Avance 1 del proyecto

Daniel Montoya , Andrés Salazar

26 de Marzo de 2017

Índice

Resumen

El siguiente documento explica todos los detalles concernientes al primer avance del proyecto de segmentación de imagenes. Este primer avance incluye el estándar de codificación escogido por el equipo de desarrollo, las herramientas investigadas disponibles que ayudan a controlar el cumplimiento del mismo,el diagrama de componentes y el diagrama UML que detalla la primera iteración de la arquitectura del sistema y el conjunto de actividades de aseguramiento de calidad a realizar al finalizar cada sprint.

1. Estándar de Codificación

Al comenzar un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

1.1. Estándar de codificación Java

1.1.1. Organización de ficheros

Las clases se agruparán en paquetes. Estos paquetes se organizarán de manera jerárquica.

1.1.2. Fichero fuente java

Cada fichero fuente contendrá una única clase o interfaz pública. El nombre del fichero coincidirá con el nombre de la clase. Cuando existan varias clases privadas asociadas funcionalmente a una clase pública, podrán colocarse en el mismo fichero fuente que la clase pública.

1.1.3. Declaraciones

Se usará una declaración por línea, promoviendo así el uso de comentarios. Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente. Las declaraciones deberán situarse al principio de cada bloque principal en el que se utilicen y nunca en el momento de su uso.

1.1.4. Longitud de línea

La longitud de línea no superará los 80 caracteres por motivos de visualización.

1.1.5. División de líneas

Cuando una expresión ocupe más de una línea, esta se podrá dividir en función de los siguientes criterios:

- Después de una coma
- Antes de un operador
- Se alineará la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior.
- En caso de que las reglas anteriores generen código poco comprensible, se establecerán tabulaciones de 8 espacios.

1.1.6. Nomenclatura de paquetes

Se escribirán siempre en letras minúsculas para evitar que entren en conflicto con los nombres de clases o interfaces.

1.1.7. Nomenclatura de clases

Los nombres de las clases deberán ser sustantivos y deberán tener la primera letra en mayúscula. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúscula. Los nombres serán simples y descriptivos.

1.1.8. Nomenclatura de interfaces

Se nombrarán siguiendo los mismos criterios que los indicados para las clases. Con la excepción de que toda interfaz se nombrará con el prefijo “I” para diferenciarla de la clase que la implementa.

1.1.9. Nomenclatura de métodos

Deben ser verbos escritos en minúscula. Cuando el método este compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúscula.

1.1.10. Nomenclatura de variables

Se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúscula. Los nombres serán cortos y sus significados expresarán con claridad la función que desempeñan en el código.

1.1.11. Nomenclatura de constantes

Se escribirán en mayúsculas. Cuando los nombres de constantes sean compuestos, las palabras se separarán entre sí mediante el carácter "_".

1.1.12. Visibilidad de atributos de instancia y de clase

Los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases heredadas, en tales casos serán declarados como protegidos. El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes.

1.1.13. Asignación sobre variables

- Se evitarán las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia. Por ejemplo: int a = b = c = 2 ;
- No se utilizará el operador de asignación en aquellos lugares donde sea susceptible de confusión con el operador de igualdad. Por ejemplo: If ((c = d++) == 0) {}
- No se utilizarán asignaciones embebidas o anidadas. Por ejemplo: c = (c = 3) + 4 + d ;

1.1.14. Documentación : Javadoc

Se incluirá en la entrega de la aplicación la documentación de los ficheros fuente de todas las clases. Dicha documentación será generada por la herramienta "javadoc". La herramienta "javadoc" construirá la documentación a partir de los comentarios (incluidos en las clases) encerrados entre los caracteres "/**" y "/*" . Dentro de los comentarios "javadoc" se incluirán etiquetas especiales de documentación. Estas etiquetas comenzarán con el símbolo "@" , se situarán al inicio de la línea del comentario y permitirán incluir información específica de la aplicación de forma estándar. Se utilizarán las siguientes etiquetas:

- @author Nombre : Permite añadir información sobre el autor o autores del código.
- @version Información versión : Permite incluir información sobre la versión y fecha del código.
- @param nombre de parámetro descripción: Inserta el parámetro especificado y su descripción en la sección "Parameters:" de la documentación del método en el que se incluya . Este tag no puede utilizarse en comentarios de clase, interfaz o campo.
- @return Descripción: Inserta la descripción indicada en la sección "Returns:" de la documentación del método. Este tag debe aparecer en los comentarios de documentación de todos los métodos , salvo en los constructores y en aquellos que no devuelvan ningún valor.
- @trhows Nombre de la clase descripción : Añade el bloque de comentario "Trhows:" incluyendo el nombre y la descripción de la excepción especificada.
- @see Referencia : Permite incluir en la documentación la sección de comentario "see also:", conteniendo la referencia indicada. Además permite hacer referencia a la documentación de otras clases o métodos.
- @deprecated explicación : Esta etiqueta indica que la clase, interfaz, método o campo está obsoleto y que no debe utilizarse y que dicho elemento posiblemente desparecerá en futuras versiones.
- @since versión : Se utiliza para especificar cuándo se ha añadido a la API la clase, interfaz, método o campo. También se incluye el número de versión u otro tipo de información.

1.2. Herramientas para verificación de cumplimiento de estándar

1.2.1. PMD

Es un analizador de código que utiliza unos conjuntos de reglas para identificar problemas dentro del software. Detecta cosas como código duplicado, código muerto (variables, parámetros o métodos sin usar), complejidad de métodos (if innecesarios). Trabaja principalmente con lenguaje java.

1.2.2. Checkstyle

Es una herramienta de análisis estático de código que se utiliza para comprobar que el código analizado cumple con una serie de reglas de estilo. Por ejemplo , analiza el código según el estándar "Sun code Conventions" (mira las cabeceras, importaciones de paquetes, Javadoc, etc) .

1.2.3. Sonar

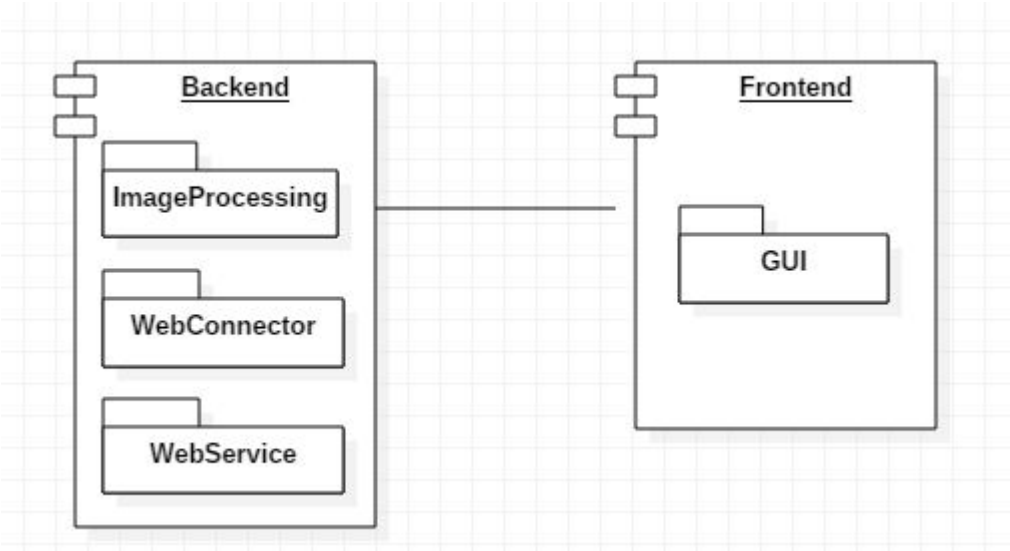
Es una herramienta de software libre y gratuita que permite gestionar la calidad del código fuente. Al instalarla se puede recopilar , analizar y visualizar métricas del código fuente. Permite visualizar informes con resúmenes de las métricas del proyecto. Trabaja principalmente para Java.

1.2.4. Google CodePro Analytix

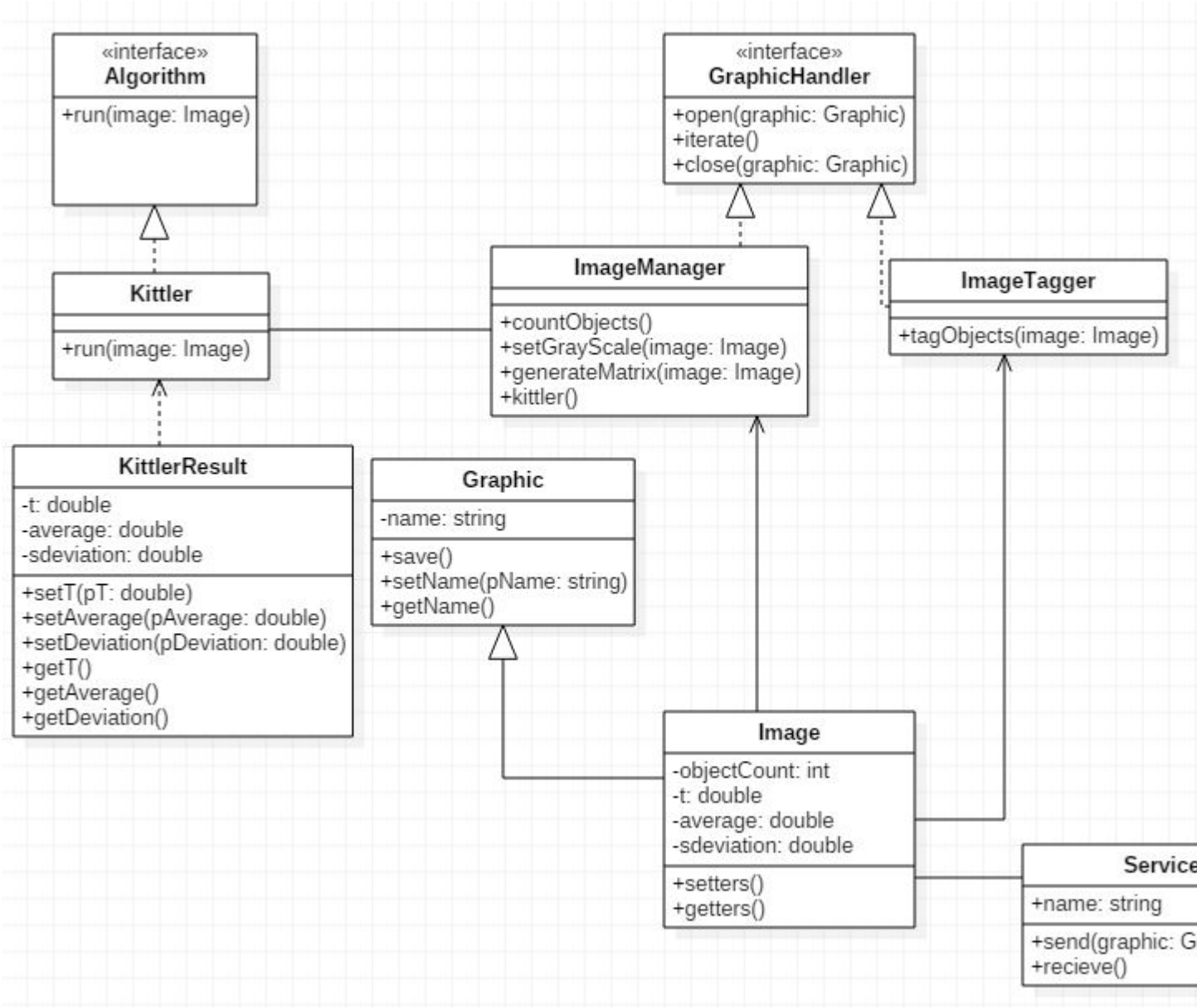
Ofrece un entorno para evaluación de código, métricas, análisis de dependencias , cobertura de código y generación de pruebas unitarias , mira las excepciones , refactorizaciones potenciales , convenios de Javadoc y métricas. Trabaja para Java concretamente en Eclipse.

2. Diagramas

2.1. Diagrama de componentes



2.2. Diagrama de clases (Lógica del sistema)



2.3. Patrón de diseño Strategy

2.3.1. Definición

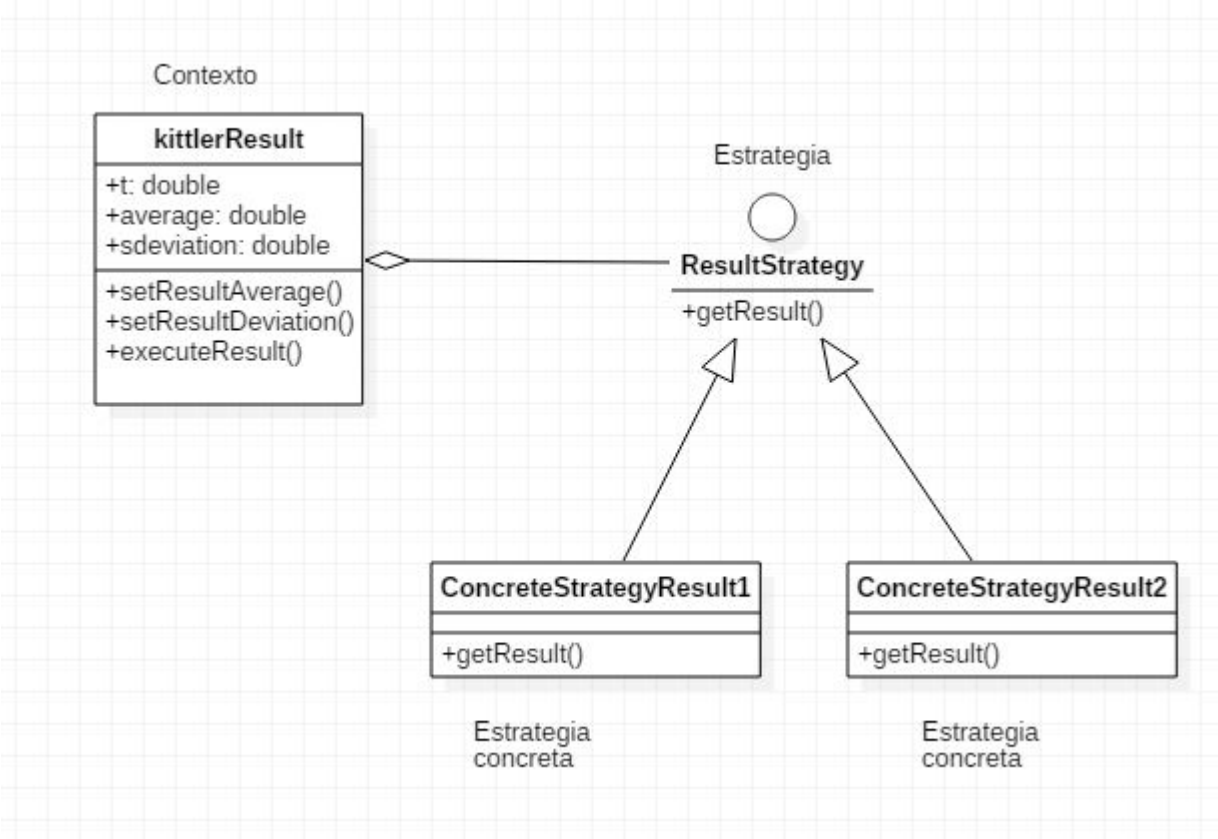
Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

2.3.2. Participantes

- Contexto: Es el elemento que usa los algoritmos, por tanto, delega en la jerarquía de estrategias. Configura una estrategia concreta mediante una referencia a la estrategia necesaria. Puede definir una interfaz que permita a la estrategia el acceso a sus datos en caso de que fuese necesario el intercambio de información entre el contexto y la estrategia. En caso de no definir dicha interfaz, el contexto podría pasarse a sí mismo a la estrategia como parámetro.

- Estrategia: Declara una interfaz común para todos los algoritmos soportados. Esta interfaz será usada por el contexto para invocar a la estrategia concreta.
- Estrategia concreta: Implementa el algoritmo utilizando la interfaz definida por la estrategia.

2.3.3. Uso del patrón Strategy



3. Actividades de aseguramiento de calidad del software

Sprint	Actividades por etapa	Responsables
1	Análisis	
	Recopilar, examinar y formular los requisitos del cliente	Daniel Montoya, Andrés Salazar
	Examinar cualquier restricción que se pueda aplicar	Daniel Montoya, Andrés Salazar
	Coordinar el proceso de software	Daniel Montoya, Andrés Salazar
2	Diseño	
	Examinar requisitos generales de la arquitectura de la aplicación	Daniel Montoya, Andrés Salazar
	Definición precisa de cada subconjunto de la aplicación	Daniel Montoya, Andrés Salazar
	Evaluación de conformidad del diseño respecto a los requerimientos	Daniel Montoya, Andrés Salazar
3	Implementación	
	Cumplimiento de estándares (estándares de codificación)	Daniel Montoya, Andrés Salazar
	Evaluación de conformidad de la implementación respecto al diseño	Daniel Montoya, Andrés Salazar
	Documentación del diseño interno del software	Daniel Montoya, Andrés Salazar
4	Pruebas	
	Construcción y revisión de los test unitarios	Daniel Montoya, Andrés Salazar
	Evaluación de conformidad de la implementación respecto a los requerimientos	Daniel Montoya, Andrés Salazar
5	Despliegue	
	Realizar evaluaciones del producto	Daniel Montoya, Andrés Salazar
	Evaluar la aceptabilidad con el usuario	Daniel Montoya, Andrés Salazar
	Evaluar el soporte al producto	Daniel Montoya, Andrés Salazar

4. Referencias Bibliográficas

- Lab.dit.upm.es. (2017). Documentación de código. [online] Available at: http://www.lab.dit.upm.es/~lprg/material/apuntes [Accessed 25 Mar. 2017].
- Google Docs. (2017). Estándares de Codificaciónv1[1].3.doc. [online] Available at: https://docs.google.com/document/d/1rb [Accessed 25 Mar. 2017].
- prezi.com. (2017). Las Métricas y la Calidad de Software. [online] Available at: https://prezi.com/hiups1psmdhh/las-metricas-y-la-calidad-de-software/ [Accessed 25 Mar. 2017].
- Caro, F. and Caro, F. (2017). Sonar: Medida de la calidad de tu código - Paradigma. [online] Paradigma. Available at: https://www.paradigmadigital.com/dev/sonar-medida-de-la-calidad-de-tu-codigo/ [Accessed 25 Mar. 2017].
- Juntadeandalucia.es. (2017). PMD y la calidad estática del código | Marco de Desarrollo de la Junta de Andalucía. [online] Available at: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/374 [Accessed 25 Mar. 2017].
- Juntadeandalucia.es. (2017). CheckStyle | Marco de Desarrollo de la Junta de Andalucía. [online] Available at: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/373 [Accessed 25 Mar. 2017].
- García, D. (2017). Patrones de Comportamiento (IV): Patrón Strategy. [online] Let's code something up!. Available at: https://danielggarcia.wordpress.com/2014/05/12/patrones-de-comportamiento-iv-patron-strategy/ [Accessed 26 Mar. 2017].
- Dropbox. (2017). IEEE 730 2014.pdf. [online] Available at: https://www.dropbox.com/sh/ih439k2p0szpxh0/AADnnhFJWK [Accessed 26 Mar. 2017].