Evaluación de la eficiencia de métodos de identificación del defecto de diseño GodClass

Carlos López[1], Esperanza Manso[2], y Yania Crespo[2]

[1] Universidad de Burgos, EPS Edificio C, C/Francisco Vitoria s/n, Burgos, España,
clopezno@ubu.es,
[2] Universidad de Valladolid, Campus Miguel Delibes, Valladolid, España,
manso@infor.uva.es, yania@infor.uva.es

Resumen La identificación de defectos de diseño en entidades de código es una de las tareas del proceso de mantenimiento del software que sirve para evaluar la calidad de un sistema. Un defecto de diseño describe una situación que sugiere un problema potencial en la estructura del software. La intención de diseño de la entidad, que puede ser expresada como estereotipos de clasificadores estándar de UML, proporciona una fuente de información utilizada en algunas definiciones textuales de defectos. En las entidades de código de un sistema software orientado a objetos la información de estereotipos UML no suele estar disponible explícitamente, aunque los diseñadores y programadores la hayan tenido en cuenta en sus soluciones. En la práctica de la automatización de detección de defectos de diseño, esta información es obviada a pesar de su posible utilidad en el proceso de identificación de defectos. Actualmente existen métodos de identificación del defecto de diseño GodClass basados en métricas de código. Incluso existen herramientas que lo automatizan, como InCode y JDeodorant, ambas avaladas con importantes publicaciones de investigación, en las que esta información no se tiene en cuenta. Nosotros proponemos utilizar técnicas de aprendizaje supervisado basado en clasificadores de tipo árbol de decisión, para modelar el problema de la detección de defectos de diseño como una clasificación de entidades de código "con defecto" o "sin defecto". La clasificación inicial en la fase de entrenamiento se puede obtener a partir de los métodos actuales. Este trabajo presenta un caso de estudio para evaluar cómo influye la información relativa a la naturaleza de diseño de la entidad en la detección de defecto GodClass para distintos clasificadores. La evaluación consiste en comparar de medidas de rendimiento del clasificador obtenidas en la fase de entrenamiento (Recuperación, Precisión y F-Measure). Las resultados avalan la validez de considerar la naturaleza de diseño de la entidad en los métodos de identificación de defectos de código.

Palabras clave: defectos de diseño orientados a objetos, experimentación en ingeniería del software, estereotipo de clasificadores UML, aprendizaje supervisado

1. Introducción

Un *defecto de diseño* describe una situación que sugiere un problema potencial en la estructura del software. Para decidir si el problema es real o relevante la situación tiene que ser examinada con más detalle en su contexto particular.

En la literatura, este concepto tiene una plétora de términos diferentes para referirse a una familia de conceptos similares: **code smells** o **bad smells** en [1], **disharmonies** en [2], **defects** en [3] o **antipatterns** en [4].

En el área de conocimiento relativa a defectos de diseño, uno de los campos de investigación es la definición de heurísticas o reglas de detección basadas en cierta información de las entidades a evaluar. En esta línea, con este trabajo se busca, por una parte modelar el problema de la detección de defectos mediante clasificadores binarios y, por otra parte, mejorar las reglas de detección incorporando información específica de diseño de la entidad de código que se quiere evaluar.

Los estereotipos de clasificadores estándar de UML (exception,interface, entity, control, test, utility), proporcionan una fuente de información sobre la intención de diseño de la entidad, que puede ser útil en la definición de las heurísticas de detección. Entendemos que el estereotipo UML no es la única información de diseño de la entidad. De ahí que, aunque en este trabajo se utilice el conjunto de etiquetas antes mencionado, podrían incorporarse en el futuro otras etiquetas que expresen de alguna forma la intención de diseño de la entidad. A lo largo del trabajo, para expresar este concepto, se utilizan los términos *naturaleza de diseño de la entidad*, o en su forma abreviada hablaremos de *naturaleza de la entidad*.

Como base de experimentación inicial, para confirmar la influencia de la naturaleza de la entidad en el proceso de detección, hemos elegido el defecto de diseño GodClass. En un sistema software orientado a objetos (OO) una GodClass es un objeto que controla a demasiados objetos en el sistema y ha crecido más allá de toda lógica para convertirse en la clase que lo hace todo. Un caso excepcional de esta descripción, es el participante Mediador en una correcta aplicación del patrón de diseño del mismo nombre. En [5] se indica que un contexto de aplicación del patrón Mediador es el problema del control de dependencias de componentes de una interfaz gráfica. En su apartado de consecuencias se menciona que el participante Mediador es un monolito difícil de mantener y reutilizar, dicho de otro modo es una clase de tipo GodClass. Por tanto, conocer el estereotipo de la entidad, interface en este caso, puede ayudar a eliminar falsos positivos en las heurísticas de detección.

En [2, 3], se definen reglas y heurísticas de detección de este defecto basado en métricas de código. Además existen herramientas, que automatizan esas heurísticas para identificar el defecto GodClass, entre otros. Muchas de estas herramientas están avaladas por importantes publicaciones en el ámbito de la investigación, este es el caso de JDeodorant [6] e InCode [7].

En este trabajo, se propone generar nuevas heurísticas basadas en métricas de código aplicando técnicas de aprendizaje supervisado, basadas en clasificadores de tipo árbol de decisión. La detección de defectos se modela como una clasifi-

cación de entidades de código con defecto o sin defecto. La clasificación inicial necesaria en la fase de entrenamiento se obtiene a partir de las herramientas, JDeodorant e InCode, que identifican los defectos sin considerar la naturaleza de la entidad. Nuestro trabajo se basa en la idea de que "el auditor" encargado de identificar defectos en un sistema, es el conocedor del contexto del problema de la entidad y podría desear incluir información de diseño de la entidad (naturaleza de la entidad) para ser considerada en la tarea de identificación de defectos. Esta nueva funcionalidad puede provocar un cambio en las heurísticas de detección para adaptarse a la nueva información de diseño.

El resto del artículo se estructura de la siguiente manera, en la Sec. 2, se describen los objetivos del estudio. En la Sec. 3 se describe la planificación y en la Sec. 4 se describen cuestiones sobre la operación del estudio: sujetos, objetos y recogida de datos. En la Sec. 5 se presenta un análisis de los datos recogidos y en la última sección se muestran las conclusiones y líneas de trabajo futuras.

2. Definición del caso de estudio

Este estudio se ha realizado siguiendo las recomendaciones dadas en [8, 9].

La pregunta de investigación a responder en este trabajo se puede enunciar como: *¿Influye la naturaleza de la entidad, modelada como estereotipo UML, en la predicción de defectos de diseño tipo GodClass?*.

A partir de dicha pregunta el objetivo principal de este estudio enunciado en formato GQM [10] es el siguiente: *Analizar* entidades de código, *con el propósito de* estudiar cómo impacta el conocimiento de la naturaleza de diseño de la entidad en la detección de defectos de diseño basada en métricas de código, *con respecto* a la eficiencia de la detección, en particular del defecto GodClass, *desde el punto de vista* de los investigadores, *en el contexto* académico de la Universidad de Burgos (UBU) y la Universidad de Valladolid (UVa) y de dos aplicaciones de código abierto JFreeChart 1.0.14 y Eclemma 2.1.0.

Como objetivo secundario del estudio se plantea: *Analizar* entidades de código para comparar dos herramientas de predicción (InCode y JDeodorant) del defecto de diseño GodClass, *con respecto* en la similitud de la clasificación, *desde el punto de vista* de los investigadores, *en el contexto* académico de la Universidad de Burgos (UBU) y la Universidad de Valladolid (UVa) y de dos aplicaciones de código abierto JFreeChart 1.0.14 y Eclemma 2.1.0.

3. Planificación

El tipo de experimento que hemos realizado es un estudio de casos, de acuerdo a la clasificación dada por Robson [11]. En esta sección se presenta el diseño del caso de estudio (Sec. 3.1), proporcionando información relativa de los sujetos y objetos del estudio (Sec. 3.2), y cuestiones relacionadas con la instrumentación para llevar a cabo el estudio (Sec. 3.3).

3.1. Diseño y variables de estudio

Las variables independientes del estudio son, las medidas de la entidad de código y la naturaleza de diseño de la entidad que se describen a continuación:

- Medidas de código orientado a objeto, bien conocidas, de Chidamber y Kemerer [12], Lorenz y Kid [13] y otras. Se trabajará con el siguiente conjunto de métricas: Density of Comments (DC), Executable Statements (EXEC), Total Lines of Code (LOC), Depth in Tree (DIT), Number of Children (NOC), Number of Fields (NOF), Response for Class (RFC), Weighted Methods per Class (WMC), Number of Attributes (NOA), Dependency Inversion Principle (DIP), Lack of Cohesion of Methods (LCOM).
- Naturaleza de la entidad, tiene una escala nominal, cuyas categorías se corresponden con los siguientes estereotipos estandar UML: exception (que se identificará como e_1), interface (e_2), entity (e_3), control (e_4), test (e_5), utility (e_6).

En este tipo de problemas se generan conjuntos de datos no balanceados, también conocidos como desequilibrados. Hay más entidades de código sin defecto que con defecto. En este escenario interesa penalizar la detección de falsos negativos. En minería de datos para problemas similares [14] se propone definir una matriz de costes como posible solución al balanceo de datos. Por ello, hemos considerado otro factor que puede afectar al resultado: el coste de equivocarse en una "no predicción".

- Coste falsos negativos, se mide con escala nominal con las categorías "sin coste", "con coste"

La variable dependiente es:

- La eficiencia de la predicción binaria del defecto GodClass, medida mediante Precisión, Recuperación y F-Measure. La definición de estas medidas se explica más adelante.

Se van a simular dos procesos de aprendizaje supervisado (con naturaleza y sin naturaleza) basado en la generación de clasificadores obtenidos mediante el algoritmo J48 [14]. La clasificación inicial de la entidad en "con defecto" y "sin defecto" se calcula como unión de las dos clasificaciones obtenidas por sendos expertos. En nuestro caso los expertos son JDeodorant e InCode, herramientas que identifican automáticamente el defecto de diseño GodClass.

Los conjuntos de datos resultantes son desequilibrados. Es decir, existen muchas más entidades "sin defecto" que "con defecto". En [15] se recomienda aplicar el algoritmo de clasificación J48 vs C45 para conjuntos de datos desequilibrados con las opciones: no poda, no colapsar y corrección de Laplace.

Medidas de la eficiencia de los clasificadores Las medidas para evaluar la eficiencia de los clasificadores generados, "con naturaleza" y "sin naturaleza" de la entidad, han sido la Recuperación, Precisión y F-Measure descritas en [14].

Para evaluar los posibles resultados cuando se predice una clasificación con dos alternativas (si/no) se utiliza una tabla de contingencia o confusión. En ésta, la diagonal principal recoge las predicciones acertadas (Positivos ciertos TP y Negativos ciertos TN). La otra diagonal recoge los dos posibles errores: Falsos positivos (FP): se produce cuando la salida se predice incorrectamente como "si" (o positivo) cuando realmente es "no" (negativo); Falsos negativos (FN): se produce cuando la salida se predice incorrectamente como "no" (negativo) cuando realmente es"si" (positivo).

		Clase predicha	
		si	no
Clase real	si	Positivos ciertos (TP)	Falsos negativos (FN)
	no	Falsos positivos (FP)	Negativos ciertos (TN)

Las medidas de eficiencia propuestas se definen así:
$Precision = \frac{TP}{TP+FP}$ $Recuperacion = RatiodeTP = \frac{TP}{TP+FN}$

A partir de ellas se puede calcular una única medida conocida como F-measure (media harmónica o media de ratios), y se define como:
$F - measure = \frac{2*Recuperacion*Precision}{Recuperacion+Precision} = \frac{2*TP}{2*TP+FP+FN}$

En el problema de identificación de defectos de código, los dos tipos de error, falsos positivos (FP) y falsos negativos (FN), podrían tener diferentes costes, dependiendo de la organización y el contexto de uso. Es decir, en los escenarios de aplicación del proceso de identificación de defectos sobre un sistema, nos pueden aparecer dos situaciones extremas: se sospecha que muchas de las entidades son defectuosas o, por el contrario, se sospecha que la mayoría no lo son. En el primer caso el coste de un FP, es decir, el coste de identificar como defectuosa una entidad que no lo es, puede penalizarse más que el caso de detectar un FN, clasificar como no defectuosa una que sí que lo es. En el segundo caso, el coste de identificar un FP, puede penalizarse menos que el caso de detectar un FN.

En general en el problema que nosotros estudiamos se presenta la segunda situación, se cuenta con una gran cantidad de entidades donde pocas de ellas son defectuosas. Por ello definimos una matriz de costes para tenerla en cuenta en el proceso de aprendizaje cuando se genera el clasificador. Esto provoca un cambio de reglas de detección en función de los costes. Los subconjuntos de instancias de entrenamiento seleccionados como clase real en la validación cruzada no son aleatorios sino que se crean en función de los costes.

A continuación se muestra la matriz de costes utilizada en este caso de estudio. Se ha considerado ponderar los costes de equivocarse al predecir un falso negativo cinco veces más que el resto. Dicho de otra forma, la organización prefiere mayor recuperación (se decrementa FN) a costa de perder precisión (se incrementa FP). Hemos elegido 5, como peso del coste de los FN por ser el valor, entre los explorados, que proporcionaba una eficiencia mejor (F-Measure).

		Clase predicha	
		si	no
Clase real	si	0 (TP)	5 (FN)
	no	1 (FP)	0 (TN)

Hipótesis Las preguntas que responderemos se derivan de los objetivos de este estudio, propuesto en la Sec. 2. En función de ellas se plantean las correspondientes hipótesis, como se muestra en la Tabla 1.

Tabla 1: Preguntas e hipótesis

n	Pregunta n	$H_{0,n}$
1	¿Influye la naturaleza de la entidad, modelada como estereotipo UML, en la predicción de defectos de diseño tipo GodClass?	$H_{0,1a}$ Utilizar la naturaleza de la entidad (i) no mejora la eficiencia del método de identificación (clasificación) del defecto GodClass en el caso de estudio r. Donde, $r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. }, $i \in$ { sin naturaleza, con naturaleza } $H_{0,1b}$ Utilizar la naturaleza de la entidad (i) no mejora la eficiencia del método de identificación (clasificación) del defecto GodClass en el caso de estudio r cuando el coste de falsos negativos es k. Donde, $r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. }, $i \in$ { sin naturaleza, con naturaleza }, $k \in$ { sin coste, con coste FN según la matriz de costes }
2	¿Son similares los métodos respecto a la clasificación binaria del defecto GodClass?	$H_{0,2}$ Los métodos de identificación (j), no se diferencian en cuanto a los resultados de identificación del defecto GodClass en el caso de estudio r. Donde, $r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. }, $j \in$ { Incode 2.07, JDeodorant 4.0.12 }

Diseño del caso de estudio El diseño propuesto es un diseño cruzado en el que se evalúa la eficiencia de los clasificadores al cruzar las variables naturaleza (n) y coste (c):

- Naturaleza: n con naturaleza y $\overline{n}$ sin naturaleza
- Costes: c con costes y $\overline{c}$ sin costes

	Con costes	Sin costes
Con naturaleza	$MedidasRendimiento_{n,c}$	$MedidasRendimiento_{n,\overline{c}}$
Sin naturaleza	$MedidasRendimiento_{\overline{n},c}$	$MedidasRendimiento_{\overline{n},\overline{c}}$

Una interpretación de los datos de la tabla anterior respecto a la hipótesis principal es: si $MedidasRendimiento_{\overline{n},\overline{c}}$ son iguales o mejores que el resto, entonces no es interesante considerar ni el coste, ni la naturaleza en la identificación de defectos.

3.2. Selección de sujetos y objetos

El sujeto que participa en el estudio es un estudiante de doctorado especializado en el área de mantenimiento del software.

Las bibliotecas Java que son los objetos de este estudio son: JFreeChart 1.0.14, Eclemma 2.1.0. Ambas han sido obtenidas a través del repositorio de software de código abierto *SourceForge* (`http://sourceforge.net/`).

JFreeChart es una biblioteca escrita en Java que facilita a los desarrolladores crear diagramas de calidad en sus aplicaciones. Soporta muchos formatos de salida: componentes Swing, ficheros de imagenes (PNG, JPEG) y ficheros con formato vectorial (PDF, EPS y SVG). Está distribuida bajo licencia GNU-LGPL. Desde el punto de vista de tamaño, la aplicación contiene 976 clases y 244.850 líneas de código.

EclEmma es un plugin de Eclipse escrito en Java para calcular la cobertura de pruebas, y está disponible bajo la licencia Eclipse Public License. Acelera el ciclo de desarrollo de test lanzando los test JUnit a la vez que analiza su cobertura. Mejora el análisis de cobertura, resumiendo los resultados de cobertura y colorea las sentencias del código fuente que no han sido probadas. Desde el punto de vista de tamaño, la aplicación contiene 153 clases y 10.561 líneas de código.

En el estudio, las entidades de código a medir son las clases. Es decir cuando hablamos de número de entidades hablamos de número de clases. La selección de objetos se ha basado en el conocimiento de ambas bibliotecas por el sujeto y en la frecuente utilización por la comunidad de desarrolladores en Java.

3.3. Instrumentación

En el estudio hemos utilizado las herramientas siguientes:

- Una herramienta para calcular métricas de las entidades de código, RefactorIt. En el contexto de predicción de defectos de diseño basados en métricas

que se plantea en este trabajo se necesita una herramienta que calcule un amplio conjunto de métricas orientadas a objetos y permita la exportación de resultados para su análisis posterior. Bajo estas premisas, la herramienta seleccionada para obtener las medidas ha sido RefactorIt. En la Sec 3.1 se describió el conjunto de las métricas a emplear en el estudio, todas ellas proporcionadas por la herramienta.

- Una herramienta desarrollada por los propios autores [16] que ayuda a la clasificación de entidades de código según su intención de diseño, expresada como estereotipo estándar de UML (exception, interface, entity, control, test, utility)
- Dos plugins de Eclipse, Incode 2.07 [7] y JDeodorant 4.0.12 [6], que permiten obtener una predicción del defecto GodClass sobre las entidades de código de los objetos seleccionados en este caso de estudio. Se han seleccionado, Incode 2.07 y JDeodorant 4.0.12, por la buena documentación de sus métodos reflejada en sus artículos de investigación publicados.
- Una herramienta de minería de datos, Weka 3.7.5 que permite generar clasificadores con diferentes métodos de caja blanca (JRIP, J48) y calcula las medidas del proceso de aprendizaje supervisado (Recuperación, Precisión, F-Measure). La herramienta se ha elegido, por cumplir estos requisitos y por su documentación [14].
- Una herramienta estadística, R 2.7.1, para realizar el test de hipótesis de McNemar.

4. Operación

En este apartado se presenta cómo se ha llevado a cabo el estudio.

La instrumentación ha sido realizada en un único ordenador personal en el mes de febrero de 2012. La descripción del entorno tecnológico es la siguiente: Intel(R) Core(TM) 2Quad CPU Q8300 2.5 GHz, con RAM 4GB, Sistema Operativo Windows 7 Profesional, Eclipse 3.7.1, Java 1.7.0.

La principal incidencia encontrada ha sido un problema de eficiencia de memoria cuando se trabaja con el plugin de Eclipse JDeodorant, especialmente relevante en el caso de analizar proyectos grandes. Para superar el problema se optó por ampliar la memoria de ejecución de la máquina virtual de Java que lanzaba Eclipse (-Xmx1024m) o realizar la recogida por partes de cada módulo separado de la aplicación.

5. Análisis

Las dos subsecciones siguientes, presentan el análisis para cada una de las hipótesis de la Sec. 3 obtenidas de los objetivos de investigación expuestos en la Sec. 2.

5.1. Estudio de la eficiencia de los clasificadores para identificar GodClass

En este apartado se estudia la hipótesis $H_{0,1}$ analizando de forma descriptiva las medidas de eficiencia de los clasificadores generados en el proceso de aprendizaje, la Recuperación, la Precisión y F-Measure.

$H_{0,1a}$ Utilizar la naturaleza de la entidad (i) no mejora la eficiencia del método de identificación (clasificación) del defecto GodClass en el caso de estudio r.
Donde,
$r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. },
$i \in$ { sin naturaleza, con naturaleza }

La estructura del análisis para la hipótesis $H_{0,1a}$ es la siguiente:

1. Analizar cómo influye en la eficiencia de la identificación del defecto GodClass utilizar o no la naturaleza de la entidad, en el caso de estudio JFreeChart 1.0.14.
2. Analizar cómo influye en la eficiencia de la identificación del defecto GodClass utilizar o no la naturaleza de la entidad, en el caso de estudio Eclemma 2.1.0.
3. Comparar los resultados anteriores.

Todos los resultados obtenidos se pueden consultar en: `http://dl.dropbox.com/u/18996787/JISBD2012/4DCAnalisisEficienciaCompleto.pdf`.

1. En el caso de estudio JFreeChart, las medidas de eficiencia, con respecto a la clase True (entidades con el defecto GodClass), mejoran levemente cuando consideramos la naturaleza de la entidad, mientras que la recuperación se mantiene igual. Los datos que justifican la afirmación son los siguientes:
   Sin naturaleza: Precisión (0,581) Recuperación (0,537) F-Measure (0,558)
   Con naturaleza: Precisión (0,600) Recuperación (0,537) F-Measure (0,567)
2. En el caso de estudio Eclemma, las medidas de eficiencia, con respecto a la clase True (entidades con el defecto GodClass), mejoran significativamente cuando consideramos la naturaleza de la entidad.
   Sin naturaleza: Precisión (0,200) Recuperación (0,182) F-Measure (0,190)
   Con naturaleza: Precisión (0,455) Recuperación (0,455) F-Measure (0,455)
3. En conclusión, utilizar la naturaleza de la entidad ha mejorado las medidas de eficiencia de detección de defectos GodClass.

$H_{0,1b}$ Utilizar la naturaleza de la entidad (i) no mejora la eficiencia del método de identificación (clasificación) del defecto GodClass en el caso de estudio r cuando el coste de falsos negativos es k.
Donde,
$r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. },
$i \in$ { sin naturaleza, con naturaleza },
$k \in$ { sin coste, con coste FN según la matriz de costes }

La estructura del análisis para la hipótesis $H_{0,1b}$ es la siguiente:

1. Analizar cómo influye en la eficiencia de la identificación del defecto GodClass utilizar o no la matriz de costes, al utilizar o no la naturaleza de la entidad, en el caso de estudio JFreeChart 1.0.14.
2. Analizar cómo influye en la eficiencia de la identificación del defecto GodClass utilizar o no la matriz de costes, al utilizar o no la naturaleza de la entidad, en el caso de estudio Eclemma 2.1.0.
3. Comparar los resultados anteriores.

1. En el caso de estudio JFreeChart, las medidas de eficiencia cuando utilizamos la matriz de costes, con respecto a la clase True (entidades con el defecto GodClass), mejoran levemente al considerar la naturaleza de la entidad.
   Sin naturaleza: Precisión (0,570) Recuperación (0,970) F-Measure (0,718)
   Con naturaleza: Precisión (0,575) Recuperación (0,970) F-Measure (0,722)
   Las medidas de eficiencia cuando utilizamos la naturaleza de la entidad, con respecto a la clase True (entidades con el defecto GodClass), mejoran al considerar la matriz de costes para la recuperación y F-Measure, cuyo aumento compensa la disminución de la precisión.
   Sin matriz de costes: Precisión (0,600) Recuperación (0,537) F-Measure (0,567)
   Con matriz de costes: Precisión (0,575) Recuperación (0,970) F-Measure (0,722)
   Las medidas de eficiencia cuando no utilizamos la naturaleza de la entidad, con respecto a la clase True (entidades con el defecto GodClass), mejoran al considerar la matriz de costes para la recuperación y F-Measure cuyo aumento compensa la disminución de la precisión.
   Sin matriz de costes: Precisión (0,581) Recuperación (0,537) F-Measure (0,558)
   Con matriz de costes: Precisión (0,570) Recuperación (0,970) F-Measure (0,718)
2. En el caso de estudio Eclemma, las medidas de eficiencia cuando utilizamos la matriz de costes, con respecto a la clase True (entidades con el defecto GodClass), mejoran significativamente cuando consideramos la naturaleza de la entidad.
   Sin naturaleza: Precisión (0,250) Recuperación (0,364) F-Measure (0,296)
   Con naturaleza: Precisión (0,385) Recuperación (0,455) F-Measure (0,417)
   Las medidas de eficiencia cuando utilizamos la naturaleza de la entidad, con respecto a la clase True (entidades con el defecto GodClass), no mejoran, la F-Measure empeora ligeramente.
   Sin matriz de costes: Precisión (0,455) Recuperación (0,455) F-Measure (0,455)
   Con matriz de costes: Precisión (0,385) Recuperación (0,455) F-Measure (0,417)
   Las medidas de eficiencia cuando no utilizamos la naturaleza de la entidad, con respecto a la clase True (entidades con el defecto GodClass), mejoran al considerar la matriz de costes.
   Sin matriz de costes: Precisión (0,200) Recuperación (0,182) F-Measure (0,190)
   Con matriz de costes: Precisión (0,250) Recuperación (0,364) F-Measure (0,296)
3. Teniendo en cuenta que la cantidad de entidades con defectos es muy pequeña, se pone claramente de relieve que la matriz de costes es necesaria: es significativo que en presencia de la matriz de costes, utilizar también la naturaleza de la entidad ha mejorado las medidas de la eficiencia para detectar el defecto GodClass en ambos casos de estudio. La conclusión a la vista de todos los

resultados (Tabla 2), es que tanto la matriz de costes como la naturaleza de la entidad son dos factores relevantes para mejorar la detección de dicho defecto.

La Tabla 2 resume los resultados del análisis: Con o Sin Naturaleza (n, $\overline{n}$), Con o Sin Costes (c, $\overline{c}$).

	Precisión	Recuperación	F-Measure
$\overline{c}$: n vs. $\overline{n}$	Mejora en ambos casos	Mejora en ambos casos	Mejora en ambos casos
$\overline{n}$: c vs $\overline{c}$	Mejoran en ambos casos	Mejora en ambos casos	Mejora en ambos casos
n: c vs $\overline{c}$	JFreeChart mejora	JFreeChart mejora	JFreeChart mejora
c: n vs. $\overline{n}$	Mejora en ambos casos	Mejora en ambos casos	Mejora en ambos casos

Tabla 2: Resumen del análisis respecto $H_{0,1a}$ y $H_{0,1b}$

5.2. Comparación de los métodos de clasificación

En este apartado se estudia la hipótesis $H_{0,2}$ utilizando el test estadístico de McNemar.

La prueba no paramétrica de McNemar sirve para comparar las puntuaciones de dos jueces en si/no sobre k objetos. [17]. Donde,

- Los objetos a clasificar son las entidades de código de la aplicación
- La variable dependiente es si la entidad tiene o no tiene el defecto GodClass

$H_{0,2}$ Los métodos de identificación (j), no se diferencian en cuanto a los resultados de identificación de GodClass en el caso de estudio r.
Donde,
$r \in$ { JFreeChart 1.0.14, Eclemma 2.1.0. },
$j \in$ { Incode 2.07, JDeodorant 4.0.12 }

La Tabla 3 contiene la información sobre la identificación de defectos de los casos de estudio con los dos métodos utilizados.

En la siguiente tabla se muestra el resumen estadístico utilizado para contrastar la hipótesis nula $H_{0,2}$. A partir del test de hipótesis de McNemar se rechaza la hipótesis nula ($\alpha <= 0{,}05$) en las dos aplicaciones consideradas. Así que podemos concluir que los métodos de Incode y JDeodorant no son similares. Estos resultados justifican la necesidad de adaptar el método de identificación por medio de aprendizaje supervisado.

	Nº total de entidades	Nºentidades con defectos identificadas InCode	Nºentidades con defectos identificadas JDeodorant	Nºentidades con defectos identificadas JDeodorant e InCode
JFreechart	975	67	0	67
eclemma	152	0	6	6

Tabla 3: Predicciones del defecto GodClass con JDeodorant e Incode

	JFreeChart	Eclemma
p-valor	7.433e-16	0.04123
McNemar	65.0149	4.1667

5.3. Amenazas a la validez

La validez de construcción del caso de estudio, entendida como las variables que han sido correctamente medidas, está amenazada por la variable naturaleza de la entidad, pues la clasificación de entidades en estereotipos UML no es ortogonal cuando se aplica sobre entidades de código reales.

La validez interna del caso de estudio, entendida como la causalidad de nuestras conclusiones, está afectada porque la clasificación de entidades en estereotipos UML es subjetiva.

Desde la perspectiva de la validez externa, referida a cuántos de nuestros resultados se pueden generalizar en otras circunstancias, se han identificado las siguientes amenazas:

- Los proyectos utilizados y el lenguaje de programación de los mismos limita la extensión de estos resultados.
- La selección de los sujetos se ha hecho por conveniencia.
- Las medidas de eficiencia de los clasificadores parecen depender del algoritmo utilizado y su configuración para clasificar, en nuestro caso J48 -O -U -A.
- La variación en la matriz de costes puede modificar los resultados, en nuestro caso se ha elegido un peso de 5.

6. Conclusiones

El objetivo principal de este estudio es comprobar si la naturaleza de la entidad de código, entendida como estereotipos estandar de UML, mejora la eficiencia de los métodos de identificación de GodClass, basados en métricas de código. Este objetivo es de interés porque las técnicas de identificación de defectos en entidades de código son útiles para mejorar el mantenimiento del mismo.

Durante el estudio se ha observado que en el problema de identificación de defectos sobre entidades de código, se generan conjuntos de datos no balanceados.

Para resolver este problema, hemos considerado el coste del error de los falsos negativos en los métodos de detección.

Podemos concluir lo siguiente:

1. En el estudio de la eficiencia de los clasificadores para identificar el defecto GodClass, los resultados observados indican que la naturaleza de diseño de la entidad mejora su eficiencia. Además, la matriz de costes, cuando se penaliza el coste de los falsos negativos, también mejora dichas medidas. Por ello, la matriz de costes y la naturaleza de diseño de la entidad son dos factores relevantes para mejorar dicha eficiencia.
2. En el estudio de comparación de los dos métodos de identificación del defecto GodClass, los resultados indican que no existe concordancia, por tanto la clasificación tiene un grado de subjetividad. Es por ello por lo que se aboga desde este trabajo porque los métodos de identicación de GodClass sean adaptados al contexto de la aplicación con técnicas de aprendizaje supervisado.

Como líneas de trabajo futuro, para completar los resultados de este estudio estamos trabajando en:

- Añadir la supervisión humana para validar los resultados de los métodos eliminando los falsos positivos en función de la excepciones documentadas en [1].
- Repetir el caso de estudio sobre otros defectos de diseño como DataClass y FeatureEnvy, por ejemplo.
- Estudiar posibles refinamientos de la clasificación de entidades de código en esterotipos UML.

Referencias

1. Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Object Technology Series. Addison-Wesley, June 1999.
2. Michele Lanza and Radu Marinescu. *Object-Oriented Metrics in Practice - Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Springer, 2006.
3. Naouel Moha, Yann-Gaël Guéhéneuc, Anne-Françoise Le Meur, Laurence Duchien, and Alban Tiberghien. From a domain analysis to the specification and detection of code and design smells. *Formal Aspects of Computing*, May 2009.
4. William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley and Sons, March 1998.
5. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, January 1995.
6. Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. Jdeodorant: identification and application of extract class refactorings. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1037–1039, New York, NY, USA, 2011. ACM.

7. Radu Marinescu and George Ganea. incode.rules: An agile approach for defining and checking architectural constraints. In *Proceedings of the Proceedings of the 2010 IEEE 6th International Conference on Intelligent Computer Communication and Processing*, ICCP '10, pages 305–312, Washington, DC, USA, 2010. IEEE Computer Society.
8. C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*, volume 6 of *International Series in Software Engineering*. Springer, 2000.
9. Per Runeson and Martin Host. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14:131–164, 2009.
10. Victor Basili, Gianluigi Caldiera, and Dieter H. Rombach. The goal question metric approach. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley, 1994.
11. C. Robson. *Real World Research - A Resource for Social Scientists and Practitioner-Researchers*. Blackwell Publishing, Malden, second edition, 2002.
12. Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object oriented design. *ACM SIGPLAN Notices*, 26(11):197–211, 1991.
13. Mark Lorenz and Jeff Kidd. *Object-Oriented software metrics: a practical guide*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.
14. Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, January 2011.
15. David Cieslak, T. Hoens, Nitesh Chawla, and W. Kegelmeyer. Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery*, 24(1):136–158, 2012. 10.1007/s10618-011-0222-1.
16. Carlos López, Esperanza Manso, and Yania Crespo. The identification of anomalous code measures with conditioned interval metrics. In *13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010) Málaga, Spain*, Málaga, July 2010.
17. S. Siegel and N.J. Castellan. *Non-parametric statistics for the behavioural sciences (2nd ed)*. Mc Graw Hill, 1988.