Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Burgos, junio de 2012

Resumen

Aquí va el resumen del proyecto

Descriptores

Detección de defectos, minería de datos, procesamiento digital de la imagen, visión artificial, aprendizaje automático, radiografías

Abstract

Lo mismo que el resumen, pero en inglés.

Keywords

Defect detection, Data Mining, digital image processing, computer vision, Machine Learning, radiography

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Aquí irán los agradecimientos.

1. OBJETIVOS DEL PROYECTO

Los objetivos principales del proyecto son principalmente cuatro:

1. Rediseñar completamente la aplicación que presentaron los alumnos del año pasado, buscando un diseño mucho más modular y reutilizable.
2. Refactorizar todo el código que reutilicemos, buscando un mejor estilo para favorecer su comprensión y reutilización.
3. Mejorar el rendimiento de la aplicación anterior, buscando que la misma se pueda aprovechar de los procesadores actuales de más de un núcleo de proceso.
4. Mejorar la precisión a la hora de localizar los defectos.
5. Ampliación de la funcionalidad de la aplicación anterior en diversos aspectos, como la implementación de algoritmos de segmentación que permitan obtener un resumen de las características geométricas de los defectos encontrados que, en un futuro, permitirán clasificar los defectos en varios y tipos y decidir si la pieza es defectuosa y debe ser retirada o si es correcta.

En el proyecto se van a implementar los algoritmos de selección de características más relevantes o significativos que hemos encontrado, prestando especial atención al diseño para que sea fácil la inclusión de nuevos algoritmos en el futuro. Esto posibilitará que el proyecto crezca en el tiempo y amplíe su capacidad.

1.1 Objetivos Técnicos

Este proyecto se va a desarrollar utilizando el paradigma de la *Orientacion a Objetos* el cual nos permitirá un diseño fácilmente comprensible por futuros desarrolladores que deseen proseguir con el presente trabajo.

El lenguaje de programación escogido para el proyecto es *Java 6*. El motivo de su elección ha sido el conocimiento del mismo, su sencillez, portabilidad, extensa documentación y la posibilidad de utilizar la librería de *Weka*. La completa *API (Application Programming Interface)* con la que cuenta y los conocimientos adquiridos durante la carrera sobre este lenguaje de programación harán posible que el desarrollo se base en el estudio e implementación de los algoritmos evitando retrasos por tener que aprender un nuevo lenguaje de programación

El lenguaje de modelado escogido ha sido *UML* (*Unified Modeling Language*) que se trata de un lenguaje unificado y muy extendido en el diseño de aplicaciones Orientadas a Objetos (OO).

Para la creación de los diagramas se utilizará *Jude*, se trata de una herramienta para el modelado de diagramas *UML*. Al estar enfocado a la OO posibilita todo tipo de diagramas de una forma cómoda y rápida. Todos los diagramas creados a lo largo del proyecto se realizarán con dicho programa.

Para resolver algunos de los problemas comunes a los que habrá que enfrentarse se utilizarán diversos patrones de diseño [**?**] estudiados que posibilitarán ofrecer una solución única estándar sobre problemas comunes que puedan surgir. La aplicación de patrones consigue diseños de calidad y, en consecuencia, mejores resultados en la fase de implementación.

También, trabajaremos con programación multihilo, buscando el poder aprovecharnos de las posibilidades que ofrecen los procesadores actuales, con más de un núcleo de proceso. Con esto, aumentaremos notablemente el rendimiento.

Para la memoria se va a utilizar LaTeX [**?**]. Las ventajas que ofrece respecto a otros sistemas de composición de textos (como los clásicos *WYSIWYG* [**?**]) son muchas, entre ellas nos permitirá la creación de una documentación uniforme, es decir, la salida que se obtenga será la misma con independencia del dispositivo o sistema operativo empleado para su visualización o impresión.

Dado que nunca se ha trabajado con LaTeX ha sido necesaria la utilización de documentación [**?**], manuales [**?**] y [**?**] que han facilitado el aprendizaje del mismo. Como plantilla para la memoria se utilizará la de la Universidad de Deusto [**?**] modificándola para adaptarla a la Universidad de Burgos.

Para trabajar con las radiografías se ha elegido la librería de ImageJ. Se trata de un programa de procesamiento de imagen digital desarrollado en Java que permite analizar y procesar imágenes, mediante la utilización de diversas técnicas como filtros e histogramas.

1.2 Objetivos personales

Además de los objetivos propios de la realización de la aplicación, también se pretende conseguir una serie de objetivos personales. Nos gustaría poder poner en práctica todos los conocimientos teóricos adquiridos durante estos años de carrera. Además, con la realización de este proyecto queremos adquirir nuevos conocimientos en áreas tan diversas como la gestión de proyectos y el uso de metodologías ágiles, la inteligencia artificial y la visión por computador, la minería de datos, el uso de sistemas de control de versiones, etc. Por último, queremos llevar a cabo con éxito la realización de este proyecto siendo capaces de planificarnos y trabajar en equipo.

2. CONCEPTOS TEÓRICOS

En este apartado se estudiarán de manera superficial los conceptos teóricos fundamentales para la correcta comprensión del proyecto. La visión por computador, también llamada visión artificial o visión técnica, es un subcampo de la inteligencia artificial. Su propósito es programar un computador para que pueda «entender» las características de una imagen. Entre sus objetivos se encuentra la detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (en nuestro caso serían defectos). Para lograrlo se utilizan diversas técnicas como el reconocimiento de patrones, aprendizaje estadístico, geometría de proyección, etc.

2.1 Adquisición de la imagen

La primera etapa del proceso es la adquisición de la imagen. Para ello se necesitarán dos elementos:

- Un sensor de imágenes, es decir, un dispositivo físico sensible a una determinada banda del espectro de energía electromagnética (como las bandas de rayos X, ultravioleta, visible o infrarrojo). En nuestro caso será un sistema de rayos X.
- Un digitalizador, dispositivo que permitirá convertir la señal de salida del sensor a forma digital.

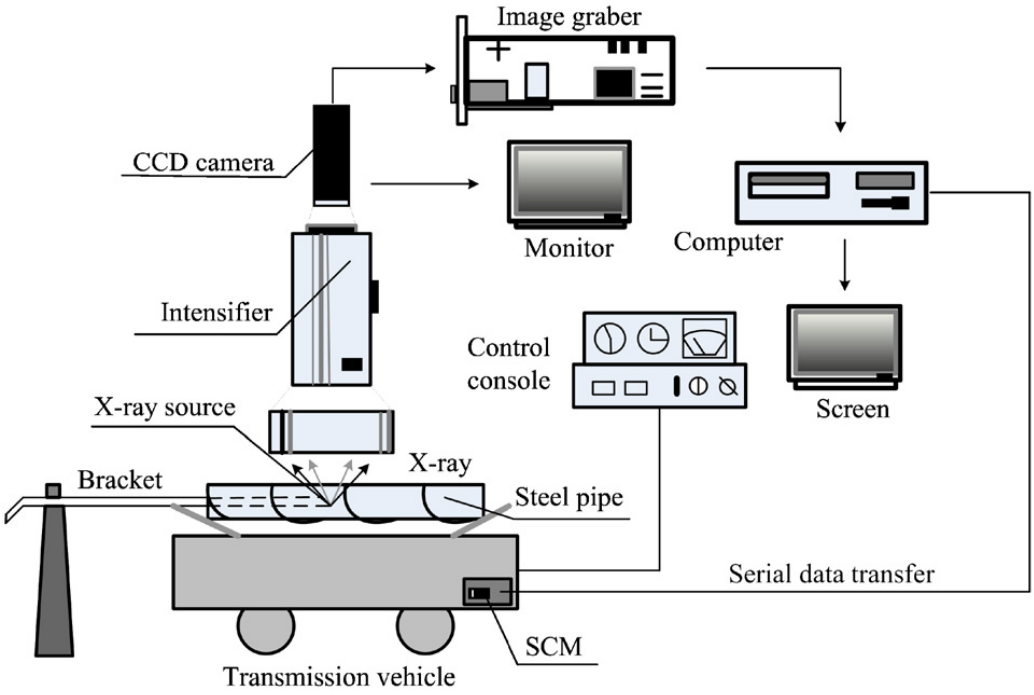

Figura 2.1.: Proceso de adquisición de imágenes de radiografía

2.2 Preprocesamiento

El preprocesamiento de la imagen es una etapa que consiste en reducir la información de la misma, de forma que pueda ser interpretada por una computadora, facilitando así la posterior fase de análisis.

Se utiliza un conjunto de técnicas que, aplicadas a las imágenes digitales, mejoran su calidad o facilitan la búsqueda de información. A partir de una imagen origen, se obtiene otra imagen final cuyo resultado sea más adecuado para una aplicación específica, optimizando ciertas características de la misma que hagan posible realizar operaciones de procesado sobre ella.

A continuación se explican algunas de las técnicas que comprenden esta etapa.

2.2.1 Binarización

La binarización de una imagen consiste en un proceso mediante el cual los valores de gris de una imagen quedan reducidos a dos: verdadero y falso. En una imagen digital, estos valores pueden representarse por los valores 0 y 1 o, más frecuentemente, por los colores negro (valor de gris 0) y blanco (valor de gris 255).

Para hacer esto, primero se debe convertir la imagen a escala de grises. Después hay que fijar un valor umbral entre 0 y 255. Una vez que se tenga dicho umbral, se convertirán a 255 todos los valores de la imagen superiores al umbral, mientras que los inferiores se convertirán a 0. El resultado será una imagen en blanco y negro que permitirá realizar tareas como la detección de contornos, separar regiones u objetos de interés del resto de la imagen, etc.

2.2.2 **Filtros de detección de bordes**

HAY QUE VER CUÁLES DE ESTAS METEMOS, SI METEMOS ALGUNA

2.2.3 **Realzado de la imagen**

El realzado de imágenes es una técnica de preprocesado cuyo objetivo principal es el de destacar los detalles finos de una imagen o intensificar detalles que han sido difuminados, bien sea por error o bien por efecto natural del método de adquisición de la imagen. De esta manera, se obtiene una imagen de salida que será más fácil de interpretar, haciendo la información relevante más visible.

Mediante el realzado se intenta acentuar las aristas de la imagen, obteniendo así una imagen con más contraste, es decir, con una mayor variabilidad entre los tonos de gris de sus diferentes píxeles. Con ello se consigue una mejora de la imagen, ya que los objetos aparecerán más resaltados, haciendo más fácil su diferenciación.

Las utilidades del realce de las imágenes son variadas e incluyen aplicaciones que van desde la impresión electrónica y las imágenes médicas hasta las inspecciones industriales e incluso la detección autónoma de objetivos en las armas inteligentes.

2.2.4 **Saliency**

El «Saliency Map» o «Mapa de Prominencia» [43] es un mapa topográfico que permite representar la prominencia visual de una determinada imagen.

Uno de los mayores problemas de la percepción es la sobrecarga de información. Se hace necesario identificar qué partes de la información disponible merecen ser seleccionadas para ser analizadas y qué partes deben descartarse. Este algoritmo busca solucionar este problema.

Koch y Ulman propusieron en 1985 [29] que las diferentes características visuales que contribuyen a la selección de atención ante un estímulo (color, orientación, movimiento, etc.) fueran combinadas en un único mapa topográfico, el Saliency Map, que integraría la información normalizada de los mapas de características individuales en una medida global de visibilidad.

La saliencia de una posición dada es determinada principalmente por cómo de diferente es dicha localización de las que la rodean, en color, orientación, movimiento, profundidad, etc.

La implementación del mapa de saliencia usada en este proyecto está basada en la variante descrita en el artículo Human Detection Using a Mobile Platform and Novel Features Derived From a Visual Saliency Mechanism [40].

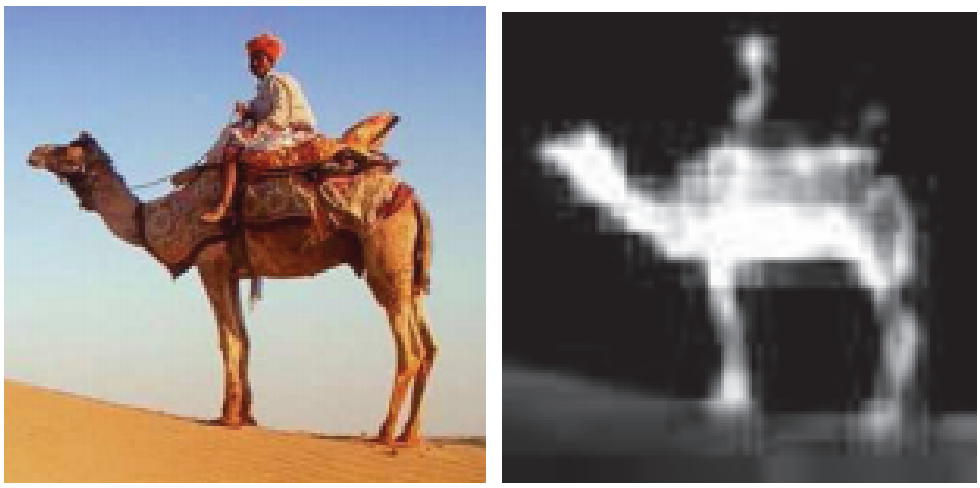

Figura 2.2.: Ejemplo de Saliency Map. La imagen original a la izquierda, con el correspondiente saliency map a la derecha

2.3 Descriptores de regiones

Una vez realizada la etapa de preprocesamiento, la imagen ya estará lista para ser analizada. Nosotros vamos a trabajar directamente con los píxeles de la imagen, a diferencia de otros métodos de análisis, que extraen otros tipos de atributos. Para analizar las características de la imagen, se utilizarán los siguientes descriptores:

2.3.1 Descriptores simples

También se les conoce como características estándar o de primer orden [46]. Son medidas que se calculan a partir de los valores de gris originales de la imagen y su frecuencia, como la media, varianza, desviación estándar, etc. En estas medidas no se considera la relación de co-ocurrencia entre los píxeles.

Las características más comunes y que se han usado en este proyecto son:

- **Media**

Se calcula el promedio de los niveles de intensidad de todos los píxeles de la imagen. Esta es una medida útil ya que nos permite determinar de forma sencilla la claridad de la imagen. Si la media es alta, la imagen será más clara, mientras que si la media es baja, será más oscura.

- **Desviación estándar**

Es una medida de dispersión que nos indica cuánto se alejan los valores respecto a la media. Nos sirve para apreciar el grado de variabilidad entre los valores de intensidad de los píxeles de una región.

■ Primera y segunda derivadas

Se utilizan operadores de detección de bordes [37] basados en aproximaciones de la primera y segunda derivada de los niveles de grises de la imagen. Ver sección 2.2.2 en la página 8. La primera derivada del perfil de gris será positiva en el borde de entrada de la transición entre una zona clara y otra oscura. En el borde de salida será negativa, mientras que en las zonas de nivel de gris constante será cero. El módulo de la primera derivada podrá utilizarse, por lo tanto, para detectar la presencia de un borde en una imagen. En cuanto a la segunda derivada, será positiva en la parte de la transición asociada con el lado oscuro del borde, negativa en la parte de la transición asociada con el lado claro y cero en las zonas de nivel de gris constante. El signo de la segunda derivada nos permitirá determinar si un píxel perteneciente a un borde está situado en el lado oscuro o claro del mismo.

2.3.2 Descriptores de textura

La texturas son propiedades asociadas a las superficies, como rugosidad, suavizado, granularidad, regularidad. En el campo de las imágenes, significa la repetición espacial de ciertos patrones sobre una superficie.

Otra definición de la textura podría ser la variación entre píxeles en una pequeña vecindad de una imagen. Alternativamente, la textura puede describirse también como un atributo que representa la distribución espacial de los niveles de intensidad en una región dada de una imagen digital.

El análisis de la textura de las imágenes nos ofrecerá datos útiles para nuestro trabajo. Hemos utilizado los siguientes descriptores:

■ Características de Haralick

Siguiendo la propuesta de Haralick [25], se extrae información de textura de la distribución de los valores de intensidad de los píxeles. Dichos valores se calculan utilizando matrices de coocurrencia que representan información de textura de segundo orden.

Haralick propuso un conjunto de 14 medidas de textura basada en la dependencia espacial de los tonos de grises. Esas dependencias están especificadas en la matriz de co-ocurrencia espacial (o de niveles de gris). La forma de calcular dicha matriz está definida en el siguiente artículo [46].

La matriz de co-ocurrencia, una vez normalizada, tiene las siguientes propiedades:

- Es cuadrada.
- Tiene el mismo número de filas y columnas que el número de bits de la imagen. Con una imagen de 8 bits ($2^8 = 256$ posibles valores) la matriz de co-ocurrencia será de 256x256, es decir, 65536 celdas.
- Es simétrica con respecto a la diagonal.
- Los elementos de la diagonal representan pares de píxeles que no tienen diferencias en su nivel de gris. Si estos elementos tienen probabilidades grandes, entonces la imagen no muestra mucho contraste, ya que la mayoría de los píxeles son idénticos a sus vecinos.
- Sumando los valores de la diagonal tenemos la probabilidad de que un píxel tenga el mismo nivel de gris que su vecino.

- Las líneas paralelas a la diagonal separadas por una celda, representan los pares de píxeles con una diferencia de un nivel de gris. De la misma manera sumando los elementos separados dos celdas de la diagonal, tenemos los pares de píxeles con dos valores de grises de diferencia. A medida que nos alejamos de la diagonal la diferencia entre niveles de grises es mayor.
- Sumando los valores de estas diagonales secundarias (y paralelas a la diagonal principal) obtenemos la probabilidad de que un píxel tenga 1, 2, 3, etc niveles de grises de diferencia con su vecino.

Una vez construida la matriz de co-ocurrencia, de ella pueden derivarse diferentes medidas. Se obtendrán matrices para las direcciones 0^o, 90^o, 180^o y 270^o y para distancias 1, 2, 3, 4 y 5. Para cada una de estas distancias se calculará un vector con las medias de las cuatro direcciones y otro con los rangos. Las características a calcular a partir de la matriz son las siguientes:

1. **Segundo Momento Angular**

   Mide la homogeneidad local. Cuanto más suave es la textura, mayor valor toma. Si la matriz de co-ocurrencia tiene pocas entradas de gran magnitud, toma valores altos. Es baja cuando todas las entradas son similares [2].

   $$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2$$

   $p(i,j)$ es el valor de la matriz de coocurrencia en la fila i y la columna j N_g es la dimensión de la matriz

2. **Contraste**

   Es lo opuesto a la homogeneidad, es decir, es una medida de la variación local en una imagen. Tiene un valor alto cuando la región dentro de la ventana tiene un alto contraste.

   $$f_2 = \sum_{n=0}^{N_g-1} \sum_{j=1}^{N_g} p(i,j)$$

3. **Correlación**

   Mide las dependencias lineales de los niveles de grises, la similitud entre píxeles vecinos. Un objeto tiene mayor correlación dentro de él que con los objetos adyacentes. Píxeles cercanos están más correlacionados entre sí que los píxeles más distantes.

   $$f_3 = \frac{\sum_i \sum_j (i,j) \cdot p(i,j) - v_x v_y}{\sigma_x \sigma_y}$$

   Donde $v_x, v_y, \sigma_x, \sigma_y$ son las medias y desviaciones estándar de p_x y p_y, las funciones de densidad de probabilidad parcial.

4. **Suma de cuadrados**

   Es la medida del contraste del nivel de gris.

$$f_4 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} (i-j)^2 \cdot p(i,j)$$

5. **Momento Diferencial Inverso**

   También llamado homogeneidad, es más alto cuando la matriz de co-ocurrencia se concentra a lo largo de la diagonal. Esto ocurre cuando la imagen es localmente homogénea de acuerdo al tamaño de la ventana.

$$f_5 = \sum_{i} \sum_{j} \frac{1}{1+(i-j)^2} \cdot p(i,j)$$

6. **Suma promedio**

$$f_6 = \sum_{i=2}^{2N_g} i \cdot p_{x+y}(i)$$

7. **Suma de Entropías**

$$f_7 = \sum_{i=2}^{2N_g} (i - f_8)^2 \cdot p_{x+y}(i)$$

8. **Suma de Varianzas**

$$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log(p_{x+y}(i))$$

9. **Entropía**

   Es alta cuando los elementos de la matriz de co-ocurrencia tienen valores relativamente iguales. Es baja cuando los elementos son cercanos a 0 ó 1.

$$f_9 = -\sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \log(p(i,j))$$

10. **Diferencia de Varianzas**

$$f_{10} = \sum_{i=0}^{N_g-1} i^2 p_{x-y}(i)$$

11. **Diferencia de Entropías**

$$f_{11} = -\sum_{i=0}^{N_g-1} p_{x-y}(i)\log(p_{x-y}(i))$$

12. **Medidas de Información de Correlación 1**

$$f_{12} = \frac{HXY - HXY1}{\text{máx}(HX, HY)}$$

Donde:

$$HXY = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)\log(p(i,j))$$

$$HXY1 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p(i,j)\log(p_x(i)p_y(j))$$

$$HXY2 = -\sum_{i=1}^{N_g}\sum_{j=1}^{N_g} p_x(i)p_y(j)\log(p_x(i)p_y(j))$$

13. **Medidas de Información de Correlación 2**

$$f_{13} = (1 - \exp(-2|HXY2 - HXY|))^{1/2}$$

14. **Coeficiente de Correlación Máxima**

$$f_{14} = \sqrt{\lambda_2}$$

Donde λ_2 es el segundo valor propio de la matriz Q definida como:

$$Q(i,j) = \sum_k \frac{p(i,k)p(j,k)}{p_x(i)p_y(i)}$$

■ Local Binary Patterns

Los Local Binary Patterns (LBP) [59] son un tipo de característica usado para la clasificación de texturas. Fueron descritos por primera vez en 1994 [52].

Debido a su poder de discriminación y su simplicidad de cálculo, se ha convertido en un método popular que se usa en varios tipos de aplicaciones [16].

Este operador de textura etiqueta los píxeles de una imagen comparando los valores de intensidad de los píxeles de una vecindad de 3x3 con el del píxel central.

Cuando el valor del píxel vecino es mayor que el del píxel central, se escribe «1». En caso contrario, se escribe «0». El resultado es un número binario de 8 dígitos, que suele convertirse a

decimal por comodidad o para mayor facilidad de cálculo. Este número recibe también el nombre de «patrón».

Luego se realiza un histograma que contendrá la frecuencia con la que se ha producido cada patrón. Dicho histograma podrá utilizarse como descriptor de textura.

Posteriormente se ha extendido el uso de diferentes tamaños, no sólo a ocho puntos, sino a muestreos circulares donde la bilinealidad se consigue con la interpolación de los valores de los píxeles, lo que permite utilizar cualquier radio y por lo tanto cualquier número de píxeles vecinos.

Para reducir la longitud del vector de características se utilizan los patrones uniformes. Un local binary pattern es uniforme si el patrón contiene un máximo de dos transiciones a nivel de bit, de «0» a «1» o viceversa.

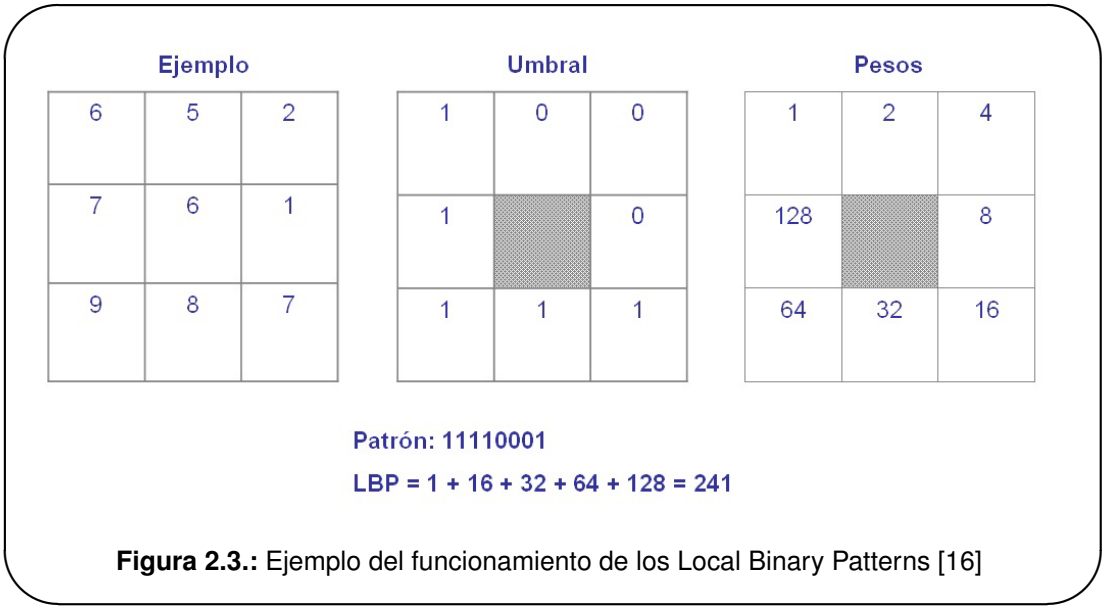

Figura 2.3.: Ejemplo del funcionamiento de los Local Binary Patterns [16]

Por ejemplo, los patrones 00000000 (0 transiciones), 01110000 (2 transiciones) y 11001111 (2 transiciones) son uniformes mientras que los patrones 11001001 (4 transiciones) y 01010010 (6 transiciones) no lo son. El número de transiciones se guarda en un valor llamado medida de uniformidad U.

En el cálculo de los LBP, se utiliza una etiqueta para cada uno de los patrones uniformes, mientras que todos los patrones no uniformes son agrupados en una sola etiqueta. Por ejemplo, cuando se usa una vecindad (8,R)(donde R es el radio), hay un total de 256 patrones, 58 de los cuales son uniformes, lo cual produce un total de 59 etiquetas diferentes. Todo esto dará como resultado un histograma con 59 intervalos.

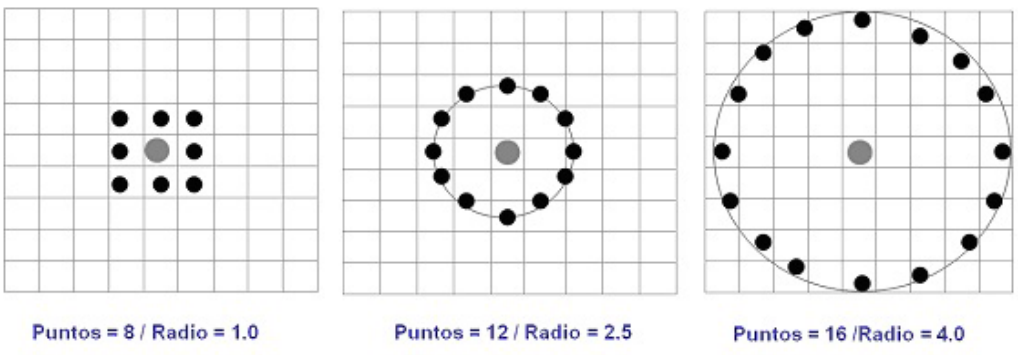

Figura 2.4.: Vecindades de distinto tamaño en los LBP [16]

2.4 Segmentación

Además de calcular ciertas características de las imágenes, como hemos visto en el apartado anterior, se hace necesaria realizar una segmentación de la imagen, para aumentar la precisión de la detección de los defectos.

La segmentación de una imagen consiste en particionar esa imagen en múltiples segmentos (conjuntos de píxeles). El objetivo es simplificar y/o cambiar la representación de una imagen en algo que sea más significativo o fácil de analizar. Se suele usar típicamente para localizar objetos y bordes. Más precisamente, la segmentación de una imagen es el proceso de asignar una etiqueta a cada píxel de una imagen, con lo que los píxeles que tengan la misma etiqueta compartirán ciertas características.

De entre todos los posibles métodos de segmentación que existen, nosotros hemos usado el denominado *Thresholding*. Es uno de los métodos más simples. Está basado en considerar un valor, llamado **umbral**, que se usa para convertir una imagen en escala de grises a una binaria. La clave está, por tanto, en el valor umbral. Hay varias opciones:

- Basados en la forma del histograma.
- Basados en clustering.
- Basados en entropía.
- Basados en atributos de objetos.
- Métodos espaciales.
- Métodos locales.

De todos estos, nosotros hemos usado los locales. Estos métodos se basan en adaptar el valor umbral en cada píxel, de acuerdo a las características del vecindario de ese píxel. Este vecindario viene dado por un radio, que se puede cambiar.

Hay varios de estos métodos. A continuación, vamos a describir los que hemos considerado.

2.4.1 MidGrey

Este método selecciona el umbral de acuerdo a la media del máximo y mínimo valor de la distribución local de escala de grises.

Por lo tanto, el umbral viene dado por la siguiente fórmula:

$$T = (\frac{\text{máx} + \text{mín}}{2}) - c$$

Donde c es una constante que sirve para afinar el método. Por defecto, es cero.

Para determinar a qué región pertenece el píxel, se comprueba con el umbral. Si es mayor que éste último, el píxel pertenece al objeto. Si no, pertenece al fondo.

2.4.2 Mean

En este caso, el umbral es la media de la distribución local en escala de grises. Por lo tanto, para determinar a qué región pertenece el píxel, se compara con la media de la región de vecinos (menos el parámetro c, en caso de que se especifique). Si es mayor, es parte del objeto. Si no, es parte del fondo.

2.5 Características geométricas

Además de los descriptores de regiones que ya hemos visto, pensamos en la posibilidad de realizar algunos cálculos de características geométricas sobre las regiones segmentadas, con la mirada puesta en poder clasificar mediante ellas a los defectos en distintos tipos.

Las características geométricas que hemos calculado son:

- **Área:** es la superficie de la región de interés medida en píxeles cuadrados.
- **Perímetro:** es la longitud del límite exterior de la región.
- **Circularidad:** descriptor de forma dado por la fórmula $\frac{4\pi \times area}{perimetro^2}$. Un valor de 1 indica que se trata de un círculo perfecto. Según se acerca a cero, indica que la forma es cada vez más alargada.
- **Redondez:** descriptor de forma dado por la fórmula $\frac{4\pi \times area}{\pi \times semieje\_mayor^2}$. Es el inverso del cociente entre el semieje mayor y el menor de la mejor elipse que puede ser dibujada en la región.
- **Semieje mayor:** semieje mayor de la mejor elipse que puede ser dibujada en la región.
- **Semieje menor:** semieje menor de la mejor elipse que puede ser dibujada en la región.
- **Ángulo:** es el ángulo entre el semieje mayor y una línea paralela al eje x.
- **Distancia Feret:** también llamada diámetro de Feret. Es la máxima distancia entre dos puntos cualesquiera de la región.

2.6 Reconocimiento e interpretación de imágenes

Una vez obtenidas las características de la imagen, el siguiente paso será reconocer dichos datos e interpretarlos. Para ello será necesario entrenar un clasificador. Una vez entrenado, podrá predecir dónde estarán los defectos que buscamos.

Un clasificador [58] es un elemento que, tomando un conjunto de características como entrada, proporciona a la salida una clase etiquetada. En nuestro caso la clase sería el «defecto», que podría tomar dos valores: «verdadero» o «falso». En el caso de que sea una regresión, en vez de una clasificación de clases nominales, los valores que devolverá el clasificador serán 0 y 1.

Utilizaremos el clasificador por su capacidad de aprender a partir de imágenes de ejemplo y de generalizar este conocimiento para que se pueda aplicar a nuevas y diferentes imágenes.

Para construir el clasificador utilizaremos un conjunto de imágenes etiquetadas. Para etiquetarlas, se creará una máscara de cada imagen en la que se marcarán a mano los defectos. Estas máscaras permitirán al clasificador saber qué partes de la imagen son defectos y cuales no.

El clasificador no puede trabajar directamente con imágenes, sino con vectores de características, que serán las que se calculen a partir de las imágenes de ejemplo. Estos vectores de características se guardarán en ficheros *ARFF*, que tendrán una serie de atributos definidos en la cabecera, cada uno de ellos correspondiente a una característica. El fichero contendrá un conjunto de instancias, que son cada serie de valores que toman los atributos. Los ficheros *ARFF* son el formato propio de *Weka*, y en su estructura se pueden diferenciar las siguientes secciones:

- **@relation:** Los ficheros *ARFF* deben empezar con esta declaración en su primera línea (no puede haber líneas en blanco antes). Será una cadena de caracteres.
- **@attribute:** En esta sección hay que poner una línea por cada atributo que vaya a tener el conjunto de datos. Para cada atributo habrá que indicar su nombre y el tipo de dato. El tipo puede ser numeric, string, etc.
- **@data:** En esta sección se incluyen los datos. Cada columna se separa por comas y todas las filas deberán tener el mismo número de columnas, que coincidirá con el número de atributos declarados.

Al entrenar al clasificador obtendremos un modelo, el cual usaremos cuando queramos detectar los defectos de una nueva imagen.

Un clasificador puede ser, según los tipos de aprendizaje:

- Supervisado: cuando se utilizan ejemplos previamente etiquetados.
- No supervisado: cuando se utilizan patrones de entradas para los no se especifican los valores de sus salidas.

Por lo dicho anteriormente, hemos utilizado clasificadores supervisados. Nos hemos basado en el estudio que hicieron los alumnos del año pasado con varios tipo de clasificadores, así que hemos usado el que ellos consideraron mejor: *Bagging*. [¿INCLUIR LA INFO DE BAGGING?]

Como algoritmo base hemos usado *REPTree*, que, por sus características, también lo hemos podido usar para la regresión. [¿METER LO DE REPTREE?]

2.7 Ensayos no destructivos

Los ensayos no destructivos [65] son pruebas que, practicadas sobre un material, no alteran de forma permanente sus propiedades físicas, químicas, mecánicas o dimensionales. Los ensayos no destructivos implican un daño imperceptible o nulo.

Los diferentes métodos de ensayos no destructivos se basan en la aplicación de fenómenos físicos tales como ondas electromagnéticas, acústicas, elásticas, emisión de partículas subatómicas, capilaridad, absorción y cualquier tipo de prueba que no implique un daño considerable a la muestra examinada.

Los datos aportados por este tipo de ensayos suelen ser menos exactos que los de los ensayos destructivos. Sin embargo, suelen ser más baratos, ya que no implican la destrucción de la pieza a examinar. En ocasiones los ensayos no destructivos buscan únicamente verificar la homogeneidad y continuidad del material analizado, por lo que se complementan con los datos provenientes de los ensayos destructivos.

Uno de los aspectos más importantes de cualquier método de ensayo no destructivo es que todo el personal debe estar entrenado, cualificado un negativo fotográfico y certificado. El personal debe estar familiarizado con las técnicas, el equipamiento, el objeto a ensayar y cómo interpretar los resultados.

En este proyecto se ha utilizado el ensayo radiográfico.

2.7.1 Radiografía

Una radiografía [70] es una imagen registrada en una placa o película fotográfica, o de forma digital en una base de datos. La imagen se obtiene al exponer al receptor de imagen radiográfica a una fuente de radiación de alta energía, comúnmente rayos X o radiación gamma procedente de isótopos radiactivos. Al interponer un objeto entre la fuente de radiación y el receptor, las partes más densas aparecen con diferentes tonos dentro de una escala de grises, en función inversa a la densidad del objeto. Por ejemplo, si la radiación incide directamente sobre el receptor, se registra un tono negro.

Sus usos pueden ser tanto médicos, para detectar fisuras en huesos, como industriales en la detección de defectos en materiales y soldaduras, tales como grietas, poros, rebabas, etc.

La radiografía se usa para ensayar una variedad de productos, tales como objetos de fundición, objetos forjados y soldaduras. Es también muy usada en la industria aeroespacial para la detección de grietas (fisuras) en las estructuras de los aviones, la detección de agua en las estructuras tipo panal y detección de objetos extraños. Los objetos a ensayar se exponen a rayos X o gamma y se procesa un film o se visualiza digitalmente. El personal de ensayos radiográficos instala, expone y procesa la película o digitalmente procesa las señales e interpreta las imágenes de acuerdo con códigos.

2.7.2 Ventajas del ensayo radiográfico

Las ventajas del ensayo radiográfico [70] incluyen lo siguiente:

1. Puede usarse con la mayoría de los materiales.

2. Puede usarse para proporcionar un registro visual permanente del objeto ensayado o un registro digital con la subsiguiente visualización en un monitor de computadora.
3. Puede revelar algunas discontinuidades dentro del material.
4. Revela errores de fabricación y a menudo indica la necesidad de acciones correctivas.

2.7.3 Limitaciones del ensayo radiográfico

Las limitaciones de la radiografía [70] incluyen consideraciones físicas y económicas.

1. Deben seguirse siempre los procedimientos de seguridad para las radiaciones.
2. La accesibilidad puede estar limitada. El operador debe tener acceso a ambos lados del objeto a ensayar.
3. Las discontinuidades que no son paralelas con el haz de radiación son difíciles de localizar.
4. La radiografía es un método caro de ensayo.
5. Es un método de ensayo que consume mucho tiempo. Después de tomar la radiografía, la película debe ser procesada, secada e interpretada (aunque este problema desaparece cuando la imagen de rayos X se registra digitalmente).
6. Algunas discontinuidades superficiales pueden ser difíciles, si no imposible, de detectar.

2.7.4 Objetivos del ensayo radiográfico

El objetivo del ensayo radiográfico [70] es asegurar la confiabilidad del producto. Esto puede lograrse sobre la base de los siguientes factores.

1. La radiografía permite al técnico ver la calidad interna del objeto ensayado o evidencia la configuración interna de los componentes.
2. Revela la naturaleza del objeto ensayado sin perjudicar la utilidad del material.
3. Revela discontinuidades estructurales, fallas mecánicas y errores de montaje.

La realización del ensayo radiográfico es sólo una parte del procedimiento. Los resultados del ensayo deben ser interpretados de acuerdo con normas de aceptación, y luego el objeto ensayado es aceptado o rechazado.

2.7.5 Principios del ensayo radiográfico

Los rayos X y gamma [70] tienen la capacidad de penetrar los materiales incluso los materiales que no transmiten la luz. Al pasar a través de un material, algunos de esos rayos son absorbidos. La cantidad de radiación que se transmite a través de un objeto ensayado varía dependiendo del espesor y densidad del material y del tamaño de la fuente que se use. La radiación transmitida a través del objeto produce una imagen radiográfica. El objeto ensayado absorbe radiación, pero hay menos absorción donde el objeto es más fino o donde se presenta un vacío. Las porciones

más gruesas del objeto o las inclusiones más densas se verán como imágenes más claras en la radiografía porque aumenta el espesor y la absorción es mayor.

2.8 Fundamentos teóricos de la versión inicial

En este apartado se describe la primera versión que realizamos de la detección de defectos. Esta primera versión es una versión mejorada de la versión final de los alumnos del año pasado.

Se ha continuado la misma idea que utilizaron los alumnos del año pasado, basada en el artículo «Automated Detection of Welding Defects without Segmentation» de Domingo Mery [38]. Lo único que se ha cambiado, aparte del diseño arquitectónico y parte del código y de la interfaz, es que ahora usamos un enfoque multihilo y que se ha cambiado un poco parte del proceso de entrenamiento.

2.8.1 Preproceso

Para la realización del análisis de la imagen y su posterior detección de defectos, hay que partir una imagen original en escala de grises. A dicha imagen se la pasa un filtro *«Saliency Map»*. Con todo esto, se obtiene la imagen original y la imagen Saliency que son analizadas paralelamente.

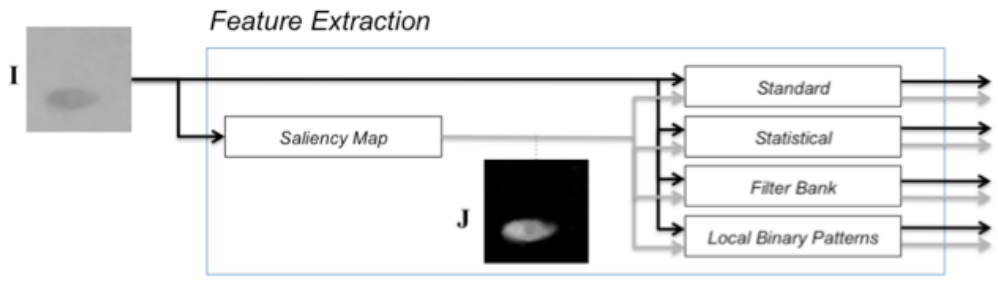

Figura 2.5.: Esquema del proceso de extracción de características [38]

2.8.2 Extracción de características

Antes de detectar cualquier tipo de defecto, es necesario entrenar un clasificador para que pueda predecir donde están los defectos buscados. Para ello se analizan un conjunto de imágenes de las que se crean máscaras en las que se pintan los defectos. Estas máscaras sirven para que el clasificador sepa qué partes de la imagen son defectos y cuáles no.

Para analizar cada imagen se utiliza una ventana que recorre toda la región de interés analizando sus características. En el artículo original sólo aparecía el número total de características, lo cual dificultó la identificación del número que había que calcular para cada tipo, por eso nosotros las hemos desglosado según su clase:

- **Características estándar:** 4 características.

- **Características de Haralick:** Se calculan 14 características, obteniendo 5 vectores de medias y 5 de rangos. $14 \times 10 = 140$ características.
- **Características LBP:** Se obtiene un histograma con 59 intervalos, es decir, 59 características.

El número final de características habrá que multiplicarlo por dos, ya que todas las características se calculan tanto para la imagen original como para la imagen con saliency map aplicado. Por lo tanto, el total sería:

(4 estándar + 140 haralick + 59 lbp)×2 = 406 CARACTERÍSTICAS

Estas características serán las que se guarden en los ficheros *ARFF* que se utilicen para entrenar al clasificador, junto con la clase, que tendrá valor «true» si la instancia tiene defecto, o «false» si no. En caso de que se use regresión lineal serán 0 y 1. El defecto se identificará gracias a las máscaras mencionadas anteriormente.

La ventana irá analizando cada vez una región de la imagen, recorriendo todos y cada uno de sus píxeles, de los que extraerá las características que posteriormente se analizarán. Cada vez que la ventana se mueva y analice una nueva región, se creará una nueva instancia, que se corresponderá con una línea del fichero *ARFF* en el que se guardan las características. Nosotros hemos hecho que el tamaño de la ventana sea configurable, aunque por defecto se utilizará una ventana de 24×24 píxeles, como en el artículo. Esta ventana puede ser de dos tipos:

■ Ventana deslizante

La ventana comienza desde la esquina superior izquierda de la imagen y se va moviendo cada cierto porcentaje del tamaño de la ventana. Cuando llega al extremo derecho baja ese mismo porcentaje y comienza de nuevo desde el lado izquierdo. A diferencia del artículo, donde se usa siempre un salto de 4 píxeles, nosotros hemos hecho que se pueda seleccionar el tamaño del salto.

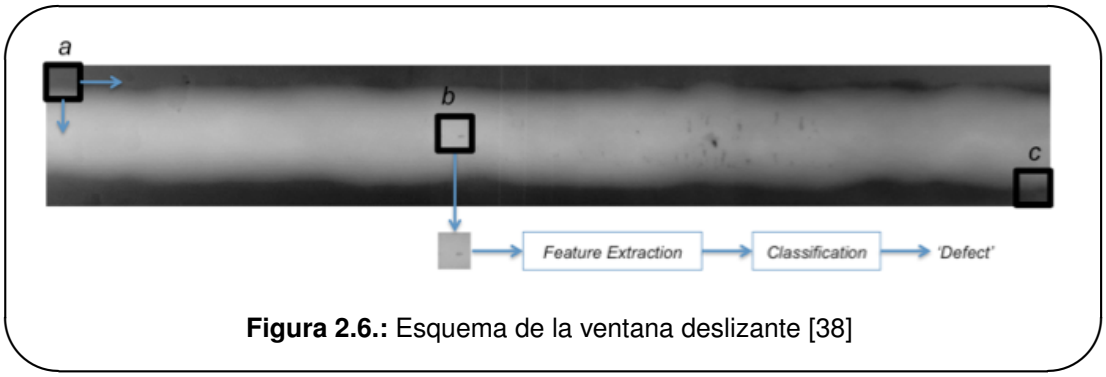

Figura 2.6.: Esquema de la ventana deslizante [38]

■ Ventana aleatoria

Se obtienen 300 muestras de ventanas por imagen, seleccionándolas aleatoriamente entre aquellas que tienen defecto y las que no.

2.8.3 Determinación de cuándo una ventana es defectuosa

Como hemos visto, se hace necesario determinar cuándo una ventana tiene defecto o no, ya que se necesita etiquetar las muestras de cada ventana como «defecto» o «no defecto». Los alumnos del año pasado utilizaron una aproximación muy simple: consideraban que una ventana era defectuosa cuando, al poner esa ventana sobre la máscara coloreada manualmente, al menos un píxel de la misma estaba coloreado. Comprobamos que esto provoca falsos positivos, con lo que se pierde exactitud. Por lo tanto, decidimos implementar otras posibilidades:

■ Píxel central

En este caso, se considera una ventana defectuosa cuando el píxel central de la misma es defectuoso. Es un poco más preciso que la versión del año pasado, pero sigue sin ser demasiado buena.

■ Porcentaje de la ventana

En esta aproximación, se considera defectuosa una ventana cuando al menos un cierto porcentaje de la ventana contiene píxeles coloreados, es decir, defectuosos. Hemos obtenido resultados muy buenos con porcentajes que van desde el 50 % al 75 %.

■ Píxel central más región de vecinos

En este caso, no sólo consideramos el píxel central, si no que creamos una región cuadrada de 3×3 píxeles a su alrededor. Consideramos entonces una ventana como defectuosa cuando un cierto porcentaje de esta región de vecinos está coloreada. Obtenemos unos resultados muy parecidos a los de la anterior aproximación con porcentajes parecidos.

2.8.4 Detección de defectos

El proceso es prácticamente igual al entrenamiento del clasificador. Se genera una imagen Saliency Map a partir de la imagen original, al igual que antes, y a partir de las dos imágenes (la original y la Saliency) se analizan mediante una ventana deslizante.

Cada vez que la ventana se desplaza se genera una instancia que contiene todas las características analizadas. Esta instancia es utilizada por el modelo que predice si la ventana está situada sobre un defecto.

Si la ventana contiene un defecto, es marcada con un cuadrado verde y a cada uno de sus píxeles se le suma una unidad (este valor de los píxeles desde ahora será llamado factor).

Estos pixeles son almacenados en una matriz global del mismo tamaño que la imagen a analizar, por lo que según se desplaza la ventana deslizante se van actualizando todos los valores de los pixeles que contienen defectos.

Una vez finalizado el proceso de detección, se binariza la matriz resultante según su factor. Para este proceso en el artículo original [38] se utiliza un factor de 24.

Éste factor es descrito como el número de veces que ha sido marcado un píxel como defecto. Si un píxel tiene un factor menor que 24, dicho pixel se considera como no defecto. En cambio, si un píxel tiene un factor de 24 o más será considerado como defecto.

Dicho esto, en el proyecto utilizamos un factor variable. Por defecto el factor es de 8, pero tras la ejecución se puede variar para observar los cambios en la imagen en tiempo real.

Como ya se ha descrito, el proceso de binarización recae en el factor de un píxel. Si el factor es 8 o mayor, al pixel se le asigna un 1 (defecto), si es menor que 8 se le asigna un 0 (no defecto). Al tener una matriz de ceros y unos es sencillo obtener una imagen binaria con la región de defectos.

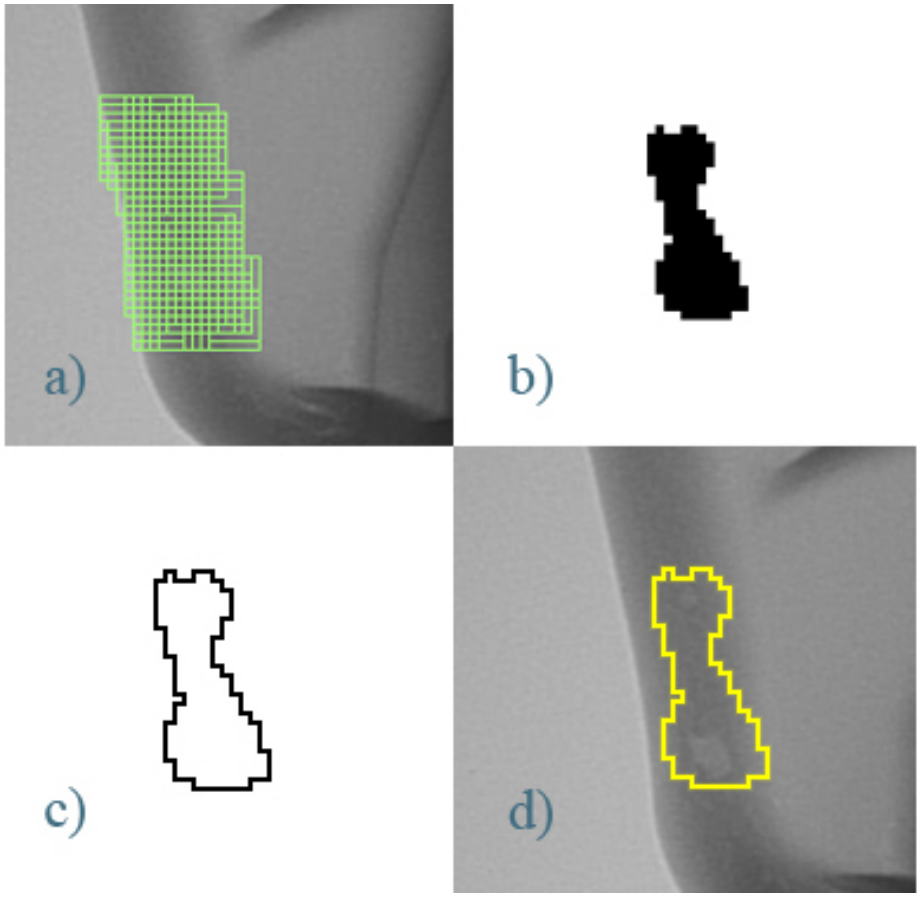

Figura 2.7.: Proceso de dibujado de defectos en nuestro proyecto.

Una vez obtenida la imagen binarizada con la región de defectos se le aplica un filtro de detección de bordes para marcar una línea que rodee el defecto mostrando su ubicación. En este caso se ha utilizado el filtro por defecto de detección de bordes de ImageJ.

Observamos que el proceso de detección de bordes invierte los colores, la línea queda en blanco y el fondo en negro, así que invertimos los colores utilizando uno de los filtros de ImageJ.

Hecho esto, solo queda poner el fondo transparente y superponer la imagen del borde sobre la imagen original.

2.8.5 Multihilo

Como ya hemos dicho varias veces, uno de los objetivos del proyecto es mejorar el rendimiento de la versión del año pasado. Una de las principales modificaciones para conseguir esto ha sido la inclusión de una estrategia multihilo, aprovechando las oportunidades que nos brindan los procesadores actuales.

Lo que hemos hecho ha sido dividir la imagen en tantas partes como procesadores disponibles tenga la máquina que está ejecutando el programa. Por ejemplo, si tenemos 2 procesadores, la imagen se dividirá en 2, en su dimensión vertical. Hay que tener en cuenta un pequeño margen, necesario para que el cálculo de algunas características sea correcto. Por ello, en el ejemplo anterior, las 2 imágenes no serían exactamente de la mitad de altura que la original, si no que serían un poco más grandes. Hay un pequeño solapamiento.

Una vez que tenemos dividida la imagen, se ejecuta el proceso de detección o de entrenamiento sobre cada uno de los trozos de imagen. Con esto obtenemos un rendimiento mucho mayor que la versión del año pasado.

2.9 Fundamentos teóricos de la versión final

Con las modificaciones descritas en el apartado anterior, ya observamos que tanto el rendimiento como la precisión habían aumentado con respecto al año pasado. Aún así, teniendo en mente la posibilidad de calcular otra serie de características (como las geométricas) sobre los defectos detectados para ,en un futuro, poder clasificarlos en sus diferentes tipos, se decidió intentar mejorar aún más esta precisión.

Se implementaron dos nuevas aproximaciones a la hora de detectar defectos (por lo tanto, el proceso de entrenamiento cambia). En ambas aproximaciones nos valemos de una imagen segmentada con los filtros de umbrales locales que ya hemos visto. Lo que cambia entre ellas es cómo consideramos estos flitros.

2.9.1 Primera opción: detección normal con posterior intersección con umbrales locales

En esta primera aproximación, primero se realiza la detección de defectos como en la primera versión del proyecto. La diferencia está en el dibujado definitivo de los defectos.

Con la detección de defectos normal, obtenemos una matriz del mismo tamaño que la imagen, a partir de la cual se dibujan los defectos. En esta matriz se guarda, en cada posición (es decir, en cada píxel), el número de ventanas que los han cubierto y que se han marcado como defecto. Si este número supera cierto umbral, habrá una nueva matriz, en la que habrá un uno en todos aquellos píxeles que son realmente defecto. A partir de esta matriz, se crea el dibujado definitivo.

En nuestra aproximación, introducimos un paso intermedio antes de generar la matriz definitiva. En vez de poner directamente un uno o un cero, lo que hacemos es segmentar la imagen mediante el filtro de umbrales locales. A continuación, los píxeles que superan el umbral seleccionado son comparados con la imagen de umbrales locales. Si en esta imagen aparecen como blancos, se consideran defecto. Si no, se descarta.

2.9.2 Segunda opción: píxeles blancos en umbrales locales

Esta aproximación difiere bastante más respecto a las opciones ya vistas.

En este caso, lo primero que se hace es generar la imagen de umbrales locales. A partir de esta imagen, generamos una lista con las coordenadas de los píxeles que se han marcado como blancos.

Después, binarizamos la imagen de umbrales locales para aplicar un análisis de partículas sobre ella, con el objetivo de obtener todas las regiones que destacan, a las que consideramos regiones candidatas de albergar un defecto.

Cuando tenemos la lista y las regiones, vamos sacando coordenadas de la lista y vamos determinando la región a la que pertenecen. Dependiendo del tamaño de la región y el tamaño de la ventana, se considera la coordenada o se desecha. En caso de que se considere, centramos una ventana sobre ella y aplicamos el cálculo de características y clasificación ya vistos anteriormente.

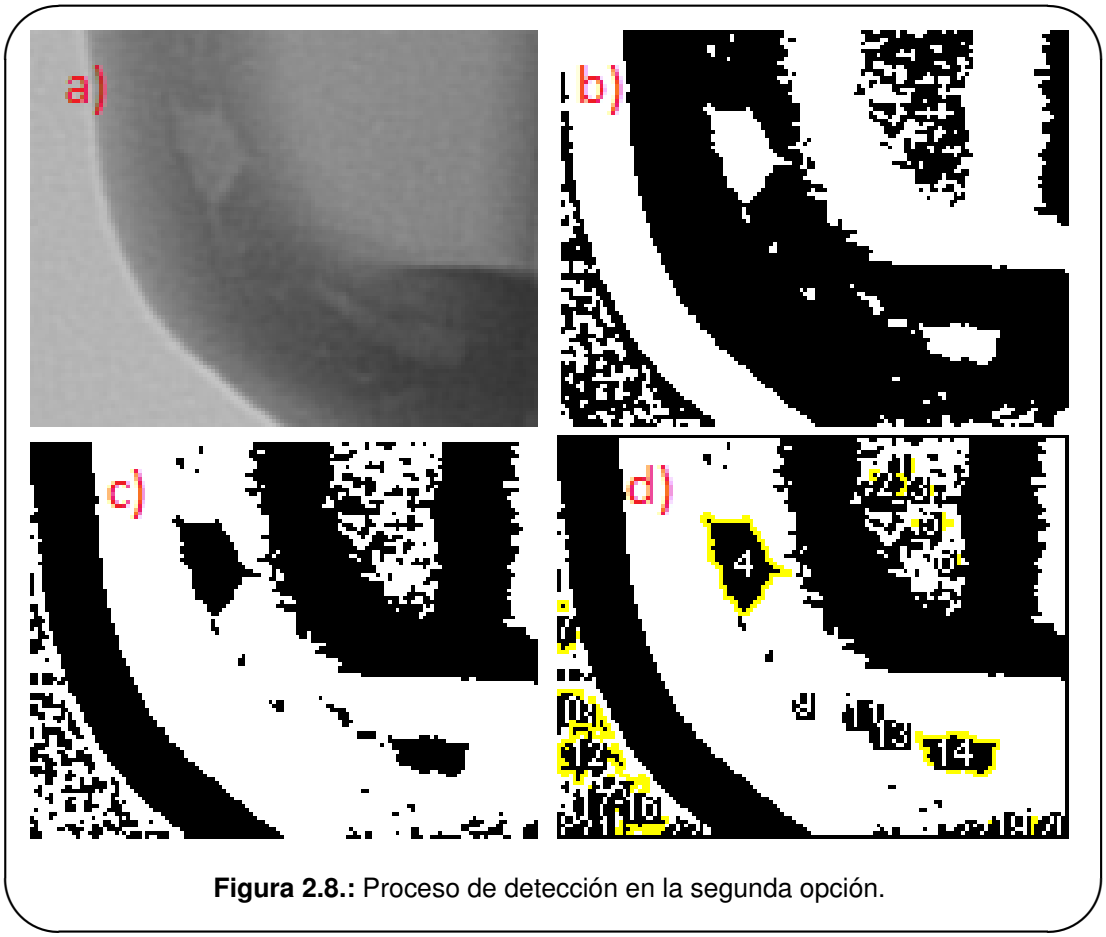

Figura 2.8.: Proceso de detección en la segunda opción.

El dibujado de los defectos se realiza con la misma matriz que en la primera versión del proyecto.

La parte de analizar las regiones para determinar si hay que considerar la coordenada o no fue añadida después de una primera versión de esta opción, en la que se consideraba toda la lista de píxeles blancos. Se decidió cambiar porque si considerábamos toda la lista obteníamos muchísimos falsos positivos, con lo que la precisión no era buena.

Para aumentar el rendimiento, antes de empezar a calcular características, se divide la lista de píxeles blancos en tantas partes como procesadores disponibles haya. Después, habrá tantos hilos como partes, en los que en cada uno de ellos se iterará sobre cada una de las partes, de forma paralela.

2.9.3 **Cálculo de características geométricas**

Una vez que se ha realizado el proceso de detección y dibujado de defectos, podemos calcular los descriptores geométricos de estos defectos. Para ello, se usa la imagen binarizada con la región de defectos para aplicar sobre ella un proceso de segmentación, a través del cual vamos a poder obtener una serie de regiones, sobre las cuales se puede aplicar el cálculo de las características geométricas ya mencionadas. Con estos resultados creamos una tabla que se irá refrescando si el usuario selecciona otro umbral de detección.

Estas características permitirán, en un futuro, aplicar un proceso de clasificación sobre las regiones detectadas como defecto en distintos tipos (burbuja, rotura...).

3. TÉCNICAS Y HERRAMIENTAS

En este apartado se comentan las técnicas y herramientas utilizadas durante el desarrollo del proyecto.

3.1 Técnicas

En esta sección aparecen recogidas las técnicas utilizadas para la realización del proyecto.

3.1.1 Metodología Scrum

Scrum [**?**] es una metodología para la gestión y desarrollo de proyectos software basada en un proceso iterativo e incremental. Cada iteración termina con una pieza de software ejecutable que incorpora una nueva funcionalidad o mejora las ya existentes. Estas iteraciones suelen durar de dos a cuatro semanas.

Enumerando los elementos clave de *Scrum* según *Control Chaos* [**?**]:

- *Scrum* es un proceso ágil para gestionar y controlar el trabajo de desarrollo.
- *Scrum* es un envoltorio para prácticas de ingeniería existentes.
- *Scrum* es una aproximación basada en equipos para desarrollar sistemas y productos iterativa e incrementalmente cuando los requisitos cambian rápidamente.
- *Scrum* es un proceso que controla el caos de necesidades e intereses en conflicto.
- *Scrum* es una forma de mejorar las comunicaciones y maximizar la cooperación.
- *Scrum* es una forma de detectar y eliminar cualquier cosa que se interponga en el desarrollo y distribución de productos.
- *Scrum* es una forma de maximizar la productividad.
- *Scrum* es escalable desde un único proyecto a organizaciones completas. Ha controlado y organizado el desarrollo e implementación de muchos productos y proyectos interrelacionados con más de mil desarrolladores e implementadores.
- *Scrum* es una forma de que todo el mundo se sienta bien con su trabajo, sus aportaciones, y que ellos han hecho lo mejor que pueden hacer.

Los principales beneficios que aporta *Scrum* son:

- Entrega mensual o trimestral de resultados lo cual aporta las siguientes ventajas:
  - Gestión regular de las expectativas del cliente y basada en resultados tangibles: El cliente establece sus prioridades y cuando espera tenerlo acabado.

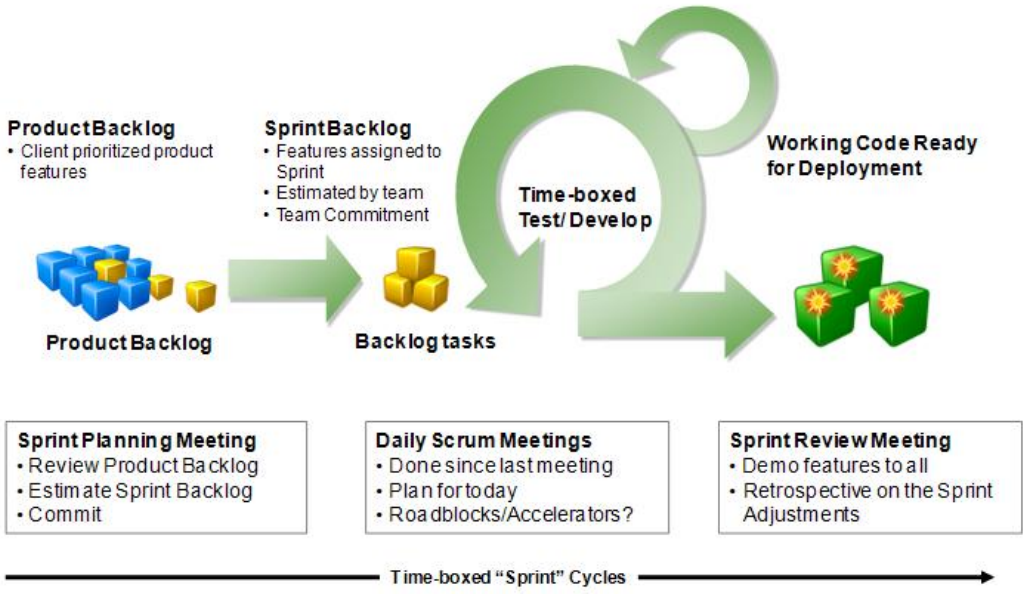

Figura 3.1.: Diagrama de la metodología Scrum

- Resultados anticipados: el cliente puede empezar a utilizar los resultados mas importantes antes de que esté totalmente finalizado el proyecto.
- Flexibilidad de adaptación respecto a las necesidades del cliente, cambios en el mercado, etc.
- Gestión sistemático del retorno de inversión (*ROI*): el cliente maximiza el *ROI* del proyecto, de este modo cuando el beneficio pendiente de obtener es menor que el coste del desarrollo el cliente puede finalizar el proyecto.
- Mitigación sistemática de riesgos del proyecto: la cantidad de riesgo a la que se enfrenta el equipo está limitada a los requisitos que se puede desarrollar en una iteración.

- Productividad y calidad: de manera regular el equipo de desarrollo va mejorando y simplificando su manera de trabajar.
- Alineamiento entre cliente y el equipo de desarrollo: todos los participantes del proyecto conocen cuál es el objetivo a conseguir. El producto se enriquece con las aportaciones de todos.
- Equipo motivado: las personas están más motivadas cuando pueden usar su creatividad para resolver problemas y cuando pueden decidir organizar su trabajo.

En el diagrama (ver figura 3.1) [**?**] se muestra el funcionamiento y actividades de la metodología *Scrum*.

Algunos conceptos básicos para entender *Scrum* son:

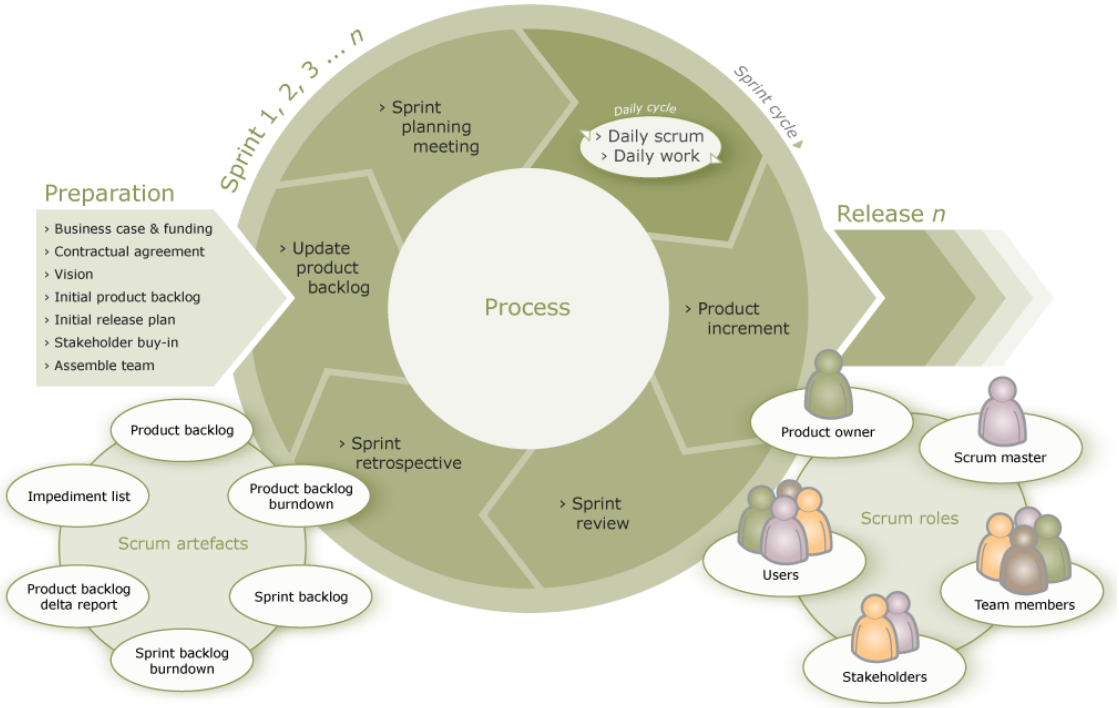

Figura 3.2.: Diagrama de etapas en Scrum

- *Product Backlog*: conjunto de historias de usuario que representan los requisitos funcionales y no funcionales. Se trata de una lista priorizada en función de lo que el cliente da mayor importancia.
- *Sprint Backlog*: conjunto de tareas extraídas del *Product Backlog* y que serán realizadas durante un *Sprint*.
- *Burndown Chart*: gráfico de tareas pendientes por hacer. Representan el esfuerzo y ofrece información sobre la evolución del proyecto.

El objetivo del diagrama adjunto (ver figura 3.2) [**?**] es el de agrupar y sintetizar todos los elementos de la metodología *Scrum*.

Roles

Scrum [**?**] define una serie de roles y que se dividen en dos grupos: gallinas y cerdos.

- Roles «cerdo»: son aquellos que están comprometidos a construir el software de manera regular y frecuente.
  - *Product Owner*: representa la voz del cliente. Debe asegurarse de que el equipo trabaja de forma adecuada desde la perspectiva de negocio. Escribe las historias de usuario, las prioriza y las coloca en el *Product Backlog*.
  - *Scrum Master*: su principal trabajo es eliminar obstáculos que puedan hacer que el equipo no alcance el objetivo al final del sprint, no es el líder del equipo, pero sirve de pantalla y protección.

    - *Team*: es el equipo de desarrollo y su responsabilidad es la de generar y entregar el producto (diseñadores, programadores...).

- Roles «gallina»: en realidad no son parte del proceso *Scrum*, pero deben tenerse en cuenta. Estos roles deben participar en el proceso.
    - *Stakeholders*: agrupa a la gente que hace posible el proyecto y para quienes el proyecto producirá el beneficio que justifica el coste. Dentro de este grupo estarían los clientes, *stakeholders* ...
    - *Managers*: es la gente que establece el ambiente para el desarrollo del producto.

Reuniones

Scrum define una serie de reuniones para el correcto funcionamiento del equipo. Éstas se encuentran bien definidas en cuanto a contenido y a tiempo empleado.

- *Daily Scrum*: Cada día de un *Sprint* se realiza una reunión sobre el estado del proyecto.
    - La duración es fija (15 minutos) independientemente del tamaño del equipo.
    - La reunión debe comenzar puntualmente a la hora. A menudo hay castigos para quien lo incumple.
    - Todos pueden estar presentes pero solo pueden hablar los "cerdos".
    - La reunión se realiza de pie, facilitando no alargar la reunión.
    - El lugar y la hora deben ser fijos todos los días.
    - Preguntas que debe responder cada miembro del equipo:
        - ¿Qué has hecho desde ayer?
        - ¿Qué vas a hacer hoy?
        - ¿Qué obstáculos te has encontrado?
- *Sprint Planning Meeting*: Al inicio de cada *Sprint* se debe llevar a cabo una.
    - Crear y planificar, el equipo completo, el *Sprint Backlog*. Se obtiene extrayendo tareas del *Product Backlog*.
    - Límite de ocho horas.
- *Sprint Review Meeting*: Al finalizar cada *Sprint*.
    - Revisar el trabajo que fue planificado y no ha sido completado.
    - Presentar el trabajo completado a los interesados. El trabajo incompleto no puede ser mostrado.
    - Límite de cuatro horas.
- *Sprint Retrospective*: Al finalizar cada *Sprint*.
    - Se analiza el *Sprint* y todos los miembros del equipo analizan qué mejoras podrían aplicarse.

- Su objetivo es la mejora continua.
- Límite de cuatro horas.

3.1.2 Java

Java es un lenguaje de programación que data de finales de los años 70. Ha tenido una gran implantación debido a la sencillez respecto a otros lenguajes orientados a objetos como *C++*.

Java ofrece un API (Application Program Interface) que ofrece a los programadores una serie de librerías y facilidades para el desarrollo de aplicaciones *Java*.

El API se encuentra divido en paquetes, que son la estructura de organización lógica. En su interior se encuentran una gran cantidad de clases que cubren un amplio abanico de funcionalidades del desarrollo software en general.

La documentación del API se encuentra disponible en la web y su consulta resulta imprescindible para cualquier tipo de desarrollo en *Java*.

■ Máquina virtual

La máquina virtual es un programa capaz de interpretar y ejecutar instrucciones expresadas en un código especial (el *Java bytecode*). Este código se obtiene al compilar el fuente original con el compilador de *Java*.

Este código es un lenguaje máquina de bajo nivel que es interpretado por la máquina virtual para realizar las operaciones. Ésto hace que el rendimiento de los programas escritos en *Java* sea inferior ya que aparece una nueva pieza intermedia que es la máquina virtual.

La ventaja de ser un lenguaje interpretado es que cualquier programa escrito en *Java* puede ejecutarse en cualquier *hardware* o sistema operativo, la única condición necesaria es que exista una máquina virtual disponible.

3.1.3 UML

El Lenguaje Unificado de Modelado (UML) [18] es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Se trata de un lenguaje gráfico, llamado «lenguaje de modelado», que se utiliza para visualizar, especificar, construir y documentar un sistema, describiendo sus métodos o procesos. Es el lenguaje en el que está descrito el modelo.

UML permite modelar la estructura, comportamiento y arquitectura de las aplicaciones. Además, la programación orientada a objetos, que ha sido la elegida para este proyecto, es un complemento perfecto de UML. Por estas razones se ha elegido UML, en su versión 2.0.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. Para la realización del proyecto se han utilizado los siguientes tipos:

- **Diagrama de Casos de Uso:** muestra los casos de uso, actores y sus interrelaciones.
- **Diagrama de Paquetes:** muestra como los elementos de modelado están organizados en paquetes, ademas de las dependencias entre esos paquetes.

- **Diagrama de Clases:** representa una colección de elementos de modelado estáticos, tales como clases y tipos, sus contenidos y sus relaciones.
- **Diagrama de Secuencias:** modela la lógica secuencial, ordenando en el tiempo los diferentes mensajes entre entidades.

El análisis, diseño e implementación del sistema se ha realizado empleando esta técnica, gracias a lo aprendido en las diferentes asignaturas de la carrera.

3.1.4 Weka

Weka es una plataforma de software para aprendizaje automático y minería de datos, diseñada por la Universidad de Waikato. Está escrito en *Java*. Se trata de software libre distribuido bajo licencia *GNU*.

Las principales ventajas que nos ofrece son:

1. Se encuentra escrito en *Java*, lenguaje que se va a usar en el proyecto.
2. Una consecuencia de lo anterior es que *Weka* es portable, puede funcionar en cualquier sistema operativo.
3. Se dispone del código fuente para ver cómo hace las cosas y cómo funciona.
4. Buena documentación (*JavaDoc*) para ayudar a la programación.
5. Completa API capaz de representar de una manera sencilla la abstracción de las instancias.
6. Es capaz de obtener los datos de diversos orígenes, tanto de texto como *ARFF* y *CSV*, como de bases de datos.
7. Su uso está muy generalizado en el mundo de la minería de datos.
8. Ya utilizado con anterioridad, por lo que su manejo no resulta nuevo.

No todo son ventajas y a continuación se detallan los inconvenientes valorados:

1. *Weka* carga todas las instancias en memoria por lo que se limita el número de instancias que es capaz de manejar.

Weka soporta varias tareas estándar de minería de datos, especialmente, preprocesamiento de datos, clustering, clasificación, regresión, visualización, y selección. Todas las técnicas de *Weka* se fundamentan en la asunción de que los datos están disponibles en un fichero plano (flat file) o una relación, en la que cada registro de datos está descrito por un número fijo de atributos (normalmente numéricos o nominales, aunque también se soportan otros tipos).

3.1.5 ImageJ

ImageJ [3] es un programa de procesamiento de imagen digital de dominio público programado en *Java* desarrollado en el National Institutes of Health.

La licencia de ImageJ es la siguiente:

> ImageJ is a work of the United States Government. It is in the public domain and open source. There is no copyright. You are free to do anything you want with this source but I like to get credit for my work and I would like you to offer your changes to me so I can possibly add them to the, "official" version.

Lo que quiere decir básicamente que somos libres de hacer lo que queramos con ImageJ pero que si realizamos algún cambio deberíamos ofrecérselos al creador para que los añada a la versión «oficial».

ImageJ fue diseñado con una arquitectura abierta que proporciona extensibilidad vía plugins *Java* y macros (macroinstrucciones) grabables. Se pueden desarrollar plugins de escaneo personalizado, análisis y procesamiento usando el editor incluido en ImageJ y un compilador *Java*. Los plug-ins escritos por usuarios hacen posible resolver muchos problemas de procesado y análisis de imágenes, desde de imágenes en vivo de las células en tres dimensiones, procesado de imágenes radiológicas, comparaciones de múltiples datos de sistema de imagen hasta sistemas automáticos de hematología.

Aunque ImageJ es extensible mediante plugins y macros, nosotros lo hemos elegido con la finalidad de utilizarlo como librería.

ImageJ puede mostrar, editar, analizar, procesar, guardar, e imprimir imágenes de 8 bits (256 colores), 16 bits (miles de colores) y 32 bits (millones de colores). Puede leer varios formatos de imagen incluyendo TIFF, PNG, GIF, JPEG, BMP, DICOM, FITS, así como formatos RAW (formato).

ImageJ aguanta pilas o lotes, una serie de imágenes que comparten una sola ventana, y es multiproceso, de forma que las operaciones que requieren mucho tiempo se pueden realizar en paralelo en hardware multi-CPU.

ImageJ puede calcular el área y las estadísticas de valor de píxel de selecciones definidas por el usuario y la intensidad de objetos umbral (thresholded objects). Puede medir distancias y ángulos. Se puede crear histogramas de densidad y gráficos de línea de perfil.

Es compatible con las funciones estándar de procesamiento de imágenes tales como operaciones lógicas y aritméticas entre imágenes, manipulación de contraste, convolución, Análisis de Fourier, nitidez, suavizado, detección de bordes y filtrado de mediana. Hace transformaciones geométricas como ampliar, rotación y flips. El programa es compatible con cualquier número de imágenes al mismo tiempo, limitado solamente por la memoria disponible.

Preferimos usar ImageJ frente a otras opciones, como OpenGL, principalmente por su buena documentación (*JavaDoc*), cosa que facilita mucho la programación.

3.1.6 Patrones de diseño

Un patrón de diseño es una solución a un problema de diseño común en el desarrollo de software.

La principal característica de un patrón de diseño es que debe ser reusable, es decir, debe poder aplicarse a diferentes problemas de diseño en distintas circunstancias y ser efectivo.

Los patrones de diseño utilizados para el desarrollo de este proyecto se explican en el anexo 3 de esta memoria.

3.2 Herramientas

En esta sección aparecen aparecen cada una de las herramientas utilizadas para la realización del proyecto.

3.2.1 Eclipse

Como entorno de desarrollo de la biblioteca se utilizará Eclipse. La decisión fue tomada por haber trabajado anteriormente con esta herramienta y conocer sus ventajas e inconvenientes. La disponibilidad de *plugins* disponibles facilita el desarrollo, integrando el control de versiones, *suites* de pruebas ...

Eclipse es un producto realizado por la *Eclipse Foundation*, que es una comunidad de codigo abierto que tiene como objetivo desarrollar una plataforma para el desarrollo software.

Se encuentra escrito en *Java* bajo una licencia propia, la *EPL* (Eclipse Public License [**?**]).

Aunque en su origen se creó para *Java* existen versiones de todo tipo para otros lenguajes como pueden ser *C* o adaptaciones comerciales para productos o lenguajes concretos.

Página web de la herramienta: `http://www.eclipse.org/`.

3.2.2 JUnit

JUnit es un conjunto de bibliotecas o *framework* que son utilizadas para realizar las pruebas unitarias de aplicaciones *Java*. Dispone de una buena reputación dentro de la literatura sobre programación de pruebas.

El propio *framework* permite visualizar los resultados en texto, como gráficos o como tarea de *Ant*.

Se utilizará el *plugin* de eclipse por la facilidad que aporta para la ejecución de las pruebas y su total integración con el entorno de desarrollo.

Se ha utilizado por haber sido utilizado durante la carrera y ser el lanzador más conocido por el autor.

Página web de la herramienta: `http://www.junit.org/`.

3.2.3 JDepend

Se trata de una herramienta de métricas que permite conocer información de utilidad de un proyecto software.

Se utilizará el *plugin* para Eclipse que permite visualizar los valores de las métricas y la gráfica comparativa de cada paquete.

Página web de la herramienta: `http://www.clarkware.com/software/JDepend.html`.

3.2.4 Source Monitor

Se trata de una herramienta de análisis de código capaz de analizar proyectos escritos en diversos lenguajes.

Su utilización ha sido motivada porque permitirá comparar las métricas obtenidas con los umbrales establecidos por la Universidad de Burgos.

Página web de la herramienta: http://www.campwoodsw.com/sourcemonitor.html.

3.2.5 Astah

Es una herramienta de modelado *UML* creado por la compañía *ChangeVision*.

Al estar pensado para *Java*, permite la importación y exportación de código fuente y la generación de gráficos automáticos.

En el proyecto se utilizará la versión *Community* porque es gratuita y cubre las necesidades de modelado del proyecto.

Página web de la herramienta: http://astah.net/editions/community.

3.2.6 GitHub

GitHub es una plataforma para el desarrollo colaborativo de software que utiliza el control de versiones *Git*.

Es una herramienta completa y robusta que permite la creación de grupos, ramas, etiquetas y todo tipo de artefactos necesarios para la organización de un proyecto de programación.

Las principales ventajas que nos ofrece son:

- Nuestro código queda alojado en la nube, permitiendo acceder a él desde cualquier lugar.
- Nos permite trabajar con la metodología de *Rama por tarea*, en la que se crea una rama por cada tarea a realizar, permitiendo trabajar en dos tareas simultáneamente, fusionando después los cambios.
- Tiene cliente propio multipltaforma, lo que nos permite gestionar el repositorio de una forma muy sencilla, pudiendo validar los cambios que hagamos en el código y subiendo estos cambios al servidor.

Página web de la herramienta: https://github.com/.

3.2.7 PivotalTracker

PivotalTracker será utilizada como herramienta de gestión y control de tareas y errores.

Está especializada en proyectos ágiles por lo que da soporte a todos los conceptos de la metodología *Scrum* utilizada en el proyecto.

Se utilizará la versión de *hosting* que permite acceder desde cualquier equipo al servidor desde un navegador. Además, permite sincornizarse con *GitHub*, con lo que podemos ver cómo se van realizando las tareas tanto en la propia herramienta como en el código. En el Apéndice A, aparece un breve manual de usuario que permite visualizar la planificación y las tareas.

Página web de la herramienta: https://www.pivotaltracker.com/.

3.2.8 MiKTeX

MikTeX es una distribución TeX/LaTeX libre de código abierto para Windows.

Una de sus características es la capacidad que tiene para instalar paquetes automáticamente sin necesidad de intervención del usuario. Al contrario que otras distribuciones, su instalación es extremadamente sencilla.

Ha sido utilizada por recomendación de César quien suministró un tutorial sobre su instalación.

Página web de la herramienta: http://miktex.org/.

3.2.9 TeXMaker

TeXMaker es un editor de LaTeX multiplataforma similar a *Kile*.

Aunque existen multitud de editores para LaTeX se ha escogido este por haber sido recomendado por César, dado que nunca antes se ha trabajado con este lenguaje de documentación se aceptó la sugerencia. Además, aportó un tutorial de como instalar y configurar el editor en el sistema operativo Windows.

Una de sus características es la posibilidad de trazabilidad de código desde PDF. Configurando el editor y un visor de PDF, el programa es capaz de detectar la línea a la que se corresponde un determinado comando LaTeX. Esto es especialmente ventajoso cuando, como en este caso, no se conoce el lenguaje.

En el enlace http://en.wikipedia.org/wiki/Comparison_of_TeX_editors aparece una comparativa entre diversos editores disponibles.

Página web de la herramienta: http://www.xm1math.net/texmaker/.

3.2.10 WindowBuilder

WindowBuilder es un *plugin* de Eclipse que permite diseñar de una forma fácil y rápida interfaces gráficas basadas en *Swing*.

Hemos elegido este editor de interfaces gráficas debido a su facilidad para crearlas, ya que incluye un editor *WYSIWYG*, con el que podemos arrastrar los elementos a la ventana que estamos creando y moverlos hasta dejarlos en la posición deseada.

La otra parte interesante de este *plugin* es que genera el código automáticamente, con lo que nos ahorra mucho trabajo. Además, el código que genera está bien agrupado, con lo que después es muy fácil refactorizarlo.

Página web del *plugin*: http://www.eclipse.org/windowbuilder/.

3.2.11 Auto Local Threshold

Auto Local Threshold es un *plugin* de ImageJ que implementa la segmentación de imágenes mediante los filtros de umbrales locales.

Por defecto, ImageJ no tiene esta funcionalidad, así que tuvimos que descargarnos e instalar este *plugin*, que permite realizar estas opciones de forma muy sencilla.

Para poder usarlo en el proyecto, tuvimos que cambiar alguna cosa de la clase del *plugin* para poder usarlo con nuestro código.

Página web del *plugin*: http://fiji.sc/wiki/index.php/Auto_Local_Threshold.

3.2.12 Apache Commons IO

Apache Commons IO es una librería de utilidades para asistir al desarrollo de funcionalidad relacionada con entrada/salida.

Hemos decidido utlizarla debido a que simplifica mucho el realizar algunas operaciones con ficheros, como es la fusión de uno o más ficheros de texto, o la exportación de un cierto texto a un fichero externo.

La página web de la herramienta es: http://commons.apache.org/proper/commons-io/.

3.2.13 Zotero

A la hora de realizar cualquier trabajo de investigación o documento de cierta envergadura, como puede ser la memoria de un trabajo final de carrera, el autor del mismo se va a encontrar con un gran número de información a su disposición. Toda esta información proviene de muchas y muy diversas fuentes (libros, artículos, revistas, bases de datos, internet...) y tiende a crecer de forma incontrolada a lo largo del desarrollo del proyecto. Es por ésto por lo que surge la necesidad de utilizar algún sistema que gestione toda esta información de forma eficaz.

Este sistema es un gestor bibliográfico, con el que además de gestionar esta información, también se puede insertar citas en los documentos y crear la bibliografía en un formato normalizado.

Zotero es un gestor bibliográfico de código abierto que se instala como una extensión del navegador Firefox y del navegador Chrome. De esta forma, permite añadir directamente a la bibliografía las páginas web visitadas y buscar documentos en línea.

Permite exportar la bibliografía en formato LaTeX directamente, lo cual es muy cómodo para integrarla directamente en el documento.

Página web de la herramienta: http://www.zotero.org/.

4. ASPECTOS RELEVANTES DEL DESARROLLO DEL PROYECTO

En este apartado se detallan los aspectos más relevantes que se han encontrado en el proceso de desarrollo del proyecto.

4.1 Problemas y retos

El tema principal del proyecto era totalmente desconocido. Para la obtención de los defectos en las radiografías se han utilizado técnicas novedosas que no se enseñan en ninguna asignatura de la carrera. Hay que añadir que algunas de esas técnicas se usan actualmente en proyectos de investigación en universidades y centros de investigación con mucha más experiencia en este campo que la Universidad de Burgos.

Se busca resolver un problema real, con imágenes reales (cedidas por el Grupo Antolín) obtenidas desde los puntos de vista y condiciones que obtiene una máquina real, piezas reales y con formas muy complejas. Es un problema mucho más complicado que el que se aborda en los artículos relacionados donde la imagen suele ser mucho mas uniforme y sencilla. Se puede ver una comparativa de algunas de las imágenes usadas en otros artículos (ver figura 4.1) y un ejemplo de una de las imágenes que usamos en nuestro proyecto (ver figura 4.2).

Por lo anterior, este ha sido un proyecto de gran incertidumbre y riesgo que ha hecho que:

1. Como se preveía que los requisitos del proyecto iban a cambiar mucho durante el desarrollo del mismo, ya que surgen nuevas ideas, nuevas aproximaciones para resolver el problema, se decidió usar la metodología Scrum, que está pensada para este tipo de entornos, en los que los requisitos cambian y es difícil hacer una planificación.
2. Haya sido necesario adquirir muchos nuevos conocimientos.

4.2 Mejoras respecto al año pasado

Uno de los objetivos del proyecto era mejorar el del año pasado. Esto presentó algunos problemas, ya que arreglar ciertos aspectos no ha sido tan sencillo.

Las mejoras introducidas han sido:

1. **Mejoras sobre el código:** se ha mejorado el código en general, procurando que sea más claro y evitando ciertos defectos de código, en la medida de lo posible. Se han refactorizado, por ejemplo, algunas clases en las que había métodos exageradamente largos, procurando dividirlos en métodos más pequeños, lo que mejora mucho la legibilidad.

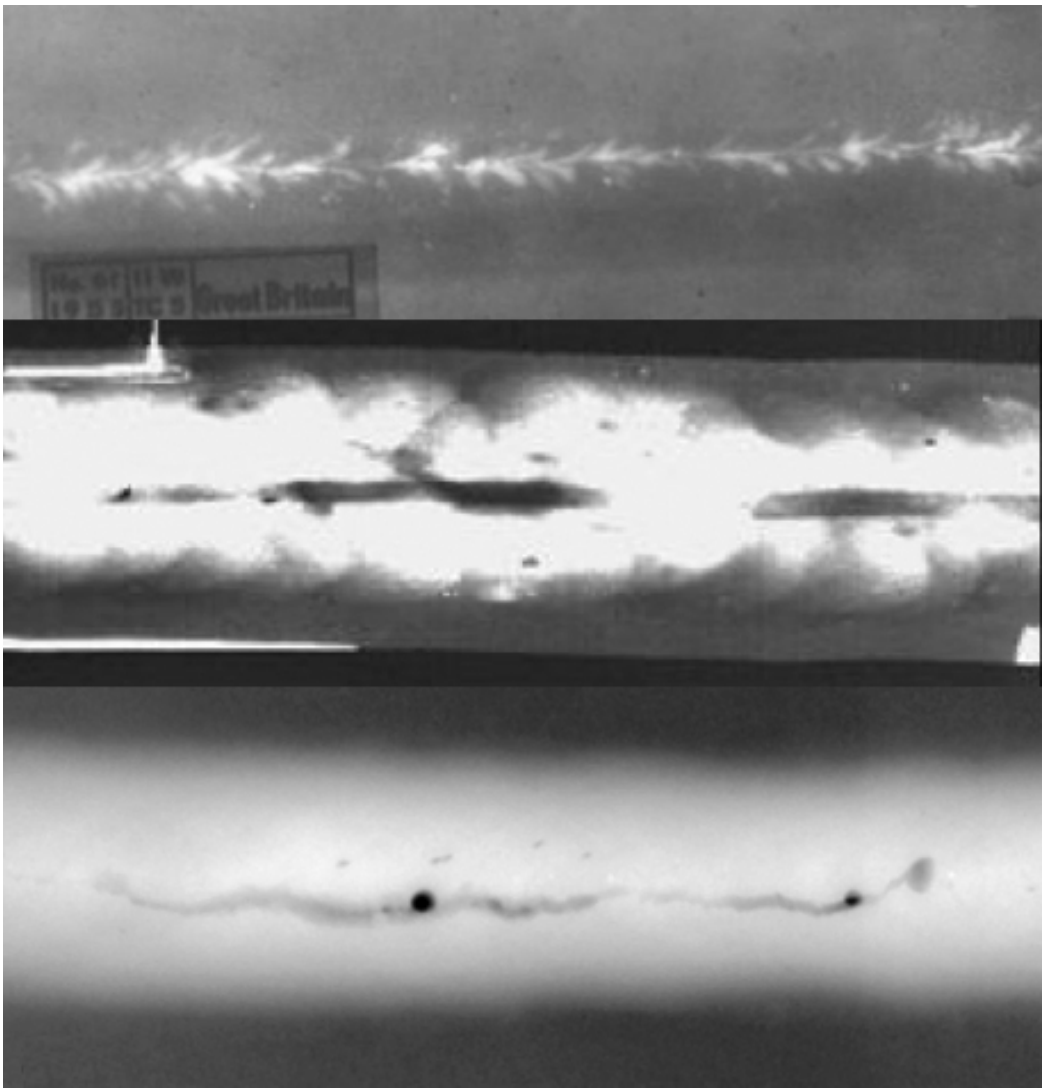

Figura 4.1.: Radiografías usadas en otros artículos[5] [36] [38]

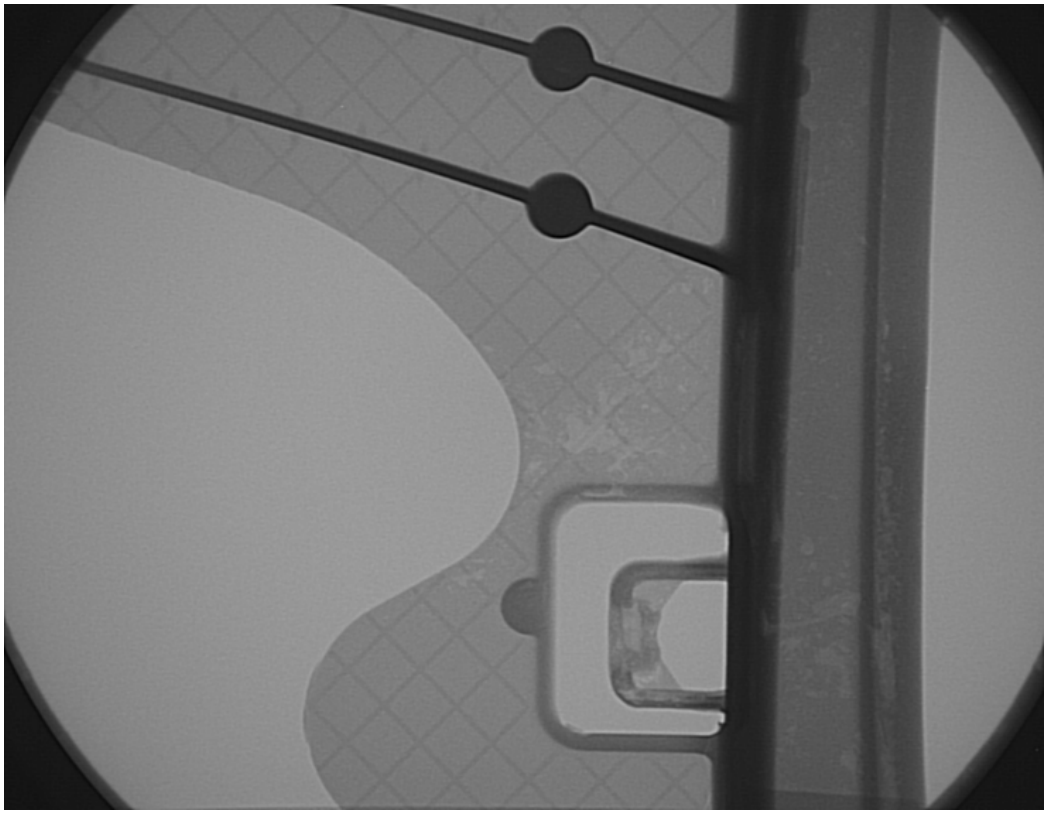

Figura 4.2.: Ejemplo de radiografía usada en nuestro proyecto

2. **Mejoras en la estructura:** se ha cambiado completamente la estructura de la aplicación, introduciendo un diseño en tres capas y algunos patrones de diseño que permiten mejorar el mantenimiento y la extensibilidad de la aplicación.
3. **Mejoras de rendimiento:** se ha intentado mejorar el rendimiento general de la aplicación. Básicamente, esto lo hemos conseguido mediante el proceso paralelo de los distintos trozos de la imagen mediante multihilo, aunque también se han cambiado algunos cálculos para que sean más eficientes.
4. **Mejoras en la interfaz:** la interfaz se cambió casi completamente, buscando una mejor intuitividad para el usuario. Por ejemplo, los botones son ahora más claros y se desactivan cuando no se pueden usar, cosa que antes no pasaba y podía llegar a causar problemas.
5. **Mejoras en la documentación y ayuda:** se ha mejorado la calidad de la API, extendiéndola a todos los elementos del código y arreglando fallos del lenguaje. Además, se ha añadido un módulo de ayuda en línea para permitir al usuario consultar cualquier duda de una forma rápida y sencilla [AÚN NO ESTÁ HECHO].
6. **Mejoras en la precisión:** se ha buscado mejorar la detección de defectos implementando nuevas aproximaciones y mejorando la que ya existía.

En definitiva, se ha intentado que, a partir de la buena base que representaba el proyecto del año pasado, se pueda ampliar esta idea de una forma mucho más sencilla. Se ha buscado, por tanto, crear una aplicación mucho más fácil de ampliar, ya que este proyecto es candidato a recibir una innumerable cantidad de mejoras prometedoras en un futuro.

4.3 Conocimientos adquiridos

Durante el desarrollo del proyecto se han ido aprendiendo y perfeccionando distintas disciplinas, las cuales aparecen detalladas a continuación.

4.3.1 Minería de datos

Al inicio del proyecto el conocimiento sobre la minería de datos estaba limitado a los conocimientos adquiridos en la asignatura de Minería de Datos de 5º curso de Ingeniería Informática.

Por este motivo cuando se expuso el proyecto se entendió como un reto y una manera de poder aprender sobre esta interesante rama de la informática en la que entran en juego grandes volúmenes de datos.

De este modo y a base de leer artículos, se han asimiladoy refinado multitud de conceptos y técnicas.

4.3.2 Weka

En la asignatura de Minería de Datos, de la que ya hemos hablado en el apartado anterior, se utilizó *Weka* para realizar las prácticas. Es por ello que ya teníamos algunos conocimientos sobre esta herramienta.

Este proyecto nos ha permitido usar *Weka* de una forma que no habíamos considerado hasta ahora, y es incluir alguno de sus métodos dentro de nuestra propia aplicación, aprovechando las posibilidades que nos brinda la herramienta.

4.3.3 Metodología Scrum

Las metodologías ágiles han adquirido un gran éxito dentro del desarrollo de software. Por este motivo el proyecto de final de carrera se presentaba como una buena base sobre la que aplicar una de estas metodologías y aprender de ella.

Se eligió *Scrum* por el hecho de que está pensado para entornos en los que cambian los requisitos y es difícil hacer una buena planificación.

Además, *Scrum* se ha explicado en la asignatura de Planificación y Gestión de Proyectos de 4º curso de Ingenería Informática. De este modo la realización del proyecto bajo esta nueva metodología ha aportado una experiencia adicional a los desarrollos clásicos en cascada que son utilizados en multitud de empresas.

4.3.4 Programación multihilo

Para poder mejorar el rendimiento de la aplicación, se hizo necesaria la programación multihilo. Ya poseíamos algunos conocimientos de algunas asignaturas de la carrera, pero nunca lo habíamos usado en Java. Por ello, ha sido necesario leer documentación al respecto y lidiar con algunos problemas que pueden presentar este tipo de aplicaciones.

4.3.5 Documentación en LaTeX

Al principio, la temida documentación se iba a desarrollar con uno de los clásicos compositores de texto, pero pronto nuestros tutores nos recomendaron utilizar encarecidamente la herramienta LATEX [30]. Este hecho, que a priori parecía un reto sencillo, se convirtió en un proceso de cierta complejidad y con una curva de aprendizaje larga e intensa.

En estos momentos damos gracias a nuestros tutores por empeñarse en convencernos a utilizar LATEX, ya no solo por el resultado estético que se obtiene, si no por darnos un nuevo reto a superar que añade valor a la realización de éste proyecto y la documentación, además de el valor para el futuro laboral.

Hemos utilizado una plantilla creada por el alumno Álvar Arnáiz González en el proyecto «Biblioteca de algoritmos de selección de instancias y aplicación orientada a su docencia» [74]. Futuros alumnos podrán disfrutarla y mejorarla. Tarde o temprano, el hecho de que cada año nuevo alumnos utilicen y mejoren esta plantilla, hará que la Universidad de Burgos tenga una plantilla estándar para la realización de cualquier memoria escrita en LaTeX.

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Anexo I - Plan del proyecto software

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

A. PLAN DEL PROYECTO SOFTWARE

A.1 Introducción

En este anexo se detalla el estudio desde el punto de vista temporal y de la viabilidad del proyecto software.

La planificación es una de las tareas más importantes en el desarrollo de un proyecto software y servirá para determinar objetivos, evaluar la viabilidad del proyecto, priorizar actividades...

En la primera parte del anexo se detallará la planificación temporal del proyecto teniendo en cuenta la metodología ágil que se va a utilizar: *Scrum*. En esta fase se determinarán los elementos que forman el *Product Backlog* y la prioridad de cada uno de ellos.

Debido a la metodología empleada, no se utilizará el clásico diagrama de *GANTT*. En lugar de esto, se definirá el *Product Backlog* y para el seguimiento se utilizará una herramienta de gestión especializada en metodologías ágiles, *PivotalTracker*, que permite el seguimiento diario de las tareas por parte del equipo de desarrollo.

En la segunda parte se calcularán los costes, analizando la rentabilidad del proyecto y justificando su desarrollo desde diversos puntos de vista: viabilidad técnica, legal, económica...

A.2 Planificación temporal del proyecto

Como se explicó en la memoria, para el desarrollo del proyecto de final de carrera se va, a utilizar una metodología ágil llamada *Scrum*. Esta metodología establece una serie de prácticas que serán llevadas a cabo con algunas limitaciones debido al reducido tamaño del equipo de desarrollo.

Para poder estimar de manera general el tiempo total que va a llevar el desarrollo del proyecto se va a realizar una estimación a partir de los casos de uso definidos en el Anexo II.

A.2.1 Estimación temporal a partir de casos de uso

Antes de mostrar la tabla de estimación temporal, conviene repasar una serie de fórmulas que se utilizan para calcular algunos valores de la tabla:

- Puntos de casos de uso no ajustados:

$$UUCP = \text{PesoDeActores} + \text{PesoDeCasosDeUso}$$

- Peso de los casos de uso:

$$\text{PesoDeCasosDeUso} = \sum_{i=0}^{i} Factor_i$$

- Peso de los factores técnicos:

$$TFC = 0,6 + 0,01 \cdot \sum_{i=0}^{i} Factor_i \cdot Peso_i$$

- Factores de entorno:

$$EF = 1,4 + (-0,03) \cdot \sum_{i=0}^{i} Factor_i \cdot Peso_i$$

A continuación (ver tabla A.1), aparece detallada la duración estimada del desarrollo del proyecto a partir de los casos de uso identificados. Se encuentra dividida en:

- Puntos de casos de uso no ajustados: sirven para conocer la envergadura del proyecto tomando como referencia los casos de uso.
- Factores técnicos: cuantifica la dificultad del proyecto en función de sus características internas.
- Factores del entorno: sirven para valorar lo familiarizado que se encuentra el equipo de desarrollo con proyectos de este tipo.

Los puntos de casos de uso se calculan según la siguiente fórmula:

$$UCP = UUCP \cdot TF \cdot EF$$

Para calcular los puntos de casos de uso hay que sustituir, en la fórmula anterior, con los valores de la tabla, es decir:

$$UCP = 68 \cdot 0,89 \cdot 0,95 = 57,494$$

Factor	Peso	F_i	Total
Actores simples	0	1	0
Actores medios	0	2	0
Actores complejos	1	3	3
Total peso actores			3
Casos de uso simples	5	5	25
Casos de uso medios	1	10	10
Casos de uso complejos	2	15	30
Total peso casos de uso			65
UUCP (Puntos de caso de uso no ajustados)			68
Sistema distribuido	1	2	2
Tiempos de respuesta críticos	1	1	1
En línea	1	1	1
Procesos internos complejos	5	1	5
El código debe ser reutilizable	3	1	3
Fácil de instalar	1	0,5	0,5
Fácil de utilizar	5	0,5	2,5
Portable	1	2	2
Fácil de modificar	4	1	4
Concurrencia	1	1	1
Incluye características de seguridad	1	1	1
Acceso a software creado por otras compañías	1	1	1
Incluye facilidades de aprendizaje para usuario	5	1	5
TF (Factores técnicos)			0,89
Familiarizado con *Scrum*	1	0,5	0,5
Experiencia en este tipo de aplicaciones	2	1	2
Experiencia en Orientación a Objetos	5	0,5	2,5
Capacidad de liderazgo del analista	5	1	5
Motivación	5	1	5
Requisitos estables	3	2	6
Trabajadores a tiempo parcial	5	-1	-5
Lenguaje de programación difícil de utilizar	1	-1	-1
EF (Factores de entorno)			0,95

Tabla A.1.: Estimación temporal a partir de casos de uso

Para obtener la duración del proyecto estimado según los casos de uso hay que multiplicar el valor de *UCP* por un factor que depende del número de factores de entorno (*EF*) a los cuales se les haya dado peso 0. En este caso, como no se ha dado ningún valor cero se multiplica por 20.

Por lo expuesto anteriormente: $N^{o}horas = 20 \cdot 57,494$, es decir, 1150 horas/hombre. [CAMBIAR]

Conviene destacar que los resultados obtenidos son meramente orientativos, ya que se va a utilizar una metodología ágil, por lo que la planificación va a ser a nivel de *Sprint* (de 15 a 30 días). Al comenzar cada uno de ellos, se definirán una serie de tareas que deberán ser completadas en dicho *Sprint*.

El proyecto comenzará en noviembre de 2012 y se desea finalizar en junio de 2013, es decir, se trabajará durante 7 meses. Se deben realizar los cálculos de horas para comprobar si es viable el desarrollo en las fechas previstas:

$$\text{Tiempo} = \frac{1150\,\text{horas}}{1\,\text{persona}} \cdot \frac{1\,\text{jornada}}{7\,\text{horas}} \cdot \frac{1\,\text{mes}}{20\,\text{jornadas}} = 8,21\,\text{meses}$$

El análisis de casos de uso junto con el horario marcado hace que la fecha de finalización deba trasladarse hasta julio para poder cumplir los plazos. De este modo la fecha de finalización del proyecto se retrasa hasta julio de 2010, manteniendo la jornada laboral de 35 horas semanales.

[TODO ESTO HAY QUE CAMBIARLO, PERO NOS SIRVE DE PLANTILLA]

A.3 Aplicando una metodología ágil: *SCRUM*

En esta sección se repasa de una manera rápida los principales aspectos del desarrollo de proyectos con *Scrum*. Al mismo tiempo se analizan los distintos roles que va a asumir cada uno de los participantes en el proyecto de final de carrera.

Conviene aclarar que los términos que define la metodología *Scrum* se encuentran en inglés, han sido mantenidos en el idioma original para no confundir con las traducciones.

Scrum [**?**] es una metodología para la gestión y desarrollo de proyectos software basada en un proceso iterativo e incremental. Cada iteración termina con una pieza de software ejecutable que incorpora una nueva funcionalidad o mejora las ya existentes. Estas iteraciones suelen durar de dos a cuatro semanas.

Scrum busca priorizar los trabajos que mayor valor aportan al negocio evitando, en la medida de lo posible, complejos manuales de documentación que no tengan utilidad en el proceso.

Los requerimientos y prioridades se revisan y ajustan durante el proyecto en intervalos cortos y regulares. De esta forma es sencillo adaptarse a los cambios solicitados por el cliente y responder de una forma rápida a los mismos.

A.3.1 Actores

Scrum define un conjunto de roles o actores detallados a continuación:

- *Scrum Master* (o facilitador): coordina el desarrollo del proyecto y trabaja de manera similar al director de proyectos. Una de sus tareas es eliminar los obstáculos que puedan dificultar al equipo de desarrollo o *team* la consecución de sus objetivos. En el proyecto, este rol lo han tomado los autores del mismo: Adrián González y Joaquín Bravo.
- *Scrum Team* (o equipo): incluye a los desarrolladores y son los encargados de realizar las tareas que se definen en cada *Sprint*. En el proyecto, este rol lo han tomado los autores del mismo.
- *Product Owner* (o cliente): representa la voz del cliente y aporta la visión del negocio. Representa el destinatario final del proyecto a desarrollar y el que, en última instancia, realiza las pruebas de aceptación. Es también el encargado de mantener al día el listado de las tareas o *Product Backlog* y sus prioridades. En el proyecto este rol sería tomado por la Universidad de Burgos, en concreto, personalizado en los tutores del proyecto César I. García Osorio y José Francisco Díez Pastor.

A.3.2 Ciclo de desarrollo

Al inicio del proyecto se definen una serie de requisitos que serán los objetivos a cumplir. Todos ellos quedan reflejados en el *Product Backlog*. Cada uno de estos objetivos serán subdivididos en tareas pequeñas y atómicas al inicio de cada *Sprint*.

El ciclo de desarrollo, como se muestra (ver figura **??**), es iterativo a nivel de *Sprint*, al inicio del mismo se extraen una serie de tareas de los elementos del *Product Backlog* que conforman el *Sprint Backlog*. Estas tareas deberán ser completadas durante el ciclo.

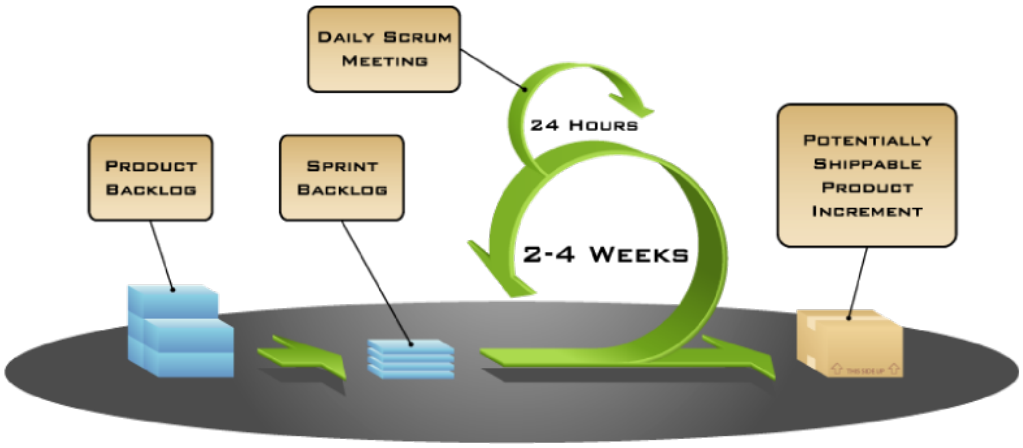

Figura A.1.: Metodología Scrum

Scrum define que debe realizarse una breve reunión diaria *Daily Meeting* en la que, cada miembro del equipo de desarrollo, explica lo que ha hecho desde la última sincronización, que va a hacer a partir de ese momento y las dificultades encontradas o que espera encontrar. En este caso, debido a que el equipo de desarrollo está formado por un único desarrollador, se va a omitir esta reunión y se sustituye por una pequeña reflexión interior en la que analizar las tareas a desarrollar durante el día.

Los *Sprints* se definirán, como establece *Scrum*, con una duración de dos a cuatro semanas que podrán alargarse o contraerse para coincidir con los tutores para las reuniones periódicas que establece:

- *Planning meeting*: reunión inicial de cada *Sprint* en la que se extraen los *item backlog*. Se numeran las tareas extraídas del *Product Backlog* para desarrollar durante el *Sprint*. Esta reunión tiene una gran importancia ya que, durante el desarrollo del *Sprint*, no se podrá modificar ni añadir nuevas tareas.
- *Review meeting*: reunión final (4 horas máximo) de cada *Sprint* donde se detallan los objetivos cumplidos durante el mismo. Se muestra al usuario el producto, en caso de que sea posible. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, replanificando el proyecto.
- *Retrospective meeting*: en esta reunión (4 horas máximo), el equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad. El *Scrum Master*, o *Facilitador*, se encargará de ir eliminando los obstáculos identificados.

En este caso, todas las reuniones requieren de la presencia de los tutores y del equipo de desarrollo ya que entre ambos, agrupan los roles que se ven implicados en dichas reuniones.

A.4 Product Backlog del proyecto

En esta sección, aparece redactado el *Product Backlog* (ver tabla A.2) o lista de objetivos, con la prioridad de cada una de los *Product Item*, el *sprint* al que pertenecen y la estimación que les dimos en su momento, basándonos en una escala de puntos tipo Fibonacci (1, 2, 3, 5, 8). Hemos decidido incluir también los *bugs* que se han ido localizando[INSERTAR CITA].

Scrum no define un método o herramienta para llevar el día a día del trabajo realizado, por lo que el seguimiento se puede realizar con una simple hoja de cálculo, o con herramientas de gestión especializadas en este tipo de desarrollos.

En este caso, se ha utilizado *PivotalTracker* (https://www.pivotaltracker.com/) para el seguimiento. Es un software potente al que se accede mediante un navegador y sirve de control de tareas y defectos. Su licencia ha sido gratuita por tratarse de un proyecto público.

ID	Backlog Item	Propietario	Prioridad	Estimación	Sprint
S-01001	Se debe poder cargar una imagen y sacar información de sus píxeles	César, José	Alta	2	Sprint 1
S-01002	Se debe poder analizar una imagen y extraer sus características	César, José	Alta	5	Sprint 2
S-01003	La aplicación debe ser capaz de calcular una imagen Saliency Map	César, José	Alta	5	Sprint 3
S-01004	La aplicación debe poder calcular las características de Haralick	César, José	Alta	8	Sprint 3
S-01005	La aplicación debe ser capaz de generar ficheros ARFF	César, José	Alta	5	Sprint 3
S-01006	La aplicación debe ser capaz de calcular los eigenvalues mediante weka.Matrix o COLT	César, José	Baja	1	Sprint 4
S-01007	La aplicación debe ser capaz de dividir la imagen según los hilos disponibles, con solapamiento entre las mismas	César, José	Alta	1	Sprint 4
S-01008	La aplicación debe ser capaz de realizar la convolución de la imagen completa, para extraer la media	César, José	Media	3	Sprint 4

continúa en la página siguiente

continúa desde la página anterior

ID	Backlog Item	Propietario	Prioridad	Estimación	Sprint
S-01009	La aplicación debe ser capaz de albergar varios tipos de ventana	César, José	Alta	1	Sprint 4
S-01010	La aplicación debe ser capaz de entrenar un clasificador y clasificar	César, José	Alta	5	Sprint 4
S-01011	La aplicación debe ser capaz de mostrar un GUI con las principales opciones	César, José	Alta	5	Sprint 4
S-01012	La aplicación debe mostrar el progreso en una Progress Bar	César, José	Baja	2	Sprint 5
S-01013	La aplicación debe mostrar un aviso si no se selecciona una región de la imagen a analizar	César, José	Baja	1	Sprint 5
S-01014	La ventana de resultados se debe limpiar después de cada análisis	César, José	Media	1	Sprint 5
S-01015	La aplicación debe escribir un ARFF por cada hilo, uniéndolos al terminar el proceso	César, José	Alta	3	Sprint 5
S-01016	La aplicación debe mostrar eventos de funcionamiento en la interfaz	César, José	Media	2	Sprint 5
S-01017	La aplicación debe guardar y cargar en un fichero las opciones	César, José	Media	3	Sprint 5
S-01018	La aplicación debe ser capaz de guardar un log de errores	César, José	Baja	2	Sprint 5
S-01019	La aplicación debe usar el filtro saliency para calcular características para entrenar y clasificar	César, José	Alta	3	Sprint 5
S-01020	La aplicación debe usar la convolución de la imagen completa, en el caso de que no se elija región, o toda la región más un margen	César, José	Alta	3	Sprint 5
S-01021	La aplicación debe poder dibujar los defectos encontrados	César, José	Alta	3	Sprint 5
S-01022	La aplicación debe ser capaz de usar varios tipos de métodos para determinar si una ventana es defectuosa o no	César, José	Alta	8	Sprint 5

continúa en la página siguiente

continúa desde la página anterior

ID	Backlog Item	Propietario	Prioridad	Estimación	Sprint
S-01023	La aplicación debe mostrar un slider para seleccionar el umbral de detección	César, José	Alta	3	Sprint 5
S-01024	La aplicación debe ser capaz de escribir texto formateado en HTML en el panel de log	César, José	Baja	3	Sprint 6
S-01025	La aplicación debe ser capaz de exportar el log en formato HTML	César, José	Baja	2	Sprint 6
S-01026	La aplicación debe tener un menú con varias opciones	César, José	Media	2	Sprint 6
S-01027	La aplicación debe usar REPTree para regresión lineal	César, José	Media	1	Sprint 6
S-01028	La aplicación debe ser capaz de calcular filtros de umbrales locales y comprobar si las regiones candidatas están dentro de la máscara	César, José	Alta	5	Sprint 6
S-01029	La aplicación debe dirigir la búsqueda de defectos usando filtros de umbrales locales	César, José	Alta	5	Sprint 6
B-01001	BUG: la primera y segunda derivadas deben estar bien calculadas	César, José	Alta	-	Sprint 6
S-01030	La aplicación debe dar la oportunidad al usuario de elegir el proceso de detección de defectos	César, José	Media	2	Sprint 7
S-01031	La aplicación debe calcular los umbrales locales con un pequeño margen	César, José	Media	2	Sprint 7
S-01032	La aplicación debe ser capaz de generar un conjunto de datos a partir de la ventana deslizante	César, José	Alta	3	Sprint 7
S-01033	La aplicación debe poder remuestrear la lista de píxeles blancos, de tal manera que no considere todos	César, José	Alta	5	Sprint 7

continúa en la página siguiente

continúa desde la página anterior

ID	Backlog Item	Propietario	Prioridad	Estimación	Sprint
S-01034	La aplicación debe ser capaz de calcular características geométricas sobre las regiones segmentadas	César, José	Alta	5	Sprint 7
S-01035	Se debe poder mostrar una tabla resumen de los defectos	César, José	Alta	5	Sprint 7
S-01036	La aplicación debe permitir al usuario qué tipo de clasificación se va a usar: clases nominales o regresión	César, José	Media	2	Sprint 8
S-01037	La aplicación debe añadir un borde a la selección en aquellas zonas que no coincidan con los bordes de la imagen	César, José	Media	3	Sprint 8
S-01038	La aplicación debe permitir al usuario qué tipo de ventana defectuosa quiere usar	César, José	Media	3	Sprint 8
S-01039	Se debe añadir un módulo de ayuda en línea (JavaHelp), accesible desde el menú y con teclas rápidas (F1)	César, José	Media	3	Sprint 8
S-01040	La aplicación debe poder permitir seleccionar una fila de la tabla de resultados e iluminar el defecto correspondiente	César, José	Alta	5	Sprint 8
S-01041	La aplicación debe dar la opción de elegir si se calculan todas las características o sólo mejores	César, José	Media	3	Sprint 8
B-01002	BUG: la aplicación debe impedir la reselección mientras se analiza	César, José	Alta	-	Sprint 8

Tabla A.2.: Product Backlog

A.5 Planificación por Sprint

En esta sección aparece el listado de los diversos *Sprints* del proyecto. La información de cada uno está dividida del siguiente modo:

- *Planning meeting*: acta de la reunión inicial de cada *Sprint* en la que se detallan los objetivos a cumplir durante el transcurso del mismo.
- *Sprint planning*: se numeran las tareas extraídas del *Product Backlog* para desarrollar durante el *Sprint*.
- *Burndown chart*: gráfico que muestra la evolución a lo largo del *Sprint* en comparación a la línea óptima.
- *Retrospective meeting*: acta de la reunión retrospectiva realizada al final del *Sprint* y donde se analizan los problemas detectados durante su ejecución y las desviaciones (en caso de haberlas).

Debido a la falta de conocimiento inicial y a la incertidumbre sobre el desarrollo, se inicia el proyecto con *Sprints* de corta duración, aproximadamente dos semanas. Se ha intentado mantener siempre esta duración, aunque a veces nos ha podido el optimismo y hemos metido demasiado trabajo o no se han tenido en cuenta ciertos problemas de tiempo.

Las reuniones inicial y final de cada *Sprint* se solaparon para poder hacerlas el mismo día, debido a problemas de tiempo y de disponibilidad. Así, media parte de esta reunión se centraba en la *Planning meeting* y la otra en la *Retrospective meeting*.

A.5.1 Sprint 1: Spike Arquitectónico 1

■ Planning meeting

En este primer *Sprint* se pretende comenzar el proyecto, empezar a usar ImageJ, familiarizarnos con el proyecto del año pasado... Se empiezan a poner en práctica algunas de las ideas que teníamos para el diseño de la aplicación. Lo llamamos *spike arquitectónico* debido a que es una primera aproximación al proyecto, tomando prestado el nombre de otra metodología ágil (*Extreme Programming*).

■ Fechas de desarrollo y tiempo estimado

Este primer *Sprint* comienza el 19/11/2012 y su fecha límite es el 27/11/2012. Es un *Sprint* muy corto debido a que es una primera aproximación. Los puntos totales estimados en este *Sprint* son de 2, con lo pensamos que vamos a dedicarle 4 horas/hombre.

■ Sprint planning

- S-01001: Se debe poder cargar una imagen y sacar información de sus píxeles.
  - TK-00001: Carga y manejo de la imagen con ImageJ (1.5 horas).
  - TK-00002: Creación de la ventana deslizante (simple) (1.5 horas).

- TK-00003: Creación de una pequeña interfaz que muestre los resultados (1 hora).

■ Burndown chart

Para este *Sprint* no disponemos de *Burndown Chart* debido a un fallo en el uso de *Pivotal Tracker*. Como aún no estábamos familiarizados con él, finalizamos antes la *release* que la historia de usuario, con lo que no se ve reflejado en el *Burndown Chart*.

■ Retrospective meeting

¿Qué ha ido bien?

- La lectura de artículos y comprensión del proyecto avanza por buen camino.
- Nos vamos familiarizando con ImageJ y el resto de nuevas herramientas.

¿Qué dificultades ha habido?

- Familiarizarse con ImageJ es difícil. Por ello, llegamos a plantearnos el uso de alguna otra librería, como *OpenCL*, pero acabamos descartándolo debido a que ImageJ tienen muy buena documentación, cosa que el resto no.
- Adaptación a la nueva metodología.

A.5.2 Sprint 2: Spike Arquitectónico 2

■ Planning meeting

Como seguimos probando cosas más que implementar funcionalidad nueva, consideramos a este *Sprint* otro *spike*, aunque este está ya más orientado a nueva funcionalidad. Básicamente, lo que buscamos en este *Sprint* es:

1. Seguir familiarizándonos con ImageJ.
2. Convertir la aplicación en multihilo, dividiendo la imagen a analizar en tantas partes como procesadores tenga la máquina y analizando cada trozo en un hilo separado.
3. Implementar los algoritmos de extracción de características, usando lo que se pueda del proyecto del año pasado.

También comentamos otras cosas como que, de momento, vamos a sacar los resultados por pantalla, pero que hay que ir pensando en sacarlos a un fichero. También tenemos que pensar que hay que usar *Saliency Map* para el preprocesamiento.

■ Fechas de desarrollo y tiempo estimado

Este *Sprint* comienza el 27/11/2012 y la fecha límite es el 11/12/2012. El número total de puntos estimado es de 5, con lo que pensamos que vamos a dedicarle 12 horas/hombre.

Sprint planning

- S-01002: Se debe poder analizar una imagen y extraer sus características.
  - TK-00004: Implementación de la ventana deslizante multihilo (6 horas).
  - TK-00005: Implementación de los algoritmos de selección de características (5 horas).
  - TK-00006: Implementación de las llamadas a los algoritmos de selección de características y muestra de los resultados (1 hora).

Burndown chart

En el *Burndown Chart* (ver figura A.2) podemos ver 2 cosas:

1. Hay una incoherencia en las fecha de inicio, ya que, de nuevo, por la falta de conocimientos del uso de *PivotalTracker*, metimos una historia de usuario antes de tiempo y después nos limitamos a cambiar el nombre, dándonos cuenta mucho después de que lo habíamos hecho mal.
2. Tuvimos ciertos problemas de tiempo y de implementación, por lo que hay un tiempo totalmente plano en el que no pudimos terminar ninguna tarea, lo que llevó a pasarnos de fecha hasta el 18/12/2012.

Retrospective meeting

¿Qué ha ido bien?

- Cada vez dominamos más ImageJ.
- Hemos conseguido implementar con éxito el multihilo.
- Las implementaciones de los algoritmos de extracción de características del año pasado parece que van bien, por lo que decidimos mantenerlas.
- El diseño que habíamos pensado se está probando ser bueno.

¿Qué dificultades ha habido?

- Hemos planificado mal este *Sprint*, ya que fuimos muy optimistas respecto al tiempo y a las dificultades. Tuvimos mucha carga de trabajo de otras asignaturas.
- Tuvimos problemas con las coordenadas, lo que nos quitó mucho tiempo.

A.5.3 Sprint 3

Planning meeting

En este *Sprint* lo que se pretende es:

1. Implementar las características de Haralick, que por ser muy pesadas, no se hicieron en el anterior *Sprint*.

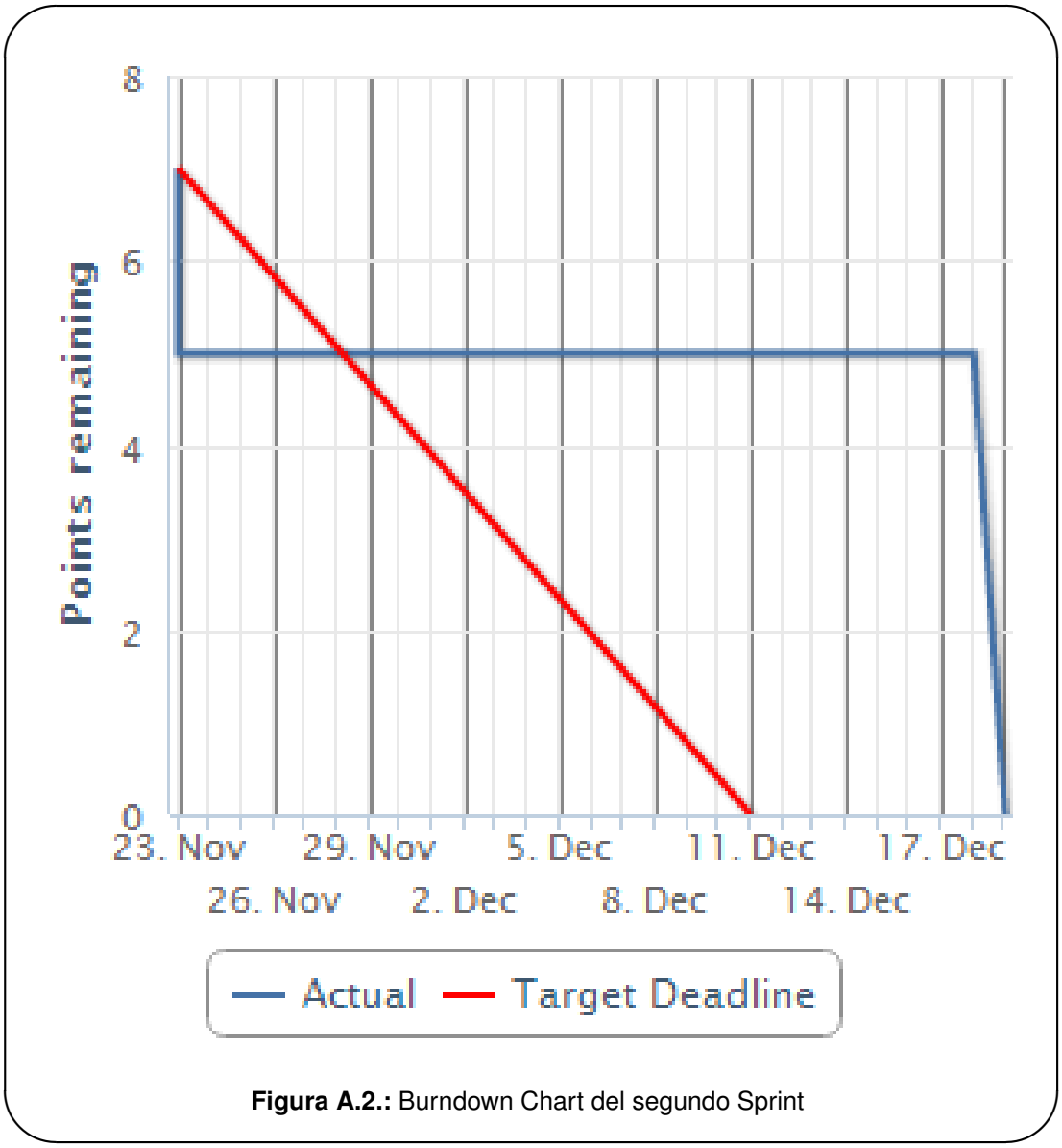

Figura A.2.: Burndown Chart del segundo Sprint

2. Calcular el *Saliency Map* para el preprocesamiento de la imagen.
3. Generar un fichero ARFF con el resultado de los análisis.

Como vemos, algunas de las cosas que se hablaron en anteriores reuniones se ponen en práctica en este *Sprint*.

Fechas de desarrollo y tiempo estimado

Este *Sprint* se planificó de manera especial, ya que por medio están las vacaciones de Navidad y el periodo de exámenes del primer cuatrimestre. Por eso es más largo de lo normal: del 19/12/2012 al 24/01/2013. El número total de puntos estimado es de 18 y creemos que vamos a dedicarle 20 horas/hombre.

Sprint planning

- S-01003: La aplicación debe ser capaz de calcular una imagen Saliency Map.
  - TK-00007: Implementar el cálculo de Saliency Map (3 horas).
  - TK-00008: Adición de los Saliency Map a la estructura del programa (0.5 horas).
- S-01004: La aplicación debe poder calcular las características de Haralick.
  - TK-00009: Implementar características de Haralick (7 horas).
  - TK-00010: Añadirlo a la estructura del patrón Estrategia de algoritmos (1 hora).
- S-01005: La aplicación debe ser capaz de generar ficheros ARFF.
  - TK-00011: Generación de ARFF a partir de los datos de las características (8.5 horas).

Burndown chart

En el *Burndown Chart* (ver figura A.3) podemos ver que hay un gran periodo totalmente plano, en el que no pudimos prácticamente hacer nada del proyecto. Esto estaba previsto y controlado, y por eso planificamos este *Sprint* con tanto tiempo.

Por lo demás, vemos que en este *Sprint* no nos pasamos de la fecha límte.

Retrospective meeting

¿Qué ha ido bien?

- Los nuevos algoritmos parecen funcionar bien lo que había el año pasado, por lo que decidimos mantenerlo (excepto una cosa, de la que hablaremos más tarde).
- El diseño que habíamos pensado se vuelve a probar ser bueno, ya que permite añadir nuevos algoritmos muy fácilmente.

¿Qué dificultades ha habido?

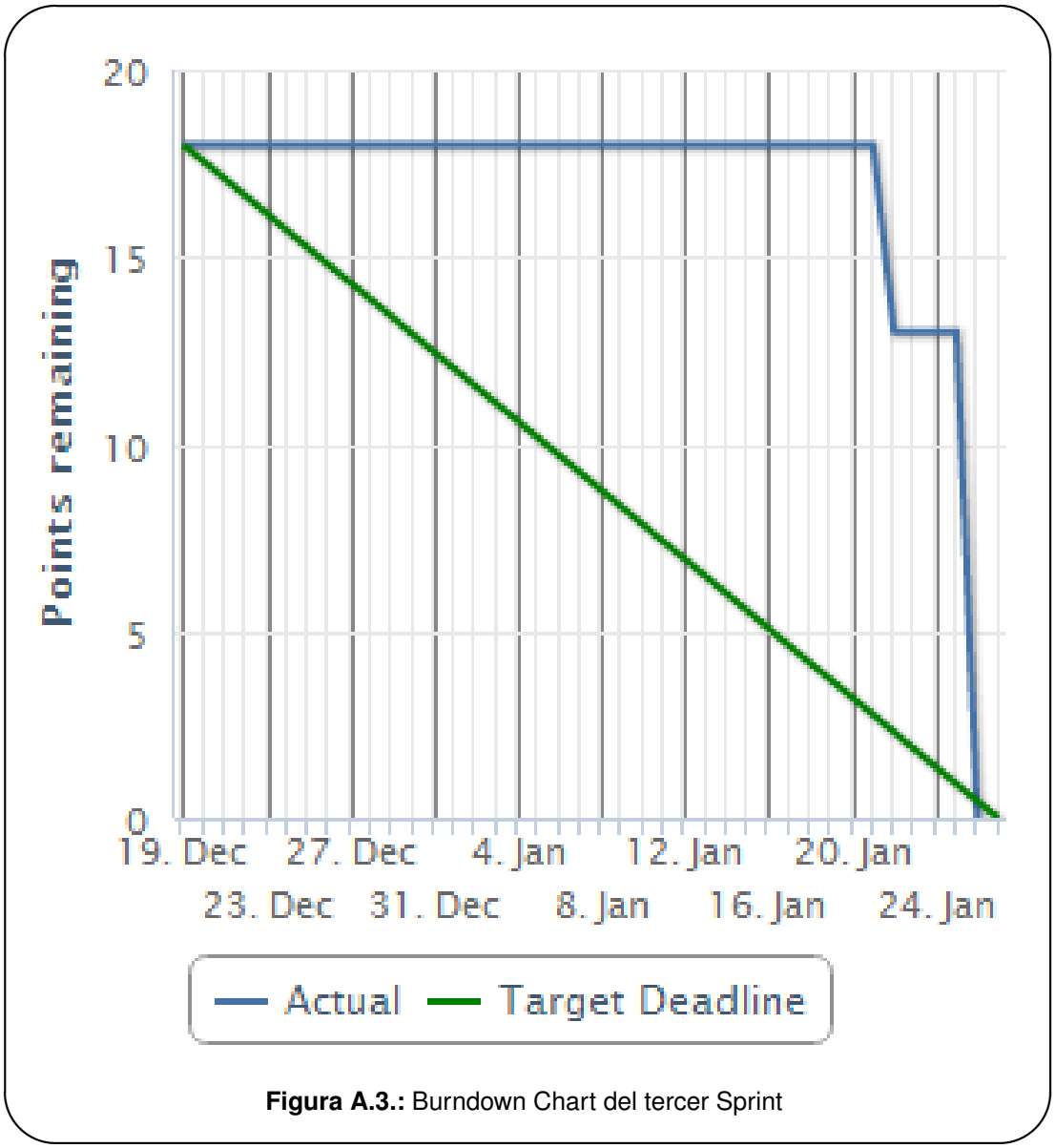

Figura A.3.: Burndown Chart del tercer Sprint

- La implementación de la última característica de Haralick es muy lenta. Hemos observado que los alumnos del año pasado hicieron su propia clase de cálculos de matrices, cosa que nos pareció absurda y contraproducente. De hecho, ellos anularon esta característica para presentar el proyecto.
- En este *Sprint* quizás nos pasamos estimando el tiempo de cada tarea, debido en parte a lo que nos pasó en el anterior *Sprint*.

A.5.4 Sprint 4

■ Planning meeting

En este *Sprint* planeamos hacer:

- Probar implementaciones alternativas para los autovalores de una matriz (Weka, COLT...).
- Añadir un solapamiento al dividir las imágenes para el multihilo, para tener en cuenta la importancia de los pixeles de la división, ya que pueden ser importantes en la otra parte de la imagen.
- Modificar la convolución de las imágenes para hacerlo con la imagen completa.
- Entrenar un clasificador, para lo que hace falta una ventana aleatoria.
- Implementar la interfaz de usuario.

Además, se habló de otras cosas, como el ir probando los umbrales adaptitivos en ImageJ, o cosas que deben ir en la memoria (la metodología debe aparecer en el apartado de técnicas, resaltar los aspectos relevantes y nuevos del proyecto de este año *versus* la versión del año pasado...). Llegamos incluso a hablar de la idea de clasificar los defectos en tipo, para lo cual es necesario calcular características geométricas.

■ Fechas de desarrollo y tiempo estimado

Este *Sprint* se inicia el 26/01/2013 y tiene como fecha límite el 15/02/2013. Tiene un total de puntos estimados de 20, con lo que planeamos dedicarle 27 horas/hombre.

■ Sprint planning

- S-01006: La aplicación debe ser capaz de calcular los eigenvalues mediante weka.Matrix o COLT.
  - TK-00012: Añadir las librerías necesarias (0.5 horas).
  - TK-00013: Sustituir la llamada a eigenvalues por el método correspondiente de las nuevas librerías (1.5 horas).
- S-01007: La aplicación debe ser capaz de dividir la imagen según los hilos disponibles, con solapamiento entre las mismas.

    - TK-00014: Modificar la división de las imágenes para permitir un solapamiento (1 hora).
- S-01008: La aplicación debe ser capaz de realizar la convolución de la imagen completa, para extraer la media.
    - TK-00015: Calcular la convolución de la imagen completa (1.5 horas).
    - TK-00016: Calcular sobre la convolución la primera y segunda derivadas (1.5 horas).
- S-01009: La aplicación debe ser capaz de albergar varios tipos de ventana.
    - TK-00017: Crear el patrón estrategia (1 hora).
    - TK-00018: Creación de la clase de la ventana aleatoria (1 hora).
- S-01010: La aplicación debe ser capaz de entrenar un clasificador y clasificar.
    - TK-00019: Crear 2 listas de píxeles centrales: una con defectos y otra sin defectos (2 horas).
    - TK-00020: Seleccionar de forma aleatoria ventanas sobre las que calcular características, a partir de las 2 listas (1 hora).
    - TK-00021: Entrenar a un clasificador y guardar el modelo obtenido (4 horas).
    - TK-00022: Clasificar defectos (2 horas).
- S-01011: La aplicación debe ser capaz de mostrar un GUI con las principales opciones.
    - TK-00023: Implementación de la GUI (8 horas).
    - YK-00024: Adaptación del resto del código (2 horas).

Burndown chart

En el *Burndown Chart* (ver figura A.4) podemos ver que el desarrollo fue bastante escalonado, teniendo periodos planos en los que no podíamos avanzar debido a las dificultades.

Todo esto al final hizo que nos pasáramos de fecha, ya que acabamos el 21 de febrero.

Retrospective meeting

¿Qué ha ido bien?

- El diseño que habíamos pensado se vuelve a probar ser bueno, ya que, usando la misma idea para las ventanas que para los algoritmos, nos permite añadir nuevos tipos de ventana fácilmente.
- El entrenamiento funciona bien. Esto nos permitió ganar mucha comprensión acerca de cómo funciona esta parte del proyecto.
- La GUI implementada es más intuitiva que la del año pasado.

¿Qué dificultades ha habido?

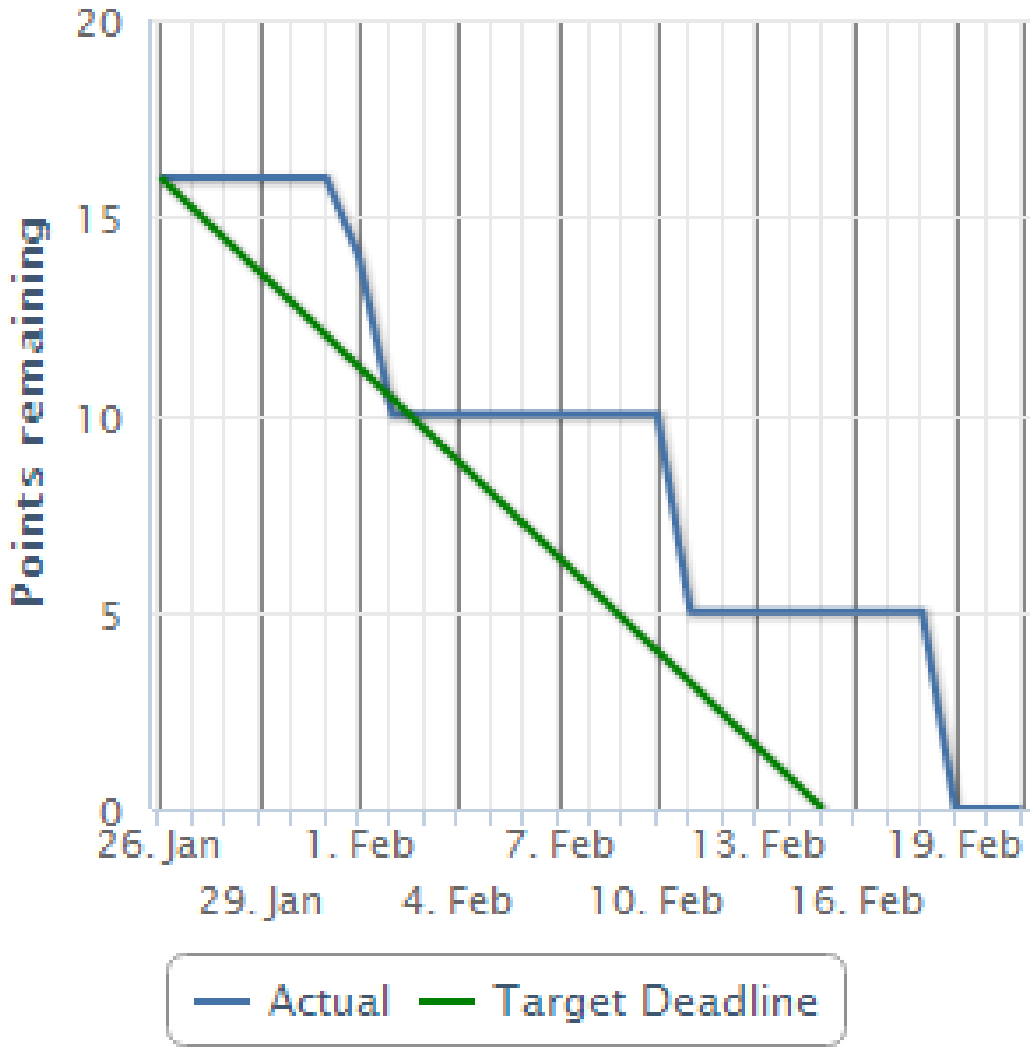

Figura A.4.: Burndown Chart del cuarto Sprint

- Los autovalores siguen siendo muy lentos.
- Mala planificación: nos pasamos metiendo historias en este *Sprint*, por lo que nos ha llevado algo más de tiempo del previsto.
- Fallo extraño al crear el ARFF de entrenamiento: hay veces que los hilos no lo hacen bien.

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Anexo II - Especificación de requisitos

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Anexo III - Especificación de diseño

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Anexo IV - Manual del programador

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Anexo V - Manual del usuario

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Apéndice A - Guía rápida de Version One

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor

Universidad de Burgos

Escuela Politécnica Superior

Ingeniería Informática

Área de Lenguajes y Sistemas Informáticos

Apéndice B - Licencia GNU GPL

X-RayDetector: Detección automática de defectos en piezas metálicas mediante análisis de radiografías

Adrián González Duarte y Joaquín Bravo Panadero

Directores: Dr. César I. García Osorio
José Francisco Díez Pastor