ser convertidas a

una representación en forma de huella acústica que pueda ser almacenada en una base de datos y comparada con señales desconocidas.

Una vez generada y almacenada la huella acústica, está lista para ser utilizada por un sistema de detección de huella acústica. El proceso de detección debe ser capaz de tomar una señal de audio desconocida y encontrar su par en la base de datos de huellas acústicas, regresando información sobre la huella acústica encontrada como su identificador y la posición en el tiempo de la canción en la que estas dos coinciden. Si el sistema no encuentra coincidencias entre la señal analizada y las huellas almacenadas en la base de datos, es preferible que indique que no se encontró a que regrese una respuesta equivocada. Esto significa que el algoritmo debe ser muy preciso, ya que además de limpiar el ruido, no debe confundir dos audios distintos, en la medida de lo posible.

En el diagrama de la figura 2, se puede observar de forma gráfica la manera en la que un sistema de detección de audio realiza el proceso de detección.

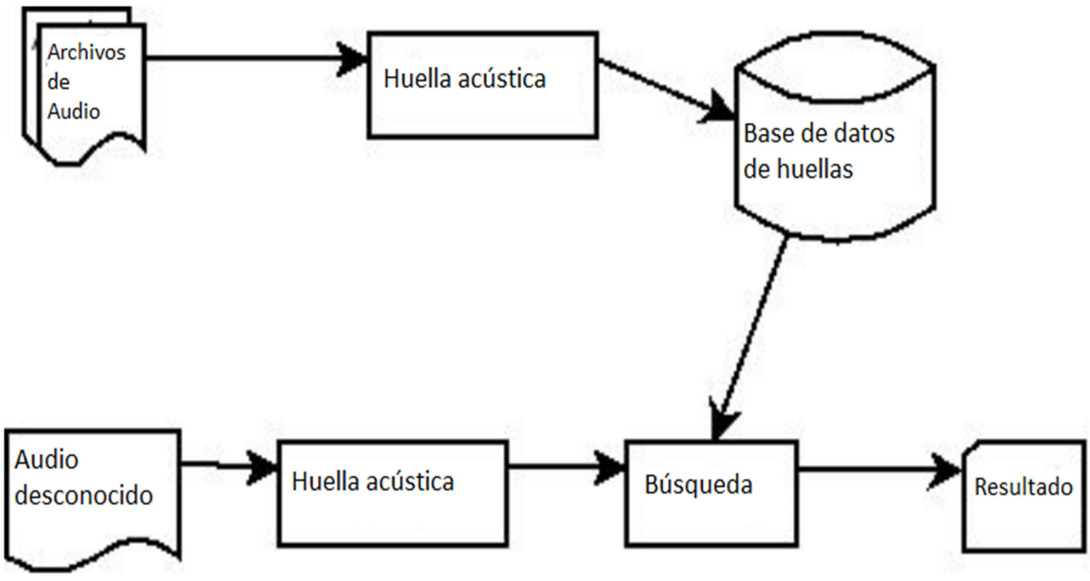

Figura 2: Proceso de detección de huella acústica

3.1.4 ECHOPRINT

Echoprint es un servidor y un algoritmo de generación de huella acústico gratuito, creado por la compañía de tecnología musical The Echo Nest. Este algoritmo está diseñado para ser robusto contra el ruido en el audio analizado y el servidor puede ser escalado para soportar más de 50 peticiones de detección concurrentes por segundo. Tanto el algoritmo de generación de huellas como el servidor están disponibles bajo licencias de código abierto (MIT y Apache 2.0, respectivamente). The Echo Nest tiene un servidor de huellas acústicas que realiza más de 5 millones de búsquedas diarias[8].

El algoritmo *echoprint* funciona encontrando *onsets*, que son puntos en el tiempo en donde ocurren notas musicales. Las características son creadas calculando la diferencia en tiempo entre estos *onsets* y creando un *hash* de estos valore de tiempo. Las detecciones se realizan cuando se encuentran *hashes* idénticos en la base de datos.

3.1.5 PREPROCESAMIENTO Y TRANSFORMACIÓN

Las señales de audio que son introducidas al algoritmo de *echoprint* son convertidas a monoaurales y su tasa de muestreo es reducida a 11025 Hz. Para prevenir que eventos ruidosos como truenos o alarmas sean confundidos con el audio original de la canción, la señal de entrada es “blanqueada”. Dicho blanqueamiento se realiza por medio de un filtro predictor lineal de 40 polos, el cual es generado a partir de la señal de entrada. Este filtro cambia cada muestra a un valor más plano calculado a partir de las 40 muestras previas. Este proceso reduce la amplitud de picos inesperados en la señal.

3.1.6 CREACIÓN DEL HASH

Los hashes en *echoprint* son calculados con base en la diferencia de *onsets* musicales en cada banda. El primer paso en el proceso de generación del *hash* consiste en detectar los *onsets* en la señal de audio, los cuales son detectados en cada banda de frecuencia de forma independiente. En cada banda, un seguidor de envolvente es utilizado para medir la amplitud de la misma, registrando un *onset* cuando dicha amplitud alcanza cierto umbral. Después de que haya sido detectado un *onset*, deben pasar 128 muestras antes del siguiente *onset*. La amplitud del *onset* es multiplicada por una curva exponencial decreciente para calcular un nuevo valor de umbral. Los *onsets* subsecuentes detectados deben exceder dicho umbral para ser tomados en cuenta, y el multiplicador se adapta al número de *onsets* que han sido detectados. *Echoprint* tiene el objetivo de generar un *onset* por segundo en cada banda de frecuencia, por lo que si los *onsets* son generados a una tasa más alta, el multiplicador es incrementado, dando como resultado un umbral más alto que debe ser excedido. Si la tasa de *onsets* decrece demasiado, el multiplicador es decrementado para compensar. La siguiente figura muestra una representación gráfica de una pista de audio generada por el banco de filtros y dividida en ocho bandas. Los *onsets* en cada banda están superpuestos y pueden ocurrir en diferentes tiempos en cada banda de frecuencia.

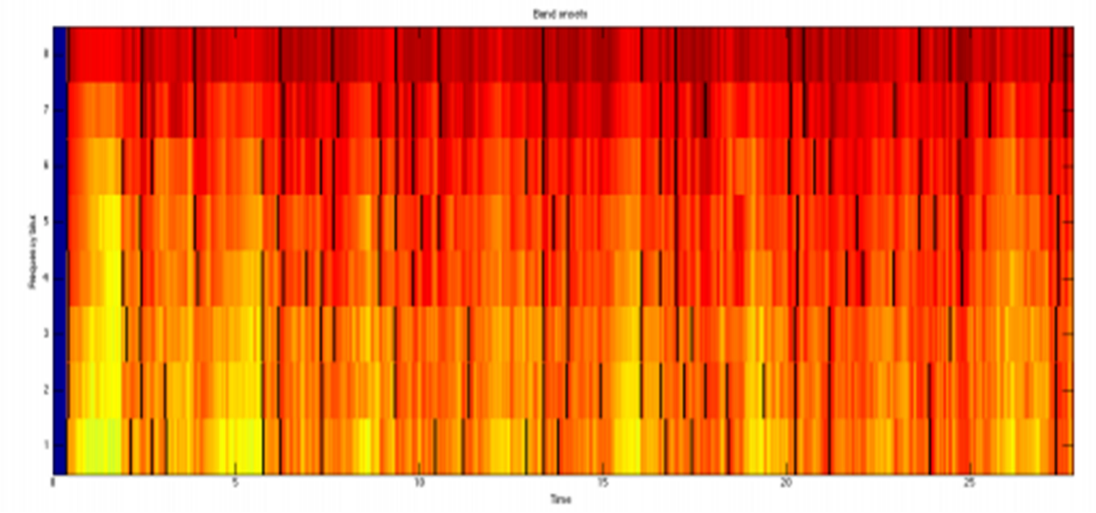

Figura 3: Ejemplo de onsets en bandas diferentes

Para codificar los *onsets* en valores numéricos, el algoritmo considera el tiempo de cada *onset* (o) y el tiempo de sus cuatro sucesores (s1-s4). Un valor *hash* es creado tomando la diferencia de tiempo entre pares de los cinco *onsets*, y la banda en donde ocurren, como se muestra en la siguiente figura:

bytes 1–2	$s1 - o$	$s1 - o$	$s2 - o$	$s1 - o$	$s2 - o$	$s3 - o$
bytes 3–4	$s2 - s1$	$s3 - s1$	$s3 - s2$	$s4 - s1$	$s4 - s2$	$s4 - s3$

Figura 4: Diferencia de tiempo entre pares de onsets

Los dos valores *hash* y el índice de la banda son almacenados en un número de 40 bits (5 bytes), en el que dos bytes son utilizados para cada delta y el byte sobrante representa el índice de la banda. Posteriormente el número es reducido a un entero de 32 bits utilizando el algoritmo Murmur Hash.

Cada *onset* y su conjunto de sucesores generan seis *hashes*. Al emparejar los *onsets* y sus sucesores, el algoritmo se vuelve más robusto contra errores de detección de *onsets*. Si no se detecta un *onset*, aun así habrá algunos *hashes* generados para ese punto en el tiempo. Con este método se obtienen aproximadamente 48 *hashes* por cada segundo de audio (8 bandas, 1 *onset* por segundo, 6 *hashes* por *onset*).

3.1.7 ALMACENAMIENTO

Los hashes son empatados con el tiempo en que ocurre el *onset* en la solicitud de detección de audio. Los pares de *hashes* desfasados en una única grabación son divididos en un número de sub-grabaciones, cada una de 60 segundos, sobreponiéndose a la sub-grabación previa por 30 segundos. Los *hashes* son divididos de esta manera porque el componente que encuentra coincidencias en el sistema califica las grabaciones por el número de veces que un *hash* en la grabación coincide con los *hashes* en el audio a analizar. Si los *hashes* no se dividieran, entonces las grabaciones largas recibirían una ventaja injusta en la fase

de búsqueda porque las grabaciones con contenido repetido podrían generar el mismo valor de *hash* en varios puntos de la grabación. Los valores de los *hashes* son almacenados con un índice invertido, en donde cada valor hace referencia a la sub-grabación y al tiempo en la grabación completa en que ocurrió el *hash*. El servidor de aplicaciones de *echoprint* utiliza Apache Solr[9], un motor de búsqueda de texto veloz, para almacenar el índice del *hash*. El conjunto completo de diferencias de valores *hash* para cada grabación es almacenado en una base de datos distinta para ser utilizado durante el proceso de búsqueda.

3.1.8 BÚSQUEDAS

Las búsquedas son realizadas en dos pasos. El mismo proceso de generación de huella es realizado sobre la señal a analizar, dando como resultado un conjunto de diferencias de pares de *hashes*. En el primer paso, los valores de tiempo son descartados y se busca en el índice invertido para encontrar todas las sub-grabaciones de 60 segundos que contengan un valor *hash* que exista también en la señal que se está analizando. Estas sub-grabaciones son ordenadas por el número de veces en que un *hash* de una grabación coincide con el *hash* de la señal que se está analizando. Si en esta fase se obtienen más de una sub-grabación para la misma grabación, todas excepto una son descartadas.

La calificación final de cada grabación candidata es calculada tratando de empatar los *hashes* de la señal analizada con las grabaciones. Esto se hace calculando la diferencia de tiempo entre el tiempo del *onset* en la señal analizada y el tiempo del *onset* en la grabación por cada *hash* en la grabación, y llevando una suma del número de veces en que ocurre cada diferencia de tiempos. Si una huella de la señal analizada es similar a una huella de la grabación, entonces la suma de diferencias de tiempo dará un número muy grande. Esta suma se utiliza como la calificación de cada grabación candidata. Si la grabación con el número mayor de *hashes* coincidentes tiene más del doble de coincidencias que la siguiente grabación, entonces es considerada como la par de la señal analizada. En caso contrario el algoritmo decide que no encontró coincidencias.

3.2 BASES DE DATOS NOSQL

Las bases de datos relacionales se han convertido en una necesidad básica en cualquier organización, y hay excelentes razones para ello, pues permiten que las organizaciones almacenen y exploten su información de forma segura y confiable. Sin embargo, las aplicaciones que están siendo desarrolladas hoy en día comienzan a superar las capacidades que este tipo de bases de datos ofrece, y a continuación menciono algunas de las razones por las que los desarrolladores y DBAs están buscando otras alternativas:

- Los desarrolladores trabajan con aplicaciones que utilizan tipos de datos nuevos y que cambian dinámicamente.
- Debido a la gran competencia que existe en el mercado de software, cada vez hay menos tiempo para que los desarrolladores programen sus soluciones. La tendencia actual es que los equipos de desarrollo trabajen en *sprints* ágiles y rápidos, desarrollando módulos completos en cuestión de días o incluso de horas, diseñando la base de datos de forma paralela a la programación.
- Antes, las aplicaciones estaban diseñadas para que las utilizara un número limitado de usuarios, sin embargo hoy en día cada vez salen al mercado más aplicaciones utilizadas por números ilimitados de usuarios como es el caso de las redes sociales.

Las bases de datos relacionales tradicionales no están preparadas para satisfacer las nuevas necesidades de los desarrolladores, por lo que cada vez se vuelve más común la migración a la tecnología NoSQL.

La principal diferencia entre las bases de datos NoSQL[11] y las SQL es el modelo de datos que utilizan. A pesar de que hay docenas de bases de datos no relacionales con distintos modelos de datos, los tres más importantes son los siguientes:

- **Modelo de documentos**

  Mientras que las bases de datos relacionales almacenan la información en tablas y columnas, las bases de datos orientadas a documentos almacenan los datos en documentos. Estos documentos generalmente utilizan una estructura como JSON (Javascript Object Notation). Los documentos proporcionan una forma intuitiva de modelar los datos, siguiendo los lineamientos de la programación orientada a objetos. De hecho, cada documento es un objeto, pues puede contener diferentes tipos de datos como *Strings*, *Dates*, *arrays*, etc. En lugar de almacenar un campo en una tabla y acceder a él desde diferentes tablas a través de llaves foráneas, en las bases de datos orientadas a documentos cada campo y sus datos asociados son almacenados en el mismo objeto. Esto simplifica el acceso y elimina la necesidad de costosas y complejas consultas a través de JOINs.

  En una base de datos orientada a documentos, los esquemas son dinámicos porque cada documento puede contener diferentes tipos de campos. Esta flexibilidad puede ser de particular ayuda para modelar datos polimórficos y no estructurados. También facilita la evolución de la aplicación durante el desarrollo, pues permite agregar campos dinámicamente sin la necesidad de cambiar la estructura del esquema o de la base de datos.

  Las bases de datos orientadas a documentos tienen la capacidad de realizar consultas sobre cualquier campo dentro de un documento. Algunos manejadores como MongoDB proporcionan un conjunto muy amplio de opciones de indexado para optimizar una gran variedad de consultas. Además, este tipo de base de datos pueden realizar análisis de datos en tiempo real, sin la necesidad de replicar los datos para utilizar un motor analítico o de búsqueda dedicado.

- **Modelo de grafos**

  Este tipo de bases de datos utiliza estructuras de grafos con nodos, aristas y propiedades para representar los datos. Esencialmente, los datos son

modelados como un conjunto de relaciones entre elementos, en forma de red. Este tipo de bases de dato es poco intuitiva y es difícil de aprender, pero puede ser muy útil para cierto tipo específico de consultas.

Este tipo de base de datos es de gran utilidad cuando las relaciones entre elementos son cruciales para la aplicación, como topologías de red, cadenas de suministro o conexiones de red.

Este modelo de base de datos proporciona modelos de consulta potentes en donde las relaciones simples y complejas pueden ser analizadas para hacer inferencias sobre los datos que hay en el sistema. Los análisis de tipo relacional tienden a ser muy eficientes cuando se utiliza este tipo de base de datos, pero otro tipo de análisis puede ser menos óptimo. En resumen, las bases de datos de grafos son poco utilizadas para aplicaciones de propósito general.

- **Modelos llave-valor y de columnas anchas**.

Este tipo de base de datos es el que tiene una estructura más básica, pues consiste únicamente en una llave que hace referencia a un valor específico. El valor no es relevante para el sistema, pues éste sólo puede ser mostrado cuando se realiza una consulta a través de la llave. Este modelo puede ser muy útil para representar datos polimórficos y no estructurados, pues el valor al que apunta determinada llave puede cambiar dinámicamente.

Este tipo de base de datos es de gran utilidad para un pequeño conjunto de aplicaciones que realiza consultas únicamente a través de una llave única. La ventaja de estos sistemas es su desempeño y escalamiento, ya que puede ser optimizado por la simplicidad de los patrones de acceso a datos y la opacidad de los datos por sí mismos. Además ofrece la posibilidad de obtener y actualizar datos basados en un rango limitado de llaves o en una única llave primaria.

A comparación de las bases de datos relacionales, las bases de datos NoSQL ofrecen una mayor facilidad para escalamiento, proporcionan un desempeño mayor y su modelo de datos ataca diversos problemas que las bases de datos relacionales no están diseñadas para resolver, como son:

- Grandes volúmenes de datos no estructurados, semiestructurados o de estructuras dinámicas.
- *Sprints* ágiles, iteraciones rápidas de esquemas y desarrollo rápido de módulos.
- Programación orientada a objetos que es flexible y fácil de usar.
- Arquitectura distribuida geográficamente y escalable en lugar de arquitectura cara y monolítica.

Las bases de datos relacionales requieren que los esquemas sean diseñados antes de almacenar los datos. Por ejemplo, si queremos almacenar datos sobre nuestros clientes como es el número telefónico, nombre, apellido, dirección, ciudad y estado en una base de datos SQL, el arquitecto de base de datos debe definir dichos campos antes de almacenar la información.

Cuando se están realizando desarrollos ágiles, esta característica resulta contraproducente porque cada vez que sea necesario agregar una nueva funcionalidad, es muy probable que el esquema de la base de datos cambie. Esto implica una migración completa de la base de datos al nuevo esquema. Si la base de datos es grande, este proceso puede ser muy lento, lo que llevaría a un retraso en el desarrollo. Y peor aún, si estos cambios en el alcance del proyecto son frecuentes, porque el desarrollo se está haciendo ágil y veloz, este tiempo muerto se vuelve frecuente y puede llegar a afectar la fecha de entrega del sistema. Además, en las bases de datos relacionales tampoco es posible utilizar datos que son completamente no estructurados o desconocidos.

Las bases de datos NoSQL están diseñadas para permitir la inserción de datos sin tener un esquema definido previamente. Esto les facilita a los programadores el realizar cambios constantes en tiempo real, sin tener que preocuparse por la interrupción de los servicios o por los atrasos que la actualización del esquema

pueda implicar, lo que conlleva a un desarrollo más ágil y rápido, la integración del código es más confiable y se necesita menos administración de base de datos.

3.3 PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos[4] es un paradigma de programación que se basa en el concepto de objetos como entidades con características propias, que pueden interactuar con otros objetos, pero siempre conservando las características que lo definen.

Este paradigma surgido en la década de los 60, se ha vuelto cada vez más popular debido a que tiene características que facilitan en gran medida la programación y el mantenimiento del código.

El código en un programa puede estar organizado de una de las siguientes formas: con base en los datos y con base en el código. Es decir, el código de un programa puede estar enfocado en “lo que está pasando” (código) como es el caso de los programas en C, o puede estar enfocado en “qué elementos son afectados” (datos) como es el caso de Java.

A través del tiempo hemos visto que la programación orientada a código (llamada formalmente programación estructurada) es muy efectiva, y el ejemplo más claro es que C, después de más de cuarenta años, sigue siendo utilizado. Sin embargo, programar con este enfoque tiene desventajas, y la más importante de ellas es que hace muy complicado darle mantenimiento al código, pues si nuestro sistema crece en complejidad o hay que integrar nuevos requerimientos, los cambios en el código son muchos, lo que lo hace muy difícil de mantener.

Para hacer más fácil el mantenimiento del código y poder agregar nuevas funcionalidades a nuestro sistema sin tener que realizar tantos cambios en el código, surgió el otro paradigma, la programación basada en los datos, conocida formalmente como programación orientada a objetos.

La base de la programación orientada a objetos es la abstracción. Los seres humanos abstraemos todos los objetos que vemos y todo con lo que interactuamos. Por ejemplo, sabemos que un automóvil tiene motor, puertas y asientos; también

sabemos que un motor está compuesto de cilindros, pistones y bujías; y que a su vez un cilindro está hecho de metal y tornillos. Pero cuando pensamos en un automóvil no pensamos en todas las miles de partes que lo forman, sino que pensamos en él como una entidad que nos sirve para transportarnos, sin preocuparnos por sus componentes ni en cómo interactúan entre ellos. A esto se le llama clasificación jerárquica, y generalmente cuando pensamos en un objeto lo vemos desde la jerarquía más alta, sin embargo podemos ir bajando de jerarquía para analizar sus componentes.

De la misma forma, en la programación orientada a objetos generamos entidades llamadas objetos que cumplen una función y que a su vez se comunican con otros objetos para realizar tareas. Una vez que estos objetos están definidos, no nos preocuparemos más por cómo están hechos, de la misma manera que con un automóvil y demás objetos en la realidad.

La programación orientada a objetos tiene tres principios que deben cumplirse para lograr dicha abstracción y tener un código mantenible: encapsulación, herencia y polimorfismo.

La encapsulación es el mecanismo de unir el código y los datos que manipula, además de protegerlo para que no pueda ser accedido por fuentes externas y para que no se le dé un mal uso. Una forma de ver la encapsulación es como una envoltura que no permite que se vea el contenido y sólo permite que dicho objeto interactúe con el exterior cuando es necesario, controlando a qué funcionalidades o características del objeto se tiene acceso desde afuera. Esto nos permite abstraer objetos sin tener que preocuparnos por sus características o por cómo funcionan, simplemente tenemos un objeto y sabemos que si le damos cierta información, nos dará el resultado que necesitamos, tal y como sucede con un automóvil en el que mientras lo conducimos sabemos que si pisamos el acelerador, éste incrementará su velocidad, pero no nos preocupamos por saber cómo sucede este fenómeno.

La herencia es el proceso mediante el cual un objeto comparte características con otro objeto. Como mencioné anteriormente, la mayoría de nuestro aprendizaje está conceptualizado por medio de clasificaciones jerárquicas, por lo que esta forma de programar facilita mucho la codificación, ya que ésta se hace de la misma forma en la que abstraemos las ideas.

El diagrama mostrado a continuación, es un claro ejemplo de cómo conceptualizamos las cosas a través de jerarquías.

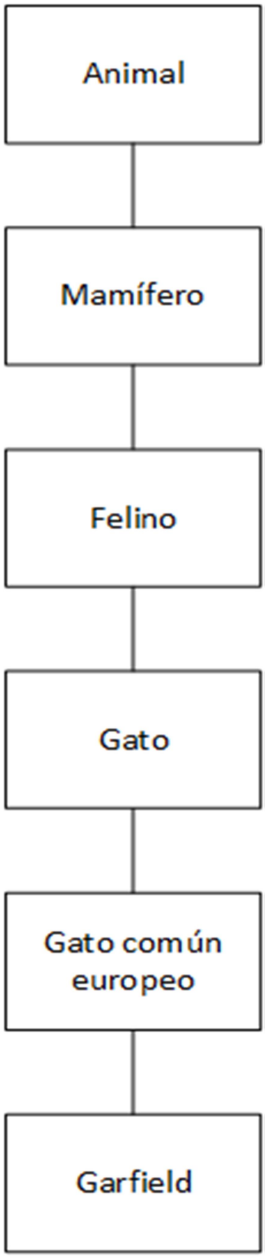

Figura 5: Ejemplo de jerarquías

En la parte superior de este diagrama se observa a un animal, el cual podríamos definir como un ser orgánico que tiene ciertas características propias y realiza ciertas acciones como vivir, alimentarse y reproducirse. Sin embargo, cuando mencionamos la palabra “animal”, no nos estamos refiriendo a algún animal en específico, sino que estamos hablando de un conjunto enorme de organismos, sin importarnos sus características particulares. Los mamíferos son un tipo de animal que además de tener las características de todos los animales (viven, se alimentan, se reproducen, etc), tiene características particulares que los diferencian de otros tipos de animales como los ovíparos, como tener glándulas mamarias. Si seguimos la jerarquía hacia abajo, vemos encontramos a los felinos, que son un tipo específico de mamíferos. Estos felinos comparten las todas las características de los mamíferos (tienen glándulas mamarias) y también comparten todas las características de los animales (viven, se alimentan y se reproducen). Si seguimos la jerarquía hacia abajo, vemos que cada vez se va haciendo más específico nuestro objeto y cada vez se reduce más el conjunto que lo conforma, hasta llegar a la base de la jerarquía, en este caso Garfield, el gato de John Bonachón, en donde ya se refiere a un animal en específico.

Como podemos observar en este ejemplo, es mucho más fácil abstraer ideas a través de la categorización jerárquica, y lo mismo pasa cuando programamos de esta forma. Si no existiera este paradigma y quisiera crear un objeto llamado Garfield y otro objeto llamado Don Gato, en cada una de estas entidades tendría que definir todas sus características y sus posibles acciones como vivir, alimentarse, amamantar, reproducirse, etc. Esto nos obligaría a repetir muchísimas líneas de código cada vez que quiera instanciar a un gato, lo que volvería muy difícil el mantenimiento ya que si quisiera hacer un cambio en alguna funcionalidad o característica de los gatos, tendría que realizar dicho cambio en todos los gatos que haya generado, lo que tomaría mucho tiempo y aumentaría las probabilidades de que haya algún error en alguno de los objetos modificados. En cambio, si nuestra clase gato hereda de otra clase todas las características que tiene en común con el resto de los gatos, cada vez que genere una instancia de un gato en particular, sólo tendré que preocuparme por programar las características particulares de ese gato, sabiendo que las características y funciones comunes a todos los gatos serán heredadas de las clases superiores en la jerarquía. Además mi código se volverá mucho más fácil de mantener porque no habrá código repetido y cuando sea

necesario hacer algún cambio en las características o funciones de los gatos, sólo tendré que realizar dicho cambio en una clase y todas las clases que hereden de la misma automáticamente heredarán ese cambio.

Finalmente, el polimorfismo es una característica que permite que una interfaz o una clase superior en la jerarquía, pueda ser utilizada como cualquiera de las clases que heredan de la misma, llamadas subclases. De esta forma, cuando nos referimos al gato Garfield, podemos referirnos a él como “el gato”, “el felino”, “el mamífero” o como “el animal”, pues sabemos que cumple con las características de todas estas entidades.

El polimorfismo es muy útil cuando programamos, pues nos permite que objetos que pertenecen a diferentes clases se comporten como su clase padre sin importar qué implementación específica tienen. Por ejemplo, supongamos que tenemos un zoológico y tenemos los siguientes objetos: gato, perro, león, cocodrilo, puma y chupacabras. Sin el polimorfismo, tendríamos que crear una función distinta para agregar cada uno de estos objetos al zoológico, pero gracias al polimorfismo podemos crear una función que agregue un objeto tipo Animal al zoológico, y como todos los objetos que quiero agregar son subclases de Animal, entonces podré realizar esta acción sin conflictos.

3.4 SCRUM

Para garantizar que un desarrollo de software se haga de forma ordenada y que cumpla con todos los requerimientos del cliente en tiempo y forma, se han creado diversas metodologías que permiten llevar un control respecto a los tiempos y el cumplimiento de requerimientos, además de que le facilitan al desarrollador cumplir con sus objetivos y al *project manager* la administración de los tiempos. La mayoría

de estas metodologías consisten en una serie de pasos ordenados que deben ser seguidos por el equipo de desarrollo para cumplir los objetivos propuestos. Sin embargo, existe otra categoría de metodologías llamadas metodologías ágiles, que abordan la planeación y desarrollo de software desde otro enfoque, centrándose en una serie de principios y en el factor humano (relación y colaboración entre el equipo de trabajo), además de explotar las lecciones aprendidas en desarrollos anteriores y la experiencia de los miembros del equipo más experimentados. Una de estas metodologías es la metodología *Scrum*[2].

Esta metodología se basa en un conjunto de valores, principios y prácticas que crea las bases para que una organización o un grupo de programadores generen una implementación única de prácticas de ingeniería. El resultado es una versión de *Scrum* única para el grupo de trabajo.

Una forma práctica de describir esta metodología, es compararla con un edificio. Imaginemos que la metodología *Scrum* forma los cimientos y las paredes del edificio. Los valores, principios y las prácticas de *Scrum* forman los componentes clave de la estructura. No es posible ignorar o cambiar completamente un valor, un principio o una práctica sin correr el riesgo de que el edificio colapse. Sin embargo, lo que sí es posible es personalizar o adecuar dentro de la estructura de *Scrum*, agregando características hasta que un proceso funcione para determinado equipo de trabajo.

Scrum es una metodología simple, centrada en los integrantes del equipo en lugar de centrarse en los requerimientos del proyecto, pues se basa en los valores de honestidad, respeto, concentración, confianza, esfuerzo, apertura y colaboración.

Las prácticas de esta metodología se manifiestan en artefactos, actividades y roles específicos, así como en sus respectivas reglas.

El desarrollo en la metodología *Scrum* requiere de uno o más equipos de trabajo. Cada uno de estos equipos puede tener un número indefinido de integrantes, pero éstos deben agruparse en tres roles distintos: el propietario del producto, el maestro

Scrum y el equipo de desarrollo. Pueden existir otros roles cuando se utilice *Scrum*, sin embargo *Scrum* sólo necesita los tres roles mencionados anteriormente.

El dueño del producto es el responsable de lo que se desarrollará, así como del orden en que se ejecutarán las tareas. Él es la única autoridad sobre el equipo y es quien decide qué funcionalidades o características se le agregarán al sistema. El dueño del producto debe mantener una comunicación constante con el resto del equipo para que queden claros los objetivos que se deben cumplir, además de que debe estar disponible para resolver dudas cuando éstas existan. Él es el responsable de que la solución que se está desarrollando cumpla con lo esperado y se desarrolle con éxito.

El maestro *Scrum* es el responsable de guiar al equipo para crear y seguir su propio proceso de desarrollo basado en los principios de *Scrum*. Este integrante del equipo le debe ayudar al resto de los integrantes para que entiendan los valores, las prácticas y los principios de la metodología. Otra de sus responsabilidades es apoyar al equipo para resolver sus dudas y a mejorar su uso en la metodología *Scrum*, además de ser el responsable de mantener el equipo unido y de resolver conflictos cuando se presentan. Aunque dicho integrante del equipo no tiene autoridad para ejercer control sobre los miembros del mismo, sólo debe guiarlos para que logren su objetivo. El maestro *Scrum* no cumple las funciones de un administrador, pero sí las de un líder.

El equipo de desarrollo tiene la tarea de determinar cómo entregar lo que el dueño del producto solicitó, así como de realizarlo. Normalmente un equipo de desarrollo está integrado por programadores, administradores de bases de datos, diseñadores de interfaces, *testers*, etc. Todas estas personas integran el rol del equipo de desarrollo dentro de la metodología *Scrum*.

Scrum define el equipo de desarrollo como una colección diversa y multifuncional de personas responsables del diseño, creación y pruebas del producto deseado. Este equipo de desarrollo debe organizarse para definir la mejor forma para alcanzar las metas propuestas por el dueño del producto.

Normalmente, el equipo de desarrollo está formado por entre cinco y nueve integrantes y sus miembros en conjunto deben tener todas las habilidades necesarias para cumplir con los requisitos del proyecto. Aunque se puede utilizar *Scrum* para equipos más grandes, se recomienda que si el número de desarrolladores está demasiado alejado de estas cifras, se formen dos o más grupos *Scrum* que cumplan con las características antes mencionadas.

Kenneth S. Rubin utiliza el diagrama de la figura 6 en su libro *Essential Scrum* para ilustrar las distintas actividades dentro de la metodología *Scrum* y cómo interactúan entre ellas.

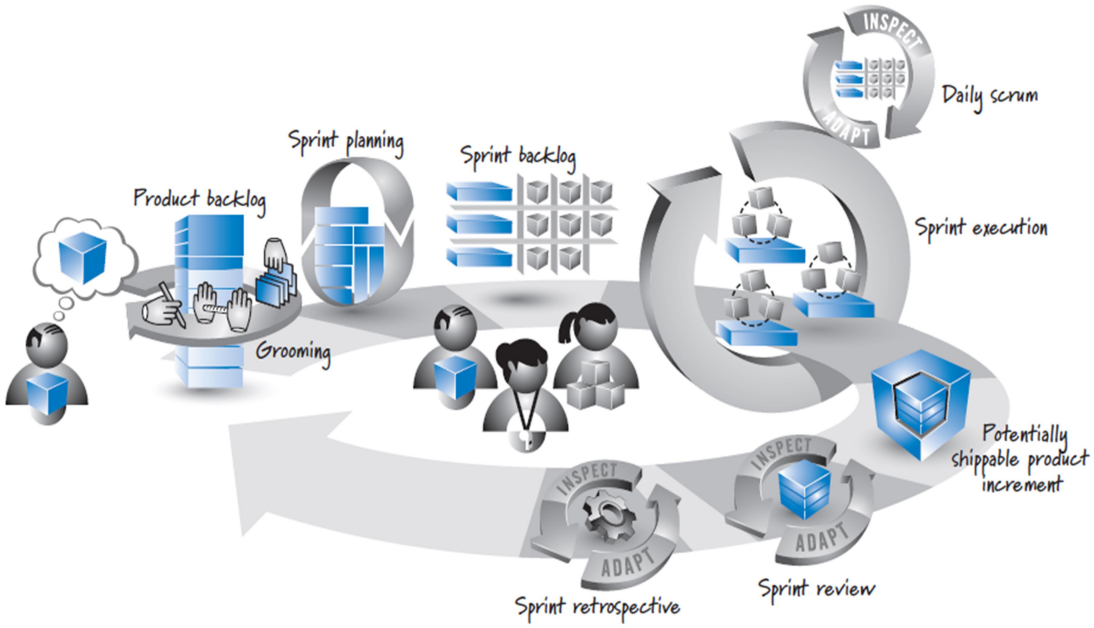

Figura 6: Actividades dentro de la metodología SCRUM

Las actividades de *Scrum* como se muestran en el diagrama, comienzan del lado izquierdo, siguiendo los procesos en el sentido de las manecillas del reloj hasta completar lo que se conoce como *sprint*, en un proceso cíclico de *sprints* hasta cumplir con el total de los requerimientos.

Primero, el dueño del producto define qué es lo que quiere crear (un gran cubo azul en el ejemplo de la imagen). Debido a que el cubo que quiere crear es demasiado

grande, éste es dividido en un conjunto de características que son establecidas en una lista de prioridades de pendientes del producto (*product backlog*).
Un *sprint* comienza con la planeación del *sprint*, que incluye el trabajo de desarrollo durante el *sprint* (llamado ejecución del *sprint*), y termina con su revisión. El *sprint* está representado por la flecha grande y circular que se encuentra en el centro de la figura. El número de elementos en la lista de pendientes del producto puede ser mayor a los que el equipo de desarrollo puede completar en un corto periodo de tiempo. Por este motivo, el equipo de desarrollo debe establecer un subconjunto de actividades de la lista de pendientes del producto al comienzo del *sprint*, de tal forma que pueda completar dicho subconjunto en un corto periodo de tiempo. Dicha actividad se llama planeación del *sprint* y está representada en la figura a la derecha del gran cubo de la lista de pendientes del producto.

Para asegurarse de que el equipo de desarrollo hizo un compromiso razonable, los miembros del equipo deben crear una segunda lista de pendientes durante la planeación del *sprint* llamada lista de pendientes del *sprint*. Esta lista describe, a través de un conjunto de tareas detalladas, cómo el equipo planea diseñar, construir, integrar y probar el subconjunto de características de la lista de pendientes del producto durante este *sprint* en particular.

El siguiente paso es la ejecución del *sprint*, durante la cual los integrantes del equipo de desarrollo realizan las tareas necesarias para desarrollar las características seleccionadas. Cada día, durante la ejecución del *sprint*, los miembros del equipo deben ayudar a administrar el flujo de trabajo al realizar una sincronización, inspección y una actividad de planeación adaptativa conocida como *scrum* diario. Al finalizar la ejecución del *sprint*, el equipo ha desarrollado un producto potencialmente entregable que representa una parte de la visión del dueño del producto.

El equipo de *Scrum* completa el *sprint* al realizar dos actividades para inspeccionar y adaptar el producto y la metodología. En la primera, llamada revisión del *sprint*, el equipo inspecciona el producto desarrollado. En la segunda, llamada retrospectiva del *sprint*, el equipo inspecciona el proceso *Scrum* que se utilizó para desarrollar el

producto. El resultado de estas actividades puede ser adaptaciones a la lista de pendientes del producto o al proceso de desarrollo del equipo.

Posteriormente, el ciclo *Scrum* se repite, comenzando de nuevo con el equipo de desarrollo decidiendo qué elementos de la lista de pendientes del producto son los más importantes para completarlos. Después de completar un número adecuado de ciclos de *sprint*, se cumplirán las expectativas del dueño del producto y la solución podrá ser liberada.

3.5 SPRING

Spring es un *framework* que ha ganado enorme popularidad en los últimos años debido a su flexibilidad y a la facilidad que le brinda a los desarrolladores para crear un flujo de comunicación entre el *front-end* y el *back-end*, con una estructura con capas bien definidas basadas en el modelo vista-controlador. Estas capas dividen la lógica de las clases para que cada clase cumpla con funciones específicas, dividiendo las distintas tareas para generar un código más entendible, mantenible y fácil de escalar. Las capas básicas en Spring son: vista, controladores, servicios y *DAOs* (explicadas en la sección de arquitectura del sistema, página 42), aunque el programador tiene libertad de agregar más capas para generar clases de propósito más específico.

Una de las principales ventajas de Spring es que, a diferencia de otros *frameworks*, no obliga al programador a hacer cambios en el código para que se adapte al *framework*, en lugar de eso, el *framework* es el que se adapta al código ya que por medio de los archivos de configuración se le indica qué paquetes mapear para que el *framework* los utilice en las distintas capas de la arquitectura.

Otra de las grandes ventajas de este *framework* es que está diseñado para que el código no esté altamente acoplado, lo que hace más flexible el código, más fácil de

probar y facilita la reutilización del mismo. Esto se logra a través de la inyección de dependencias, que explico a continuación:

Todas las aplicaciones están hechas de dos o más clases que colaboran entre ellas para realizar la lógica de negocios. Tradicionalmente, cada objeto es responsable de obtener sus propias referencias a los objetos con los que colabora (sus dependencias). A esto se le llama código altamente acoplado y es muy difícil de probar. Por ejemplo, considere la siguiente clase:

```
public class CaballeroRescataDamisela implements Caballero{
     private MisionRescateDamisela mision;

     public CaballeroRescataDamisela (){
          mision = new MisionRescateDamisela();
     }

     public void embarcarMision() throws MisionException{
          mision.embarcar();
     }
}
```

En este ejemplo podemos observar que la clase *CaballeroRescataDamisela* crea su propia misión, una *MisionRescataDamisela*, dentro del constructor. Esto hace que el *CaballeroRescataDamisela* esté altamente acoplado a una *MisionRescataDamisela* y limita el repertorio de misiones que puede tomar un caballero. Por lo que si un dragón ataca la aldea o existe otro peligro que el caballero debiera atender, no va a poder hacerlo porque la única misión que puede atender es la de rescatar a la damisela. Además, sería muy difícil probar esta clase porque a través de una prueba unitaria, pues el desarrollador querría asegurarse de que el método *embarcar()* sea llamado cuando el método *embarcarMisión()* sea llamado, pero no hay forma de lograr eso.

Con la inyección de dependencias, los objetos reciben sus dependencias durante su creación por una clase externa encargada de coordinar cada objeto en el sistema,

en lugar de crear sus dependencias desde la compilación. No se espera que los objetos creen y obtengan sus propias dependencias, sino que las dependencias son inyectadas en los objetos que las necesitan.

Para entender esto, revisemos la clase *CaballeroValiente*, que contiene a un caballero que puede embarcarse en cualquier misión que se proponga:

```
public class CaballeroValiente implements Caballero{
     private Mision mision;

     public CaballeroValiente(Mision mision){
          this.mision = new mision;
     }

     public void embarcarMision() throws MisionException{
          mision.embarcar();
     }
}
```

A diferencia de la clase *CaballeroRescataDamisela*, la clase *CaballeroValiente* no crea su propia misión, sino que la recibe cuando se crea la instancia de esta clase a través del constructor. Este es un tipo de inyección de dependencias conocido como *inyección a través del constructor*.

Esta característica es una de las principales de Spring, sin embargo tiene muchas otras características que resultan sumamente útiles, por las que este *framework* se ha convertido en el más utilizado en Java para desarrollar bajo el modelo vista-controlador. Para ver un análisis detallado del *framework* recomiendo el libro *Spring in Action* de Craig Walls[5].

CAPÍTULO 4
Antecedentes del tema

La televisión y la radio son dos medios masivos de comunicación que ejercen una enorme influencia en la opinión de las personas, debido en gran medida a que la información transmitida a través de ellos llega a millones de personas al mismo tiempo, siendo los medios de comunicación que más impacto tienen en la población. Jane Gregory y Steve Miller en su libro *Science in public*[1] mencionan que "incluso los museos científicos más importantes, como por ejemplo el Museo de Historia Natural de Londres, sólo pueden esperar tener tantos visitantes en todo un año como los que ven una única edición del programa semanal divulgativo Horizon (BBC) de televisión".

Debido al gran alcance que tienen dichos medios, las empresas y los partidos políticos han hecho uso de ellos para influir en la opinión de los espectadores para que compren cierto producto o voten por cierto candidato, teniendo resultados sumamente favorables.

En vista de estos hechos y de la competencia desleal que puede existir a través del uso de los medios masivos de comunicación por los partidos políticos para promocionarse, el Instituto Nacional Electoral publicó el Reglamento de Radio y Televisión en Materia Electoral[6], en el que establece las normas que deben cumplir los partidos y demás entidades políticas para anunciarse en medios masivos de comunicación sin perjudicar a otros actores.

A continuación menciono algunos de los artículos más relevantes en dicho reglamento:

- El artículo 7, numeral 3 establece que "El Instituto es la única autoridad competente para ordenar la transmisión de propaganda política o electoral en radio o televisión, para el cumplimiento de sus propios fines, de otras autoridades electorales federales o locales, de los partidos políticos y de los/las candidatos/as independientes de cualquier ámbito."

- En el mismo artículo 7, numeral 4 se establece que “los partidos políticos, precandidatos/as, candidatos/ as y aspirantes a cargos de elección popular, bien sean propuestos/as por partidos o de carácter independiente, en ningún momento podrán contratar o adquirir, por sí o por terceras personas, tiempos en cualquier modalidad de radio y televisión...”

- En el artículo 7, numeral 9 se establece que “la propaganda y mensajes que en el curso de las precampañas y campañas electorales difundan por radio y televisión los partidos políticos y en campañas los/las candidatos/as independientes, se ajustarán a lo dispuesto por el primer párrafo del artículo 6; y el artículo 41, Base III, Apartado C de la Constitución; así como el artículo 25, fracción I, inciso o) de la Ley de Partidos y 247 de la Ley.”

- El artículo 9, numeral 1 establece que “el tiempo del que dispongan los Partidos Políticos Nacionales y locales en las estaciones de radio y canales de televisión se distribuirá en forma igualitaria en mensajes con duración de 30 segundos cada uno”.

- El artículo 9, numeral 4 establece que “el horario de transmisión de los promocionales a que se refiere este Título será el comprendido entre las 6:00 y las 24:00 horas”.

- El artículo 10, numeral 3 establece que “los promocionales a que se refiere este Capítulo se transmitirán durante la hora que se establezca en la pauta, distribuidos en tres franjas horarias: la franja matutina, que comprende de las 06:00 a las 12:00 horas; la franja vespertina, de las 12:00 a las 18:00 horas; y la franja nocturna, de las 18:00 a las 24:00 horas”.

- El artículo 12, numeral 1 establece que “desde el inicio del periodo de precampaña electoral federal o local y hasta el día en que se celebre la

Jornada Electoral, el Instituto administrará 48 minutos diarios en cada estación de radio y canal de televisión que cubran la elección, que serán distribuidos en 2 y hasta 3 minutos por cada hora de transmisión, en cada estación de radio y canal de televisión, en el horario de programación referido en el artículo 166 de la Ley".

- Y finalmente, el artículo 15, numeral 1 establece que "el tiempo en radio y televisión que corresponda a los partidos políticos y, en su caso, coaliciones y candidatos/as independientes en su conjunto, convertido a número de promocionales, se distribuirá conforme al siguiente criterio: a) 30 por ciento del total en forma igualitaria, y 36 Instituto Nacional Electoral b) El 70 por ciento restante en proporción al porcentaje de votos obtenido por cada partido político en la elección federal o local de diputados/as de mayoría relativa, según sea el caso, inmediata anterior".

Se puede notar con facilidad que este reglamento tiene la finalidad de proporcionar equidad entre los partidos políticos y candidatos independientes para que todos puedan anunciarse en radio y televisión en las mismas condiciones. Pero, ¿cómo le hace el Instituto Nacional Electoral para validar que todos los partidos cumplan con las normas dispuestas en dicho reglamento?

Antes del 2008, no existía una forma confiable de monitorear las señales de radio y televisión en nuestro país, por lo que la entidad encargada de dicha responsabilidad le delegaba esta tarea a distintas empresas que utilizaban métodos rudimentarios (como pagarle a personas para que detecten los comerciales y midan el tiempo de los mismos a mano) para saber si los partidos y candidatos políticos cumplían con las leyes correspondientes. Por eso, en 2008 por primera vez el IFE (Instituto Federal Electoral) diseñó la Solución Integral de Verificación y Monitoreo de los Tiempos Oficiales en Materia Electoral (SIATE), la cual tenía como objetivo la adquisición del primer equipo para el monitoreo de las pautas de propaganda de los partidos políticos y candidatos independientes utilizadas en radio y televisión en tiempos electorales. La licitación para la adquisición de los equipos y el software del SIATE fue llevada a cabo en 2008.

Los equipos adquiridos por el IFE a la empresa Grupo de Tecnología Cibernética, empresa ganadora de la licitación en 2008 y en la que actualmente trabajo, conformaron la Sistema de Verificación y Monitoreo de los Tiempos Oficiales en Materia Electoral o Sistema de Verificación y Monitoreo (SIVEM), como es actualmente conocido por el INE.

Algunos de los equipos que conformaron el Sistema de Verificación y Monitoreo, puesto en marcha en 2009, fueron adquiridos considerando la vida útil de los equipos, de cinco años. Tomando en cuenta que una parte considerable de los equipos del SIVEM estarían obsoletos para el año 2015, el INE convocó a una licitación pública internacional para la adquisición de nuevos equipos y para la contratación de los servicios relacionados con su mantenimiento durante la prestación del servicio, así como para la renovación del software utilizado para realizar dichos monitoreos[10].

En Grupo Tecno trabajamos en el desarrollo de un sistema de monitoreo de señales para que las empresas que adquirieran publicidad en medios masivos pudieran corroborar que las televisoras y radiodifusoras contratadas para transmitir sus anuncios cumplieran con lo establecido en el contrato, pues la publicidad en dichos medios tiene un costo muy elevado. Es por esto que cuando fueron publicadas las bases para participar en la licitación del INE, decidimos sumar esfuerzos para adaptar nuestra solución a la que ellos solicitaron, realizando los cambios necesarios para poder participar y para finalmente salir ganadores, por lo que la herramienta que desarrollamos se convirtió en la herramienta de monitoreo de comerciales políticos a nivel nacional.

CAPÍTULO 5
Descripción del sistema

5.1 CARACTERÍSTICAS GENERALES

Fingercrab® es un sistema que Permite detectar los Spots (comerciales) que se transmiten a través de las señales de radio y TV. Para llevar a cabo dicho proceso el sistema consta de las siguientes fases:

1. Obtención de señales de Audio y Video.
   El sistema requiere de sintonizadores de radio o de TV, según sea el caso, los cuales se conectan al servidor de digitalización por medio de las tarjetas de captura. Este sistema permite la captura de un máximo de 24 señales de radio y TV, sin embargo el máximo de señales de TV es 12. Es decir, puede haber 24 señales de radio y ninguna de TV, o puede haber hasta 12 señales de TV y 12 de radio.

2. Digitalización y *Streaming* de la señal.
   Las señales analógicas son recibidos por las tarjetas de captura, las cuales se encarga de digitalizarlas y enviarlas a través de *streaming* en formato TS. A continuación se muestra un diagrama detallado sobre este proceso.

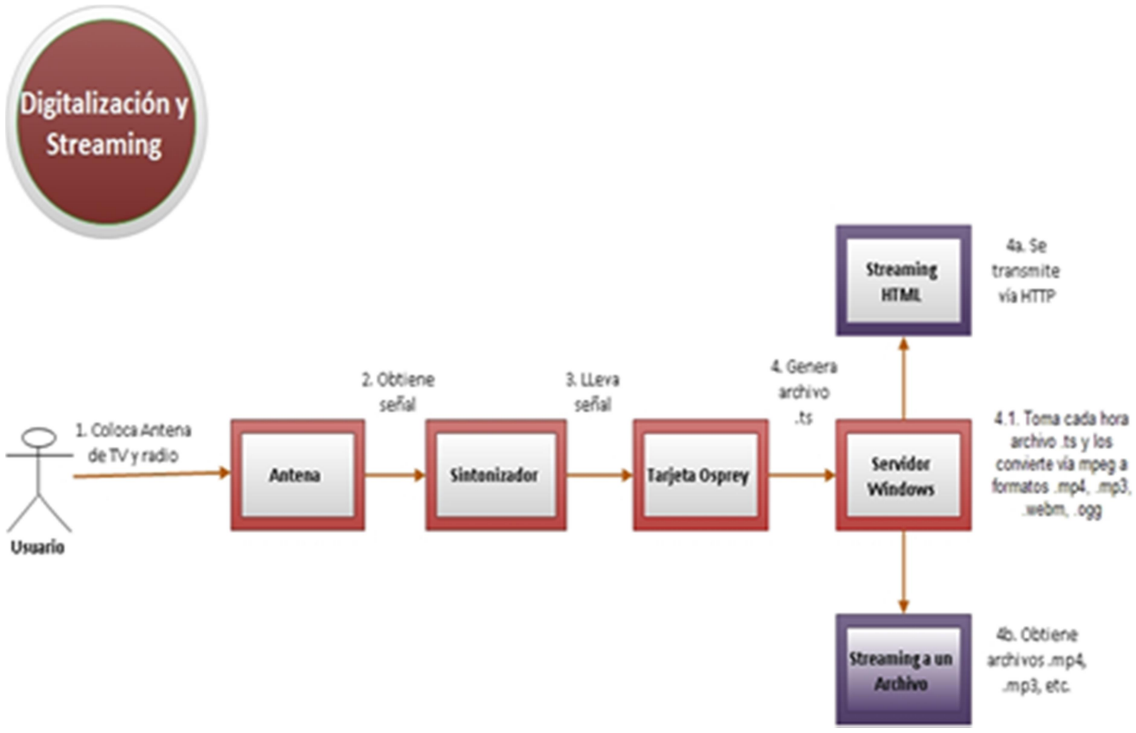

Figura 7: Proceso de digitalización.

3. Ingesta.

   La ingesta consiste en la carga de las huellas acústicas de los comerciales que se desean detectar. Como expliqué el marco teórico, los sistemas de detección de huellas acústicas requieren de una base de datos que contenga los audios que se deseen detectar para hacer la comparación al momento de enviar la solicitud de detección de una grabación desconocida. Esta ingesta se hace subiendo el audio o video del comercial, dejando la tarea de generar la huella acústica al sistema. El material subido puede estar en formato mp3, mp4, ogg o avi. A continuación se muestra un diagrama detallado sobre el flujo de este proceso:

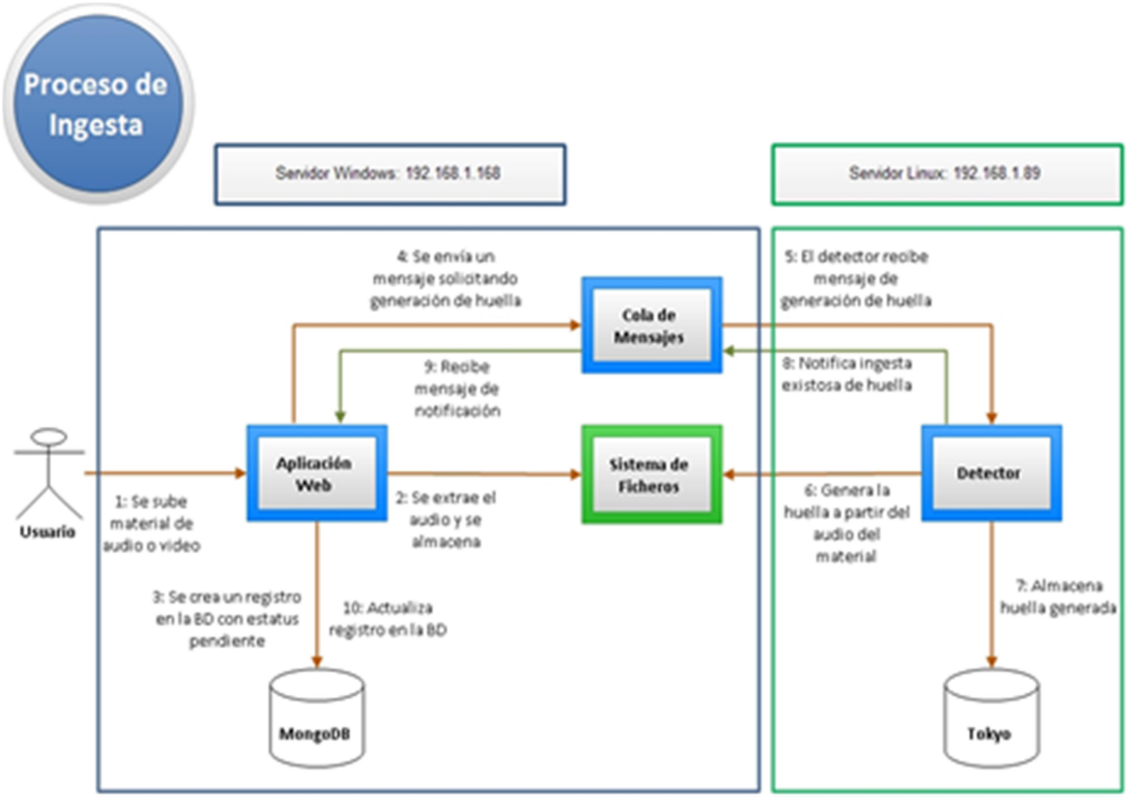

Figura 8: Proceso de Ingesta

4. Detección.

   La detección se hace enviando al detector fragmentos del streaming cada minuto, por lo que la herramienta tiene un desfase máximo de un minuto entre la transmisión en tiempo real y la detección de los comerciales. Para realizar la detección de comerciales se utilizó el algoritmo *echoprint*, descrito en el marco teórico. A continuación se muestra un diagrama detallado sobre el flujo de este proceso.

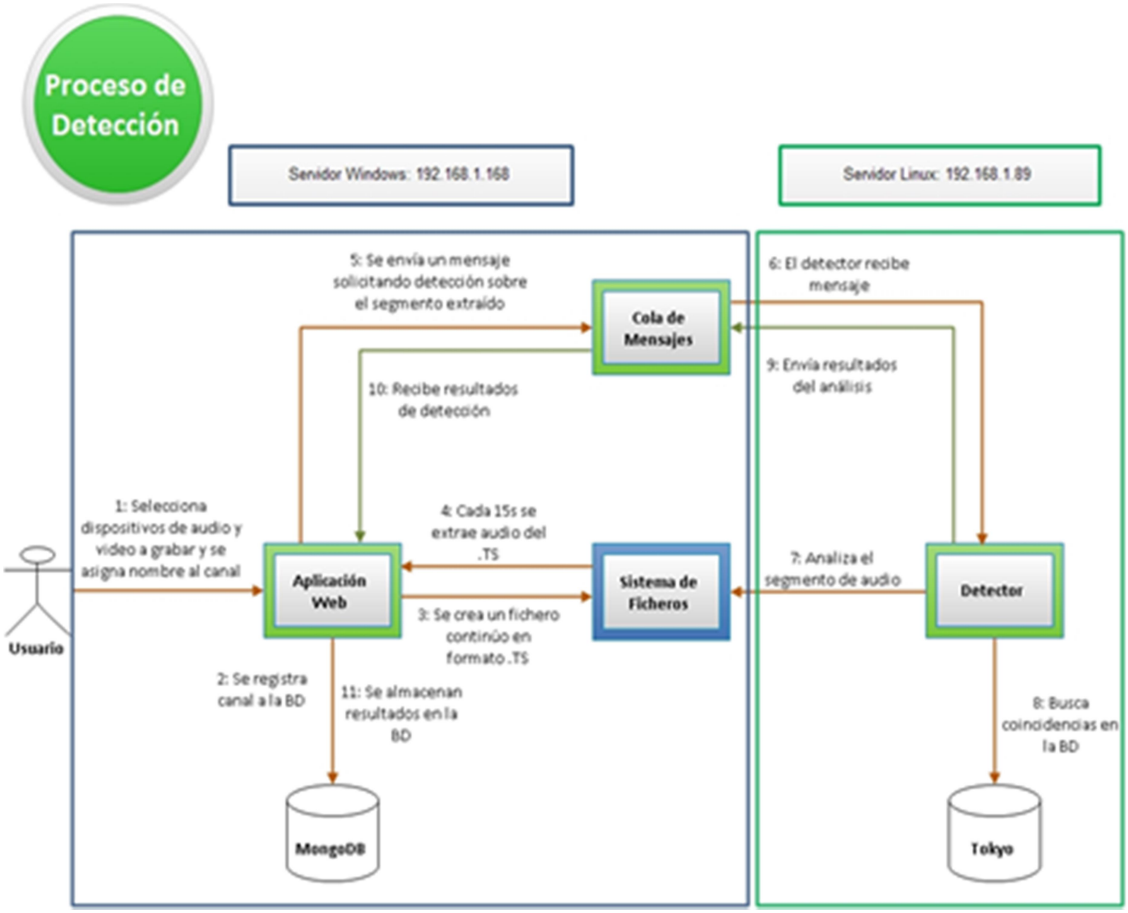

Figura 9: Proceso de detección

5. Generación de Testigos.

   Los testigos son el conjunto de grabaciones de las transmisiones del material analizado. Estos testigos se generan a partir del streaming enviado por la tarjeta de captura, el cual es capturado por el sistema, que genera un corte cada quince minutos y convierte este segmento a formato MP4 para almacenarlo. Los testigos están formados por un conjunto de videos de 15 minutos cada uno, los cuales contienen todo el material transmitido por las estaciones monitoreadas mientras el sistema estuvo en funcionamiento. A continuación se muestra un diagrama detallado sobre el flujo de este proceso.

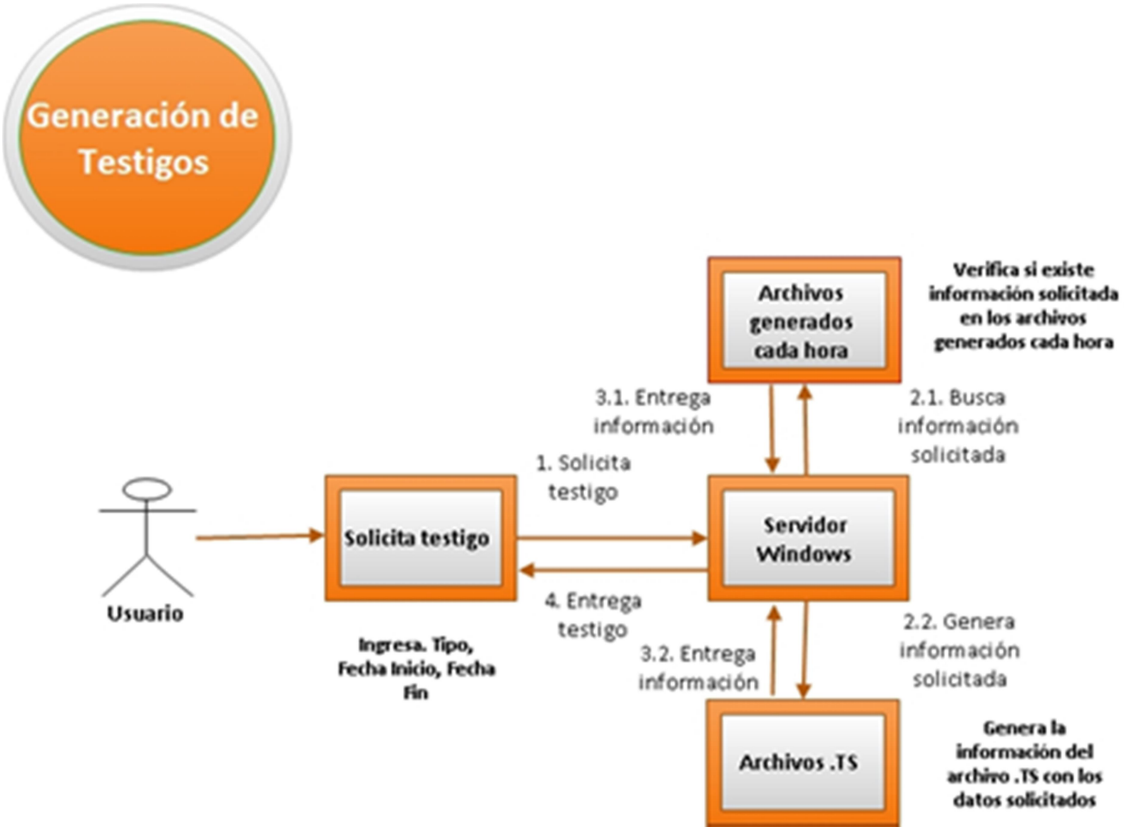

Figura 10: Proceso de generación de testigos

6. Servicios.

   Los servicios son un conjunto de funciones que permiten configurar el sistema y explotar la información del mismo a través de un API, por lo que el sistema puede integrarse a un sistema externo a través de dicho API.

5.2 ARQUITECTURA DE SOFTWARE

La arquitectura está conformada de la siguiente manera

- Cinco capas lineales
- N capas transversales como el desarrollo necesite (p.ej. Seguridad, Transaccionalidad, Logeo de errores, etc.)

En el diagrama mostrado a continuación se observa la arquitectura general de software del sistema. En este diagrama se observa que el sistema permite concurrencia de usuarios y que puede ser utilizado por todo tipo de dispositivos.
Las cinco capas lineales del desarrollo son:

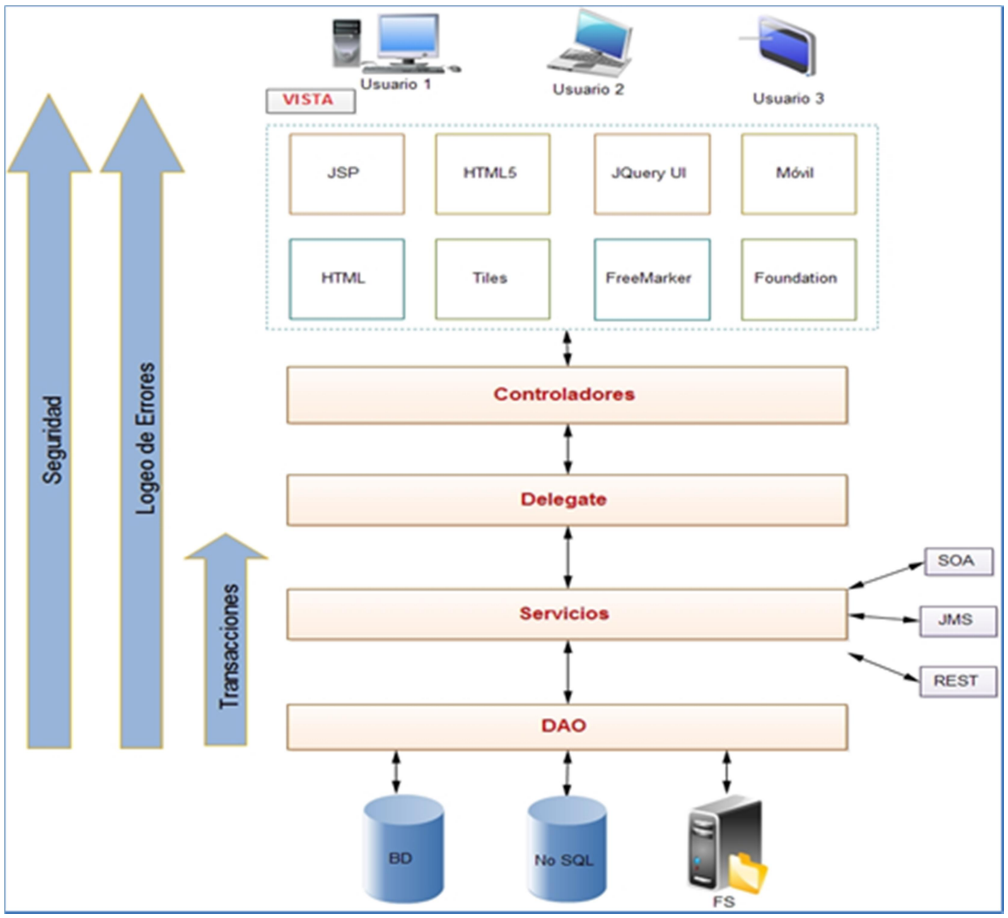

Figura 11: Arquitectura general de software del sistema.

1. Vista

   La capa de vista encapsula todos los elementos y lógica de presentación dirigida al usuario final, las interfaces de usuario pueden utilizar cualquier tecnología disponible en el mercado como JSP, Tiles, Freemarker, jQuery, etc. La selección de dichas tecnologías recae en el equipo de desarrollo. De igual manera debe implementar lógica relacionada con la presentación de

datos (p.ej. iteración de tablas, efectos visuales, habilitar y deshabilitar elementos, etc.). La presencia de elementos ajenos representa una violación al diseño original y en consecuencia no se garantiza el buen funcionamiento de la arquitectura.

2. Controladores

   La capa de controladores genera los elementos de paso entre la capa de vista y la capa Delegate (lógica de negocio) que se ejecuta sobre los datos que se reciben a través de las interfaces de usuario.

   Los controladores manejan el redireccionamiento hacia las vistas por medio de anotaciones o archivos XML de configuración, validan los datos ingresados por el(los) usuario(s) y manejan las excepciones arrojadas por la capa de Delegate (lógica de negocio) para mostrar los mensajes o vistas correspondientes.

   La etapa de validación de datos puede ocurrir en la capa de vista y/o en la capa de controladores, por lo que es importante mencionar que al realizar la validación en los controladores los datos deben viajar a través de la red consumiendo ancho de banda o generando costos, mientras que al validar en vista los datos no viajan hasta que cumplan las condiciones necesarias, sin embargo al estar del lado del cliente se pueden violar. Por lo mencionado anteriormente es decisión de los equipos de desarrollo implementar la validación de datos a nivel vista y/o controlador.

   El manejo de excepciones debe ser controlado por esta capa y poder llevar a cabo las acciones correspondientes para informar de los errores sin revelar información que represente un hueco de seguridad para el sistema.

3. Delegate

   Esta capa mantiene encapsulado el procesamiento de los datos que cumple con los requerimientos del sistema y usa las capas inferiores tales como Servicios y DAO con el fin de transmitir datos, persistirlos, almacenarlos, etc.

Esta capa se encarga también de traducir las excepciones de capas inferiores de tal forma que la capa superior (Controladores) reciba errores conocidos que faciliten la toma de decisiones. Es recomendable crear excepciones genéricas personalizadas que escalen la jerarquía.

4. Servicios

   Esta capa proporciona diferentes servicios que la capa superior Delegate puede utilizar para enviar datos a diferentes destinos, por ejemplo Web Services, peticiones vía REST, Base de Datos, Sistema de Ficheros, etc.

   Esta capa utiliza las instrucciones de bajo nivel que proporciona la capa DAO para realizar operaciones sobre Bases de Datos o Sistemas de Ficheros, por ejemplo, funciones como el insert, delete, update para Bases de Datos o remove, copy, create para el Sistema de Ficheros.

   Esta capa se considera transaccional, ya que en ella se pueden llevar a cabo varias operaciones de bajo nivel antes de llegar al resultado final (commit), si existe algún error dentro del proceso, se hará rollback y no se verá reflejado ningún cambio en la base de datos.

   La capa de servicios funciona también para conectar con servicios de mensajería como JMS, Web Services, SOA o REST. De igual manera nos permite exponer servicios en los formatos mencionados, los cuales pueden comunicar la solución con otras soluciones.

   Los servicios que se expongan desde esta capa sólo pueden comunicarse con las capas inferiores. Si se necesitan servicios que expongan lógica de negocio debe implementarse el delegate correspondiente.

5. DAO

   Esta capa implementa las operaciones de bajo nivel sobre Bases de Datos,

Sistemas de Ficheros o periféricos que requieren envío de información para almacenar o realizar otra tarea.

Es recomendable generar plantillas que eviten el reescribir métodos que realizan tareas idénticas y que permitan adicionar tareas específicas.

Las operaciones deben ser simples y sin lógica, únicamente extraen, consultan y eliminan datos, cualquier procedimiento ajeno se considera una violación de la arquitectura.

Una acción común que sirve para ejemplificar esta arquitectura es una búsqueda de detección en el sistema. Cuando el usuario desea buscar una detección, debe escribir los parámetros de su búsqueda y hacer click en el botón “buscar”. Este botón envía una solicitud al controlador correspondiente, que se encarga de interpretar dicha solicitud y enviársela al delegate como una llamada a un método de Java. El delegate recibe esta solicitud y decide qué método de qué servicio debe ser llamado para realizar dicha tarea. Una vez que llama dicho método en el servicio correspondiente, el servicio realiza toda la lógica necesaria para encontrar la detección solicitada, como validar que exista el canal, transformar los formatos de las fechas a un formato legible para Java, y demás procedimientos necesarios. Una vez que ha realizado los procedimientos necesarios, el servicio consulta la base de datos a través de un DAO, el cual realiza el query y regresa un conjunto de objetos que cumplan con las características que estableció el usuario. Una vez que se ha ejecutado el query, el DAO le regresa el resultado al servicio, quien a su vez se lo envía al delegate para que éste se lo regrese al controlador. El controlador se encarga de transformar dicho resultado a un formato que pueda interpretar el front-end y lo envía a la vista para que el usuario pueda ver la información.

Debido a la gran cantidad de información que almacena el sistema, decidimos utilizar una base de datos no relacional, lo que aceleró notablemente el desempeño del sistema. La base de datos utilizada fue MongoDB, misma que utilizan sistemas con grandes cantidades de datos y concurrencia de usuarios como foursquare.

Existen elementos auxiliares que nos permiten extender el uso de funciones genéricas hacia todas las capas o subconjunto de ellas. Estos elementos se conocen como *Helpers*, *Utils* y *DTOs*, descritos a continuación.

- Helpers

  Los *helpers* son elementos auxiliares que nos permiten generalizar funciones hacia un conjunto de clases bien identificadas, p.ej. funciones disponibles para varios servicios, *delegates* o controladores.

  Los *helpers* pueden ser estáticos o requerir instancia de uso. Los helpers estáticos no almacenan el estado de los procesos que están realizando, por lo que todos sus métodos son estáticos y no es necesario instanciarlo para usarlo.

  Los *helpers* que requieren instancia pueden almacenar su estado tras varios pasos de procesamiento.

- Utils

  La clase *Utils* contiene sólo constantes y funciones estáticas que sirven para todas las capas, en ella pueden implementarse métodos genéricos para formateo de fechas, listas inmutables, limpieza de cadenas, etc.

  Esta clase no debe requerir instancia, de lo contrario deberá implementarse un *Helper*.

- DTOs

  Los *DTOs* (Data Transfer Objects) son clases que nos permiten transportar datos desde la vista hasta la última capa de la arquitectura.

  En el paradigma MVC los *DTOs* son nuestro modelo y deben utilizarse para el paso de información entre capas. Los *DTOs* se crean a placer y necesidad del desarrollador.

5.3 ARQUITECTURA DE HARDWARE

El diagrama mostrado a continuación, se observan los elementos de hardware que utiliza el sistema y cómo interactúan entre ellos.

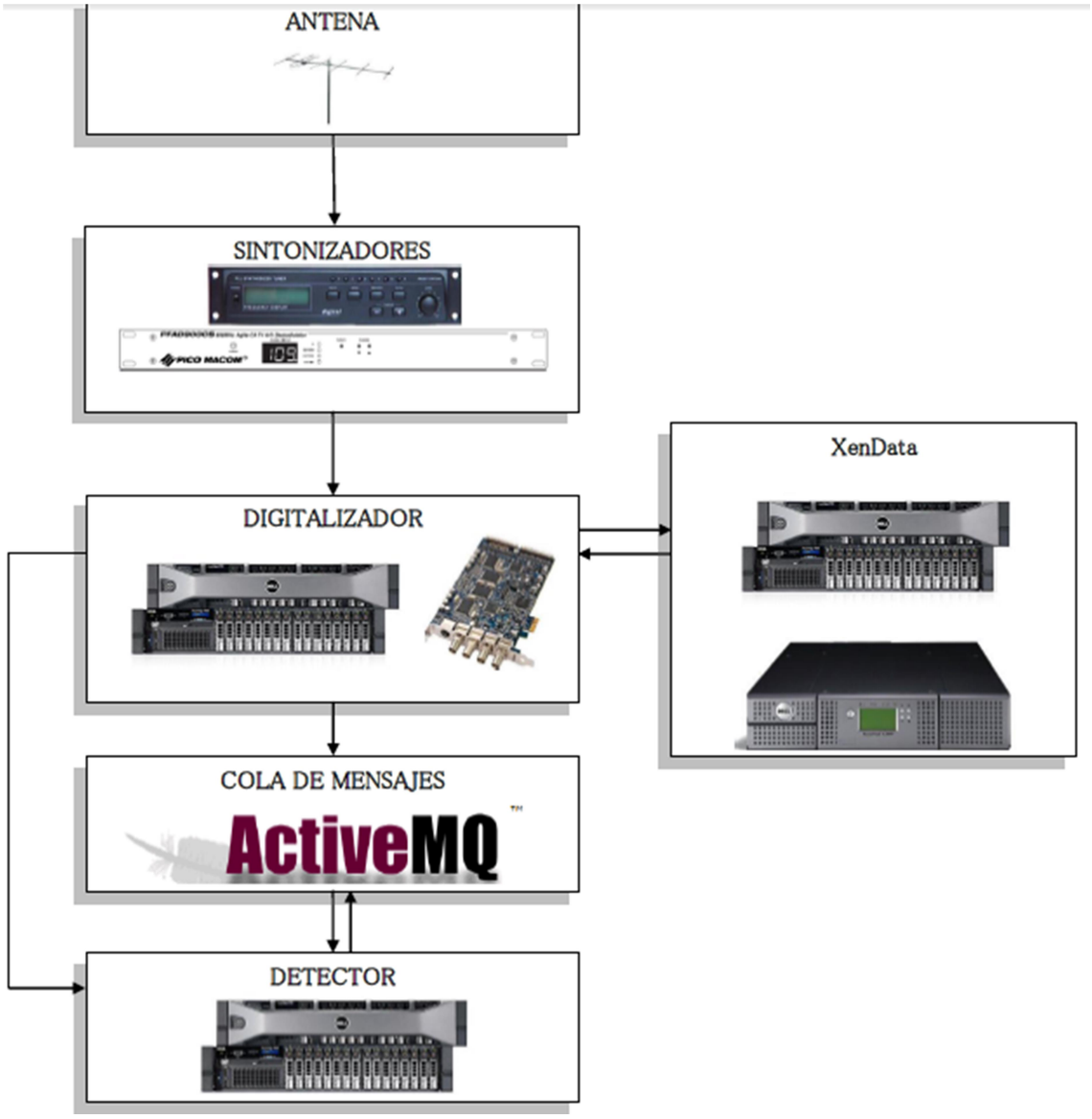

Figura 12: Arquitectura de hardware del sistema.

A continuación describo cada uno de sus elementos.

- Antena.
  Este elemento captura todas las señales de radio y TV a su alcance.

- Sintonizadores
  Los sintonizadores cumplen la función de indicar qué canal o estación va a ser transmitida al sistema. Debe haber un sintonizador por cada señal que se desee monitorear.

- Digitalizador
  El digitalizador es un servidor que tiene instalado el sistema FingerCrab® y realiza las siguientes funciones:

  Procesamiento de video
  1. Corte del *streaming* en formato TS cada 15 minutos para convertir este segmento en formato mp4 y almacenarlo.

  2. Corte y concatenación de videos (tanto los generados en formato mp4 como el *streaming* actual en formato TS) para descargar los testigos solicitados en el rango de fechas establecido.

  3. Conversión de archivo TS a mp3 para enviar a la cola de mensajes.

  4. Alojamiento de la aplicación web.

  5. Consola de administración de la aplicación web.

  6. Alojamiento de la cola de mensajes (*Broker*)

  7. Alojamiento de la base de datos de FingerCrab®

El digitalizador tiene instaladas las tarjetas de captura, las cuales reciben la señal analógica del sintonizador, la digitalizan y la envían a través de streaming al sistema para que éste la manipule.

El digitalizador se comunica con el detector a través de una cola de mensajes y con el servidor de almacenamiento. Para realizar la detección de anuncios de una señal, el digitalizador debe enviar el fragmento a analizar al detector para que éste realice la búsqueda y regrese el resultado de la misma al digitalizador, quien se encarga de procesar la información recibida y realizar las tareas correspondientes.

- Cola de mensajes

  La cola de mensajes es el medio de comunicación entre el digitalizador y el detector. Cuando el digitalizador envía un segmento a detección, debe enviar la solicitud a través de la cola de mensajes. El detector consume los mensajes de la cola, por lo que cuando consume la solicitud de detección, éste realizará dicha tarea sobre el segmento solicitado, regresando el resultado en la cola. Cuando el digitalizador consume el mensaje de la cola, lo interpreta y decide qué hacer con la información proporcionada por el detector.

- Detector

  El detector es el servidor encargado de realizar el análisis de las huellas (ver anexo 7) acústicas para identificar los comerciales a través del algoritmo *echoprint* (explicado en el marco teórico, página 13).

  Este servidor recibe la solicitud de detección del digitalizador a través de la cola de mensajes, realiza la búsqueda y regresa el resultado a través de la misma cola. También tiene la tarea de generar las huellas acústicas, por lo que cuando hay una ingesta de materiales (ver anexo 6), este servidor es el encargado de generar la huella acústica correspondiente y almacenarla para poder detectarla en un futuro.

- XenData

  Es el servidor de almacenamiento a largo plazo. Debido a que es necesario almacenar la evidencia de todas las transmisiones por mucho tiempo (lo que implica 24 horas de video diarias por canal monitoreado), es necesario utilizar un método de almacenamiento a largo plazo. Este servidor almacena la información en cintas magnéticas para que haya evidencia de lo transmitido en caso de que un partido político lo solicite.

CAPÍTULO 6
Contexto de la participación profesional

Este sistema fue desarrollado por un grupo de cinco personas: cuatro desarrolladores (incluyéndome) y un *tester*.
Para desarrollar un sistema tan complejo en un grupo tan reducido, fue necesario que cada uno de nosotros demostrara un gran conocimiento en programación, especialmente en Java, además de una gran capacidad de análisis y rápido aprendizaje, pues gran parte de la tecnología utilizada para desarrollar este sistema no había sido utilizada por ninguno de nosotros, por lo que tuvimos que aprender a utilizarla y dominarla en un periodo muy corto de tiempo.

Al tener un equipo de trabajo tan reducido, todos los integrantes participamos de alguna forma en todos los módulos, aunque cada uno de nosotros era responsable de ciertos módulos específicos. Yo fui responsable del desarrollo del módulo de digitalización, el de preferencias, el de almacenamiento a largo plazo y de la administración de la base de datos.

El módulo de digitalización consistió en todo lo referente a esta tarea, que abarca la conversión de video de formato TS a MP4, la generación de los cortes de 15 minutos para generar el historial y almacenarlo, la generación y descarga de testigos (evidencia de la transmisión) con la fecha y hora de inicio y la fecha y hora de fin indicadas por el usuario, y el encodeo de los testigos para descargarlos en el formato solicitado por el usuario.

El módulo de preferencias consistió en la configuración de la herramienta por el usuario para ajustar distintos aspectos como la resolución de video, el número de canales de audio, el bitrate, la dirección IP y las credenciales del servidor de almacenamiento a largo plazo, el tiempo que permanecen los materiales grabados en el servidor de digitalización antes de ser enviados al servidor de almacenamiento a largo plazo, entre otras.

El módulo de almacenamiento a largo plazo consistió en enviar los materiales que tuvieran un tiempo considerable en el servidor de digitalización, hacia el servidor de almacenamiento a largo plazo para ser almacenados en cintas magnéticas y archivados. El administrador del sistema, a través del módulo de preferencias, decide cuánto tiempo permanecen dichos materiales en el servidor de digitalización después de ser generados y antes de ser enviados al servidor de almacenamiento a largo plazo.

Finalmente, la administración de la base de datos fue una de las tareas más complejas que tuve que realizar. Durante mi trayectoria escolar llevé la materia de bases de datos, en la que aprendí el diseño de una base de datos relacional a través de diagramas UML, la creación de la misma, la diferencia entre los distintos lenguajes de bases de datos como son DDL (data definition language), DML (data manipulation language) y TCL (transaction control language)[7], las formas normales para el correcto diseño de las bases de datos y el lenguaje SQL para hacer consultas. Todos estos conceptos vistos en clase de base de datos fueron sobre bases de datos relacionales, sin embargo en este proyecto me enfrenté a un tipo de base de datos completamente nuevo: las bases de datos no relacionales.
A diferencia de las bases de datos relacionales, en las bases de datos no relacionales no existen llaves foráneas, por lo que no hay relación entre una tabla y otra sin tener que repetir información, así que en este tipo de bases de datos no se cumplen las cinco formas normales. Estas bases de datos tampoco utilizan lenguaje SQL para hacer consultas sobre las mismas, por lo que también tuve que aprender una forma de hacer consultas completamente nueva para mí. Este fue un enorme reto, pero al final fue muy satisfactorio porque aprendí una forma completamente nueva de administrar, almacenar y manipular datos en grandes cantidades.

Una vez que el sistema quedó listo para participar en la licitación, tres desarrolladores (incluyéndome) fuimos a presentar la prueba de concepto. Esta prueba tuvo una duración aproximada de doce horas, con veinte minutos de receso para comer, y en ella se realizaron pruebas a cada una de las funcionalidades del sistema para validar que cumplieran con lo establecido en la licitación. De las 80 características solicitadas y probadas, FingerCrab® cumplió con todas, motivo por el cual fuimos elegidos como el mejor sistema para realizar la función de monitoreo de

medios. Esta prueba representó un evento muy importante en mi carrera, pues a mis 24 años de edad estaba presentando un proyecto de una magnitud enorme, que sería utilizado en todo el país y tendría un gran impacto en las decisiones políticas de la nación, lo que significó una gran responsabilidad pero fue sumamente satisfactorio una vez que terminó la prueba y anunciaron al ganador de la licitación.

Después de resultar ganadores, comenzó la fase más compleja de todas: la fase de implementación. Durante esta fase tuvimos que instalar el sistema en 150 Centros de Verificación y Monitoreo (Cevems) en todos los estados de la república mexicana, monitoreando 1866 señales en total. A pesar de que para esta fase contrataron personal específicamente para instalar el sistema en todos estos puntos, el equipo de desarrollo fungió como responsable de dicha instalación y de proporcionar soporte en caso de que tuvieran algún problema durante la instalación, además de corregir los bugs que salieran. Esta tarea fue muy compleja y nos demandó mucho tiempo y esfuerzo, pero finalmente logramos instalar todos los centros en tiempo y forma, cumpliendo con lo establecido en el contrato y dejando instalado un sistema estable y completamente funcional.

CAPÍTULO 7
Análisis y metodología empleada

Para este proyecto decidimos utilizar una metodología ágil de desarrollo para hacerlo más rápido y fluido. La metodología que decidimos utilizar es la metodología *SCRUM*[2], ya que se adapta al equipo de trabajo y permite un desarrollo ágil y veloz con base en la experiencia y la interacción de los desarrolladores.

Mi equipo de desarrollo estaba compuesto por los siguientes integrantes:
Gustavo López: líder de proyecto,
Miguel García: líder de QA,
Rosa Vázquez: desarrollador Sr.,
Héctor Castillo: desarrollador Jr.,
Eugenio Kuri: desarrollador Jr,
Juan Carlos Macías: *project manager*.

A continuación mencionaré los roles que cada uno de los integrantes del equipo cumplió dentro de la metodología *SCRUM*:

Maestro *Scrum*: Gustavo López.
Gustavo fue el líder de proyecto y también fungió como maestro *Scrum*, pues él fue el que nos guió a través del desarrollo del proyecto para definir nuestros roles, nos ayudó a comprender esta metodología y nos enseñó a utilizarla para lograr un desarrollo ágil y que cumpliera con todos los requerimientos. Además, Gustavo cumplió su función de líder al mantener cohesión y un buen ambiente en el equipo, lo que facilitó el desarrollo haciéndolo más ameno y placentero.

Dueño del producto: Juan Carlos Macías.
Él, como *project manager*, fue el contacto entre el cliente y el resto del equipo. El éxito del proyecto estaba bajo su responsabilidad y siempre que surgía una duda por parte del resto del equipo, él era el responsable de aclararla, haciéndosela llegar

al cliente de forma inmediata para poder continuar con el desarrollo lo más rápido posible.
Dentro del equipo, Juan Carlos cumplió con las funciones del dueño del producto porque, a pesar de que él no era el cliente final, él fue el único contacto con el mismo, por lo que fue el que estableció los tiempos límites de entrega y las características que debía cumplir el sistema.

Equipo de desarrollo: Rosa Vázquez, Héctor Castillo, Miguel García y Eugenio Kuri.
Nosotros fuimos los responsables de entregar los módulos que el cliente nos solicitó, a través del dueño del producto. Los integrantes del equipo nos dividimos las distintas tareas que debíamos realizar, cada uno de nosotros eligió los módulos que más se adaptaba a nuestras capacidades y cada quien estableció una fecha de entrega para nuestros módulos basándonos en nuestro conocimiento y la complejidad de la tarea a desarrollar, siempre apegados a la fecha establecida como límite para entregar el proyecto. En caso de que se nos complicara algo durante nuestro desarrollo, el resto del equipo siempre estaba dispuesto a ayudar para solventar cualquier conflicto que surgiera y así poder entregar el producto en tiempo y forma.

Esta metodología la estudié en la materia de ingeniería de software, sin embargo fue hasta este proyecto que la puse en práctica y pude comprobar su utilidad, pues permite desarrollar de forma ágil y muy rápida, y si se utiliza correctamente, cada actividad lleva un orden y se adapta a las fechas de entrega establecidas. Además, la flexibilidad que ofrece para que cada integrante del equipo elija qué módulos va a desarrollar y su tiempo de entrega, me parece lo mejor de esta metodología porque cada quien decide de acuerdo a sus capacidades y a sus gustos, lo que resulta en un desarrollo mucho más ágil debido a que cada desarrollador se compromete más con su trabajo y desarrolla sus módulos mucho más rápido.

Para desarrollar FingerCrab® utilizamos la versión 1.7 de java sobre la versión 4 del *framework* de Spring, además de otras herramientas como ffmpeg para realizar el *encodeo* la manipulación de la media.

CAPÍTULO 8
Resultados

El resultado de este proyecto fue la renovación del sistema de verificación y monitoreo del Instituto Nacional Electoral. Gracias a este sistema, dicho instituto es capaz de monitorear 1866 señales de radio y televisión en los 32 estados del país, a través de 150 centros de monitoreo, cada uno de ellos con al menos un sistema FingerCrab® instalado.

Este sistema es el que valida que todos los partidos políticos y candidatos independientes cumplan con lo establecido en el Reglamento de Radio y Televisión en Materia Electoral[6], para que en caso de no hacerlo, el INE aplique la sanción correspondiente. La validación del cumplimiento de este reglamento es una de las tareas más importantes que realiza el INE, pues es a través de los medios de comunicación masivos que los partidos políticos consiguen influir a la mayoría de los ciudadanos que deciden darles su voto. A través de la regulación de la participación de los actores políticos en estos medios, el INE asegura unas elecciones limpias y equitativas para todos los candidatos.

Actualmente la herramienta tiene un nivel de detección superior al 90% a nivel nacional, facilitando la tarea de los monitoristas y del INE en general de una forma inigualable. Además, FingerCrab® graba todas las señales que monitorea para tener evidencia tajante ante cualquier falta de algún partido o ante cualquier solicitud para revisar que alguna entidad política esté realizando sus campañas en medios de comunicación apegada a las normas establecidas por el Instituto.

FingerCrab® es tan importante para la regulación de las campañas políticas por parte del INE y es tal la contribución que nosotros, como Grupo Tecno, hemos hecho al país, que puedo asegurar sin temor a equivocarme que sin un sistema como éste no habría forma de garantizar unas elecciones justas y equitativas para todos los candidatos políticos.

CAPÍTULO 9
Conclusiones

Este proyecto fue una gran experiencia para mí ya que considero que pocas personas tienen la oportunidad de participar en un proyecto de esta magnitud durante su trayectoria profesional, y menos son las que lo hacen a la edad en que yo lo hice. Durante este proyecto aprendí muchísimas cosas y reafirmé gran parte de lo que aprendí durante mis estudios.

Como cualquier proceso de aprendizaje, fue un proceso muy difícil, pues tener la responsabilidad de que funcione correctamente el sistema que monitorea las campañas políticas en todo el país no es tarea fácil. Como todo proyecto de *software*, en la etapa inicial tuvimos *bugs* que corregir, sin embargo a diferencia de otros proyectos, no teníamos margen de tiempo para corregirlos porque teníamos a las elecciones encima, así que tuvimos que estar prácticamente 24/7 pendientes al teléfono y metidos en el código para corregir cualquier bug en cuanto fuera detectado.

El primer centro de monitoreo instalado fue el de Cuernavaca debido a su cercanía con el Distrito Federal. Recuerdo que esa instalación fue crucial porque el INE nos condicionó a que ese centro funcionara a la perfección para continuar con las instalaciones en el resto del país. Creo que ese fue uno de los momentos más difíciles durante la instalación porque una vez instalado, nos reportaron caídas en el sistema, por lo que detuvieron las instalaciones hasta que resolviéramos el problema de Cuernavaca. Ahí nos percatamos de que es muy diferente tener un sistema en un ambiente de pruebas a tenerlo en un ambiente productivo, ya que nos percatamos de que la calidad de las señales afectaba severamente al sistema, efecto que no pudimos probar en nuestro ambiente de pruebas porque ninguna de las señales que monitoreamos en este ambiente tenía una calidad tan mala. Estuvimos tres días continuos en el sitio, prácticamente sin dormir, analizando el código, depurándolo y haciendo pruebas de hardware para dejar el sistema estable, hasta que finalmente dejamos el sistema funcionando adecuadamente. Esta

experiencia nos sirvió mucho para definir el procedimiento de instalación del resto de los sitios, además de que en cada uno de ellos obtuvimos experiencia para mejorar cada vez más nuestro proceso, y en consecuencia lo que fue sumamente complicado durante la primera mitad de las instalaciones, se volvió muy sencillo durante la segunda mitad. Durante este proceso aprendí que siempre hay que tratar de probar todos los escenarios posibles antes de salir a producción, pero aún así es prácticamente imposible abarcar todo el universo de posibles fallas en el sistema, por lo que hay que estar siempre preparado para corregir errores en tiempo real cuando el sistema ya esté en un ambiente productivo.

Los primeros tres meses de implementación estuvimos trabajando sin descanso para dejar la herramienta completamente estable, pero poco a poco fuimos tomando experiencia y detectando los errores comunes en la instalación que ocasionaban problemas en el sistema, además de afinar el código para que la herramienta funcionara cada vez mejor, y aunque fue un esfuerzo gigante, una presión que nunca había tenido, noches de desvelo y sin días de descanso, cuando finalmente logramos estabilizar la herramienta fue una satisfacción inigualable. Hoy en día me siento muy orgulloso de haber participado en este proyecto y de pertenecer a Grupo Tecno, ya que gracias a esto aprendí cosas que hubiera tardado años en aprender en otros lados, incluyendo metodologías ágiles y buenas prácticas de programación, configuración de servidores, instalar estos servidores en un rack, configuración de red, afinar el sistema operativo para mejorar el desempeño de una aplicación y toda la logística a la hora de implementar un proyecto de este tamaño. Además conocí a personas muy valiosas, aprendí que cuando un equipo de trabajo pasa tanto tiempo junto y comparte tanta responsabilidad, todos se vuelven una parte muy importante de nuestras vidas y se forman una familia, de lo contrario no sería posible terminar exitosamente el proyecto.

Uno de mis mayores aprendizajes no fue en el área técnica, sino en el área administrativa y consiste en tratar a los clientes, pues durante la implementación estuve trabajando directamente en las oficinas del INE, colaborando con el personal de la institución coordinando esfuerzos para que la herramienta quedara estable e instalada correctamente en todos los sitios, ya que a pesar de que nosotros éramos los responsables de la instalación, el INE era el responsable de manipular la

herramienta, y aunque suena fácil, la verdad es que es muy complicado tratar con los clientes. Recuerdo que en la clase de Ingeniería de Software, el Dr. Daniel Trejo nos comentó lo difícil que era el trato con los clientes, pero creo que es imposible realmente entender lo complicado que es hasta que se vive en carne propia, y más tratándose de un proyecto con un alcance y un costo como éste.

Bibliografía

1. JANE GREGORY Y MILLER S. (1998). *Science in Public Communication, Culture and Credibility*. Nueva York, EUA: Plenum Press.
2. RUBIN S. K. (2012). *Essential Scrum. A practical guide to the most popular agile process.* Michigan, EUA: Addison-Wesley.
3. PORTER A. (2012). *Evaluating musical fingerprinting systems.* Montreal, Canadá: McGill University.
4. SCHILDT H. (2014). *Java, The Complete Reference*. EUA: McGraw-Hill.
5. WALLS C. (2011). *Spring in Action.* EUA: Manning Publications Co.
6. http://norma.ine.mx/documents/27912/276864/2014_Reglamento_Radio_TV.pdf (enero 2016)
7. https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_1001.htm (enero 2016)
8. http://notes.variogr.am/post/27796385927 (enero 2016)
9. http://lucene.apache.org/solr (enero 2016)
10. http://ine.mx/archivos3/portal/historico/recursos/IFE-v2/DEA/DEA-Licitaciones/DEA-licitaciones-pdfs/2015/01_Enero/LP-INE-012-2014_tm.pdf (enero 2016)
11. https://s3.amazonaws.com/info-mongodb-com/10gen_Top_5_NoSQL_Considerations.pdf (enero 2016)