



Evaluación de herramientas de detección y corrección de defectos de diseño

Alberto Mendo Alonso

Javier Sobrino Fernández

Francisco Javier Pérez García
jperez@infor.uva.es

Yania Crespo González-Carvajal
yania@infor.uva.es

Informe Técnico GIRO - 2011/02

ÍNDICE DE CONTENIDOS

CAPÍTULO 1 – INTRODUCCIÓN

1.1	Objetivos	7
1.2	Introducción a los <i>Design Smells</i>	7
1.3	Tipos de <i>Design Smells</i>	8
1.3.1	Antipatterns (Antipatrones)	8
1.3.2	Bad Smells	10
1.3.3	Disharmonies	11
1.4	Planificación del proyecto	12
1.4.1	Estimaciones temporales y tareas	12
1.4.1.1	Lista de tareas	12
1.4.1.2	Distribución temporal	14
1.4.1.3	Seguimiento del proyecto	16

CAPÍTULO 2 – TAXONOMÍA

2.1	Introducción	23
2.2	Taxonomía	23
2.2.1	Entorno de Trabajo	24
2.2.2	Target Artefact	24
a.	Tipo de artefacto	25
b.	Versiones	26
c.	Tipo de representación	26
2.2.3	Design Smells.....	27
a.	Tipo de <i>smells</i>	28
b.	Granularidad	28
c.	Relación entre entidades	28
d.	Herencia	28
2.2.4	Actividad	29
a.	Tipo de actividad	29
b.	Técnica	31
c.	Automatización	32
d.	Resultado	32

CAPÍTULO 3 – ANÁLISIS DE LAS HERRAMIENTAS

3.1	Búsqueda de herramientas	37
3.2	Análisis de herramientas	39
3.2.1	Analyst4j	42
3.2.2	ArgoUML.....	48
3.2.3	Argus Codewatch	52
3.2.4	CodePro Analytix	58
3.2.5	Cultivate	62
3.2.6	Eclipse	70
3.2.7	FxCop	76
3.2.8	inCode	82

3.2.9	jDeodorant	88
3.2.10	jRefactory	94
3.2.11	M2 Resource Standard Metrics	100
3.2.12	PMD	106
3.2.13	Reek	112
3.2.14	RevJava	118
3.2.15	Roodi	124
3.2.16	STAN	128
3.2.17	Structural Analysis for Java	134
3.2.18	Structure101	140
3.2.19	StyleCop	148
3.2.20	Together	154
3.2.21	TRex	162
3.2.22	XRefactory	168
3.3	Otras Herramientas	172
3.3.1	CodeClinic	172
3.3.2	Hammurapi	174
3.3.3	jCosmo	175
3.3.4	Sissy	176
3.3.5	Croccopat	178
3.3.6	XRay	178

CAPÍTULO 4 – TABLAS COMPLETAS Y CONCLUSIONES

4.1	Target Artefact	183
4.2	Design Smells	185
4.3	Activity.....	187
4.4	Conclusiones	190

CAPÍTULO 5 – PÁGINA WEB

5.1	Introducción	195
5.1.1	Objetivos.....	195
5.1.2	Lista de Tareas	195
5.2	Especificación de requisitos	195
5.2.1	Captura de requisitos	196
5.2.2	Definición de requisitos	197
5.2.2.1	Introducción	197
5.2.2.2	Requisitos de datos	198
5.2.2.3	Requisitos de interfaz	199
5.2.2.4	Requisitos navegacionales	200
5.2.2.5	Requisitos de personalización	201
5.2.2.6	Requisitos no funcionales	201
5.2.2.7	Casos de uso	202
5.2.3	Validación de requisitos	206
5.3	Introducción a los CMS	206
5.4	Joomla!	207
5.5	Diseño.....	212
5.6	Mantenimiento Web	215

REFERENCIAS	223
APÉNDICE I	231
ÍNDICE DE FIGURAS	235
ÍNDICE DE TABLAS	239

Capítulo 1

INTRODUCCIÓN

1.1. Objetivos

A medida que un sistema evoluciona y sufre modificaciones, su estructura interna, su diseño, se degrada, empeorando a su vez la mantenibilidad y la reusabilidad del mismo, estos problemas se conocen como defectos de diseño (*Design smells*).

Este proyecto consiste en realizar un estudio y evaluación de una serie de herramientas de detección y corrección de defectos de diseño, para lo cual se parte de un estudio previo, ya existente, en el que se detallan cuáles son las características deseables y comunes para una herramienta de este tipo y, por lo tanto, las características evaluables, y en el que ya se ha realizado una evaluación de un pequeño número de herramientas [1]. Tras la evaluación de las herramientas se elabora un sitio web en el que se presentan los resultados, permitiendo a sus autores la modificación de las tablas, bien sea cambiando los resultados expuestos o bien completándolos.

1.2. Introducción a los Design Smells

Los sistemas software deben evolucionar continuamente para hacer frente a requisitos y entornos en constante cambio. Sin embargo, al contrario que los patrones de diseño, los “Design Smells” (“*soluciones pobres de diseño para problemas software generalizados y recurrentes*” [1]) pueden impedir su evolución, dificultando a los desarrolladores de software llevar a cabo cambios en el código. Estos *Design smells* incluyen problemas de bajo nivel o locales, tales como “Code smells” que suelen ser síntomas de defectos de diseño a un nivel más global, como los antipatrones (Los cuales aparecerán definidos posteriormente en este trabajo). Dichos *Code smells* son indicadores o síntomas de la posible presencia de *Design smells*. Fowler presenta veintidós *Code Smells* en su libro [6], todos ellos estructuras en el código fuente que sugieren la posibilidad de refactorizaciones. *Duplicated code*, *long methods*, *large classes*, y *long parameter lists*, son sólo algunos síntomas de *Design smells* y de oportunidades de refactorización de entre los veintidós presentados.

“Un ejemplo de Design Smell es el antipatrón Spaghetti Code, que es característico del pensamiento procedimental de la programación orientada a objetos. Spaghetti Code se revela por contener clases sin estructura que declaran largos métodos sin parámetros. Spaghetti Code no explota los mecanismos de orientación a objetos como el polimorfismo y la herencia y evita su uso” [32].

“Se utiliza el término “smell” para designar tanto los errores en código (Code smells) como los Design smells. Este uso no excluye que, en un contexto particular, un smell pueda ser la mejor forma de diseñar o implementar de manera efectiva un sistema” [32]. Por ejemplo, los análisis generados automáticamente por los generadores de análisis son a menudo *Spaghetti Code*, es decir, clases muy grandes con métodos muy largos. Sin embargo, a pesar de tales clases *smell*, los ingenieros de software deben evaluar manualmente los posibles efectos negativos en función del contexto.

La detección de *smells* puede reducir sustancialmente el coste de las actividades posteriores de desarrollo y las fases de mantenimiento. Sin embargo, la detección en los sistemas grandes requiere mucho tiempo y consume muchos recursos y es una actividad propensa a errores, porque los *smells* abarcan clases y métodos, y sus descripciones dejan mucho espacio para la interpretación. Han sido propuestos varios métodos para especificar y detectar *smells*, pero presentan tres limitaciones. En primer lugar, los autores casi nunca explican el análisis que se lleva a cabo para obtener las especificaciones de los *smells* y el marco de detección subyacente. En

segundo lugar, la traducción de las especificaciones en algoritmos de detección es a menudo una caja negra, que impide la replicación. Por último, los autores no presentan los resultados de su detección en un conjunto representativo de *smells* ni, normalmente, sistemas para permitir la comparación entre enfoques.

1.3. Design Smells

Algunos de los principales tipos de defectos de diseño que detectan las herramientas que se han analizado en este trabajo son:

Bad Smells: “*Síntomas de errores de diseño que indican la necesidad de refactorización*”. [5]

Antipatterns: “*Un antipatrón es una forma literaria que describe una solución recurrente que genera consecuencias negativas*”. [2]

Disharmonies: “*Defectos de diseño que afectan a entidades individuales tales como clases y métodos*”. [7]

Bad Coding Style: Son violaciones en las convenciones de estilo o los estándares de código ya establecidos, como los del catálogo de Wake [89], bien en forma de reglas basadas en expresiones regulares o bien mediante mediciones métricas que se deben encontrar en ciertos intervalos de valores.

Además hay otros *Design Smells* a tener en cuenta como Bad Pattern Usage (Referente a la mala aplicación de un patrón en el código a analizar como puede suceder, por ejemplo, con el patrón Singleton) o Metrics Warnings (Donde la herramienta detecta defectos a través del cálculo de métricas).

Los tres primeros tipos de defectos de diseño se basan en catálogos en constante ampliación ya que hay cientos de defectos de diseño diferentes y que en ocasiones todavía no tienen un referente en los documentos ya existentes. Esto principalmente se da en el catálogo de *Antipatterns*, y en menor medida en el de *Bad Smells*. Para la realización de este proyecto se ha partido de los catálogos que aparecen en [2], [3] y [4] para los *Antipatterns* (ver *tablas 1 y 2*), [5] y [6] para los *Bad Smells* (ver *tablas 3 y 4*), y [8] para *Disharmonies* (ver *tablas 5 y 6*). Debido a su proceso de constante ampliación y a su gran variedad, a continuación se detallan los catálogos utilizados para estos tres defectos de diseño.

1.3.1. Antipatterns

Los antipatrones (*Antipatterns*) son descripciones o soluciones de situaciones recurrentes que producen consecuencias negativas. Un antipatrón puede ser el resultado de una decisión equivocada sobre cómo resolver un determinado problema, o bien, la aplicación correcta de un patrón de diseño en el contexto equivocado.

Según Brown *et al.* [2] “*Un antipatrón es una forma literaria que describe una solución recurrente que genera consecuencias negativas*”. Analizando parte por parte esta definición se entiende:

- Forma literaria: descripciones de problemas, no de código.

- Recurrente: si no es un patrón, entonces no es un antipatrón. Se deben establecer diversas ocurrencias del mismo comportamiento erróneo preferentemente en diversos contextos.
- Consecuencias negativas: el diseño debe producir un impacto negativo.

La esencia de un antipatrón es la aportación de dos soluciones en lugar de un problema y una solución como sucede en los patrones de diseño. Reúnen lo que ha producido efectos negativos y proveen dos soluciones: la problemática (aquella con consecuencias negativas), y la refactorizada (aquella que transforma la situación negativa en una teóricamente más saludable).

Relación con patrones de diseño y refactorizaciones.

Al igual que los patrones de diseño, los antipatrones proveen un vocabulario común con el fin de mejorar la comunicación en el equipo de desarrollo, permitiendo poder identificar problemas y discutir soluciones. Pero se diferencian en que mientras que los patrones de diseño documentan soluciones exitosas, los antipatrones documentan soluciones problemáticas: qué salió mal y por qué. Algunos desarrolladores sostienen que los antipatrones son el lado oscuro de los patrones de diseño, ya que muchas de las veces que se implementan soluciones basadas en patrones no se evalúa cómo de aplicables son para el problema que se está intentando resolver. A su vez, muchos otros desarrolladores conocedores de patrones de diseño tienden a clasificar todo como “capaz de ser resuelto con patrones de diseño”.

Antipatterns		
Abstraction Inversion	Corn Cob	Fool Trap
Accidental Complexity	Cover your Assets	Functional Decomposition
Accidental Inclusion	Creeping Featuritis	Game Over You Lose?
Acme Pattern	Copy and Paste Programming	Geographically Distributed Development
Alcohol Fueled Development	Dead End	Give Me Estimates Now
Ambiguous View point	Death by Planning	Glass Wall
Analogy Breakdown Antipattern	Decision by Arithmetic	Golden Hammer
Analysis Paralysis	Design by Committee	Ground Hog Day Project
An Athena	Design For The Sake Of Design	Heir Apparent
Anchored Helper?	Discordant Reward Mechanisms	Hero Culture
Architecture by Implication	Doer and Knower	Hidden Requirements
Asynchronous Unit Testing	Dry Waterhole	Idiot Proof Process.
Autogenerated Stove pipe Antipattern	Egalitarian Compensation	IfItIs Working don't Change
Bear Trap	Emails Dangerous	Import Beast?
Big Ball of Mud	Emperors New Clothes	Implementation Inheritance
Blame Storming	Empire Building	Input Kludge
Blow hard Jamboree	Exception Funnel	Intellectual Violence
Boat Anchor	False Economy	Internationalization Fully Supported?
Cargo Cult	False Surrogate Endpoint	Irrational Management?
Cascading Dialog Boxes Antipattern	Fear of Success	Its an Operator Problem
Configuration Abomination?	Fire Drill	Job Keeper
Continuous Obsolescence	Floating Point Currency	Jumble Antipattern
Control Freak	Floating Point Fractions	Junkyard Coding

Tabla 1. Catálogo de Antipatterns. De Abstraction Inversion a Junkyard Coding¹.

¹ <http://c2.com/cgi/wiki?AntiPatternsCatalog>.

Antipatterns		
Kill Two Birds With One Stone	Reinvent The Wheel	The Customers are Idiots
Kitchen Sink Design	Roll Your Own Database Requirements Tossed over the Wall	The Feud?
Lava Flow	Rube Goldberg Machine	The Grand old Duke Of York
Leading Request	Scape Goat	They Understood Me
Magic Container Manager	Secret Society	Thrown over the Wall
Controls Process	Shoot The Messenger	Tower of Voodoo
Mushroom Management	Single Function Exit Point	Train the Trainer
Naked Documents?	Smoke and Mirrors	Trusting Souls?
Nationalism	Software Merger	Untested but Finished
Net Negative Producing Programmer	Spaghetti Code	Vendor LockIn
Not Invented Here	Specify Nothing	Viewgraph Engineering
Null Flag	Standing on the Shoulders of Midgets	Walking Through a Mine Field
Over Generalization of Business Logic	Stovepipe Enterprise?	Warm Bodies
Over Use of Patterns	Stovepipe AntiPattern	We are Idiots
Path of Least Resistance	String without Length	Wolf Ticket
Passing Nulls to Constructors	Sumo Marriage	What and How?
Plug Compatible Interchangeable Engineers	SweepIt under the Rug Antipattern	Yet Another Meeting will SolveIt
Poltergeists	Swiss Army Knife	Yet Another Programmer
Project Mismanagement?	That's not Really An Issue	Zero Means Null
Ransom Note Antipattern?	The Blob	

Tabla 2. Catálogo de Antipatterns. De Kill Two Birds With One Stone a Zero Means Nulls¹.

1.3.2. Bad Smells

Los *Bad Smells* se definen como síntomas de errores de diseño que indican la necesidad de refactorización. La detección de *bad smells* puede llevarse a cabo, por ejemplo, mediante la comprobación de métricas, que pueden dar alguna pista al programador para aplicar la refactorización conveniente.

En [5] se puede encontrar una lista de *Bad Smells* (ver tablas 3 y 4), entre los que se incluyen algunos como *Duplicated Code*, *Long Method*, *Large Class*, *Feature Envy*, *Comments*, etc.

Bad Smell	Grupo	Granularidad
Data Clumbs	Bloater	Class
Large Class	Bloater	Class
Long Method	Bloaters	Method
Long Parameter List	Bloaters	Method
Primitive Obsession	Bloaters	Method
Alternative Classes with Different Interfaces	O-O Abusers	Method
Refused Bequest	O-O Abusers	Class
Switch Statements	O-O Abusers	Method
Temporary Field	O-O Abusers	Class

Tabla 3. Catálogo de Bad Smells utilizado. Del grupo Bloater al grupo O-O Abusers.

¹ <http://c2.com/cgi/wiki?AntiPatternsCatalog>.

Bad Smell	Grupo	Granularidad
Divergent Change	Change Prevents	Class
Parallel Inheritance Hierarchies	Change Prevents	Class
Shotgun Surgery	Change Prevents	Class
Data Class	Dispensables	Class
Duplicate Code	Dispensables	Method
Lazy Class	Dispensables	Class
Speculative Generality	Dispensables	Class
Feature Envy	Couplers	Method
Innapropriate Intimacy	Couplers	Class
Message Chains	Couplers	Class
Middle Man	Couplers	Class
Comments	Not defined	Class
Incomplete Library	Not Defined	Class

Tabla 4. Catálogo de Bad Smells utilizado. Del grupo Change Prevents a Not Defined.

1.3.3. Disharmonies

Las *disharmonies* son defectos de diseño que afectan a entidades individuales tales como clases y métodos. La particularidad de estas es que su efecto negativo sobre el diseño se puede considerar como un elemento aislado, es decir, son problemas que se encuentran en determinadas partes del código y únicamente afectan a estas partes, no viéndose alteradas de manera indirecta otras zonas del código.

Se pueden identificar tres aspectos distintos que contribuyen a la identidad de las *disharmonies*: su tamaño, su interfaz y su aplicación. Para cada uno de ellos se define una regla:

- Las operaciones y las clases deben tener un tamaño armonioso, es decir, se debe evitar el excesivo tamaño en las extremidades.
- Cada clase debe presentar su identidad, es decir, su interfaz, por un conjunto de los servicios que tienen una responsabilidad única y que proporcionan un comportamiento único.
- Los datos y las operaciones deben colaborar armónicamente en la clase a la que pertenecen semánticamente.

DISHARMONIES
Identity Disharmonies
<ul style="list-style-type: none"> • Rules of Identity Hamony • Overview of identity Dishamonies • God Class • Feature Envy • Data Class • Brain Method • Brain Class • Significant Duplication • Recovering from Identity Dishamonies

Tabla 5. Catálogo de Dishamonies utilizado. Identity Dishamonies.

Collaboration Disharmonies
<ul style="list-style-type: none">• Collaboration Harmony Rule• Overview of Collaboration Disharmonies• Intensive Coupling• Dispersed Coupling• Shotgun Surgery• Recovering from Collaboration Disharmonies

Tabla 6. Catálogo de Disharmonies utilizado. Collaboration Disharmonies.

Para el análisis y evaluación de las herramientas de detección y corrección de los defectos de diseño anteriormente descritos se parte de las tablas y esquemas propuestos por Pérez *et al.* [1] pero no de una manera única, es decir, puede surgir la necesidad de ampliar algunos apartados dado que dicho estudio está realizado para un número limitado de herramientas y pueden aparecer nuevas características en los análisis efectuados en este trabajo (En el capítulo correspondiente a la fase de análisis de herramientas se detallarán los esquemas utilizados y sus principales puntos).

Como ya se ha comentado, tras el análisis y evaluación de las herramientas se realiza una página web por lo que la creación de esta memoria se divide en dos grandes partes, una relacionada con el análisis de las herramientas y que se completa a la vez que se realizan los análisis y otra con todo lo referente a la realización del sitio web.

1.4. Planificación del proyecto

A continuación se presenta una planificación inicial del proyecto en cuanto a las tareas que se deben realizar y el tiempo estimado para la realización de cada una de ellas (Posteriormente se realiza un seguimiento de dichas tareas en el que se detallan los posibles problemas surgidos al intentar seguir dicha planificación). Durante los primeros meses desde la concesión del proyecto apenas se espera avanzar ya que uno de los alumnos tiene varios exámenes durante el mes de enero, por lo que las horas disponibles serán mínimas debido a las horas de clase y las horas de estudio necesarias. A partir de ahí se espera avanzar a gran ritmo para intentar finalizar el proyecto en el mes de Septiembre, antes de la finalización del curso académico 2009/2010.

1.4.1. Estimaciones temporales y tareas

1.4.1.1. Lista de tareas

Planificación: Estudio de la distribución temporal de las diferentes etapas del proyecto

Búsqueda de herramientas: Realización de la búsqueda de herramientas que aparecen en el listado inicial facilitado (ver *tablas 7 y 8*).

Análisis de herramientas y creación de la memoria: Se completan las tablas que aparecen en el artículo de Pérez *et al.* [1], y que son la base del proyecto, con las herramientas obtenidas en la tarea anterior. En esta fase se podrán descartar o añadir algunas herramientas por no estar directamente relacionadas con los objetivos del proyecto.

Análisis de requisitos de la página web: Establecer los requisitos iniciales para la creación de la página web (Lenguaje de desarrollo, características, diseño...).

Evaluación de herramientas de detección y corrección de defectos de diseño

Creación de la página web: Exposición de los análisis realizados en una web que permita la lectura de los mismos de forma simple y que facilite el entendimiento de estos.

Finalización de la memoria: Se completa la memoria con todo lo que queda pendiente al llegar a este punto, como son las conclusiones finales y los detalles de todo lo referente al sitio web. Se realizan tareas de reorganización, limpieza, maquetación y todo lo necesario para dar por finalizada la memoria.

Name	Contribution	Contributor	Downloadable	Free (as beer)	Free (as speech)
ArgoUML	Tool	Community	Yes	Yes	Yes
Chatzigeorgiou	Tool and Approach	Academia	No	Yes	Yes
CheckStyle	Tool	Community	Yes	Yes	Yes
CodeClinic	Tool and Approach	Academia	No	¿?	¿?
CodeCrawler	Tool and Approach	Academia	Yes	Yes	Yes
CodeNose	Tool and Approach	Academia	No	¿?	¿?
CodePro	Tool	Industry	Yes	No	No
Croccopat	Tool and Approach	Academia	Yes	Yes	Yes
Cultivate	Tool	Academia	No	¿?	¿?
DECOR	Tool and Approach	Academia	Yes	Yes	No
Dependency Finder	Tool	Community	Yes	Yes	Yes
Eclipse	Tool	Industry and Community	Yes	Yes	Yes
Eclipse Metrics Plugin	Tool	Industry and Community	Yes	Yes	Yes
FxCop	Tool	Industry	Yes	Yes	No
Hammurapi	Tool	Industry	Yes	Yes	No
iPlasma	Tool and Approach	Academia	Yes	Yes	No
Java PathFinder	Tool and Approach	Academia and Industry	Yes	Yes	Yes
jCosmo	Tool and Approach	Academia	Yes	Yes	No
jDeodorant	Tool and Approach	Academia	Yes	Yes	No
jDepend	Tool	Community	Yes	Yes	Yes
Lint4j	Tool	Industry	Yes	Yes	No
GIRO	Tool and Approach	Academia	No	---	---
Munro	Approach	Academia	No	---	---
OptimalAdvisor	Tool	Industry	Yes	No	No
PMD	Tool	Community	Yes	Yes	Yes
PRODEOOS	Tool and Approach	Academia	No	---	---
Reek	Tool	Community	Yes	Yes	Yes
RefactorIt	Tool	Industry	Yes	Yes	Yes
RevJava	Tool	Academia	Yes	Yes	No
Roodi	Tool	Community	Yes	Yes	Yes
SA4J	Tool	Industry	Yes	Yes	Yes
Sahraoui	Tool and Approach	Academia	No	---	---
Alikacem	Tool and Approach	Academia	No	---	---
SemmlCode	Tool	Industry	Yes	Yes	Yes
Shimba	Tool and Approach	Academia	No	---	---
Sissy	Tool and Approach	Academia	Yes	Yes	Yes
Smalllint	Tool / Approach	Academia	Yes	Yes	Yes / No
M2 Resource Standard Metrics	Tool	Industry	Yes	No	No
Sotograph	Tool	Industry	Yes	No	No
Stan	Tool	Industry	Yes	No	No
Structure101	Tool	Industry	Yes	No	No

Tabla 7. Listado Inicial. De ArgoUML a Structure101.

Name	Contribution	Contributor	Downloadable	Free (as beer)	Free (as speech)
Tahvildari and Kontogiannis	Tool and Approach	Academia	No	---	---
Travassos	Approach	Academia	No	---	---
Trifu	Tool and Approach	Academia	No	---	---
Incode	Tool and Approach	Industry	Yes	No	No
Together	Tool	Industry	Yes	No	No
Argus Code Watch	---	Community	---	---	---
Analyst4J	---	Industry	---	---	---
GOOSE	---	---	---	---	---
JRefractory	---	Industry	---	---	---
StyleCop	---	Industry and Community	---	---	---
Tool to detect TTCN-3 Test Smells	---	Industry	---	---	---
XRefractory	---	Industry	---	---	---

Tabla 8. Listado Inicial. De Tahvildari and Kontogiannis a XRefractory.

1.4.1.2. Distribución temporal

La planificación temporal del proyecto se realiza de acuerdo al anterior listado de tareas. Por ello, y debido a que en el sitio web se muestran los resultados obtenidos en la etapa de análisis de las herramientas, no se realiza ninguna tarea relacionada con el sitio web hasta finalizar dichos análisis. La planificación general es la mostrada en el diagrama de Gantt de la *figura 1*. Los principales puntos que la planificación incluye son:

- Planificación razonada de tiempos de desarrollo, entregables, plazos e hitos.
- Reuniones de proyecto: Al menos, el grupo se reunirá cada lunes, miércoles y viernes por las mañanas de 10 a 13. El resto de días no es posible por la asistencia a clase de uno de los alumnos. Se parará en Navidades y hasta el 20 de Enero debido a los exámenes de uno de los miembros del grupo durante comienzos de dicho mes. En caso de necesidad se producirán reuniones por las tardes durante el tiempo que se estime necesario pero inicialmente se evitarán para así dejar tiempo para el estudio al alumno con asignaturas pendientes.
- Herramientas de desarrollo: Inicialmente no se parte de ninguna herramienta especial dado que no todas las herramientas trabajan bajo un mismo entorno de desarrollo o programa; ni tan siquiera trabajan bajo un mismo sistema operativo.
- Elección de tecnologías de desarrollo y frameworks, y herramientas: Fase que aparece únicamente en la etapa de desarrollo del portal web ya que, como ya se ha indicado, en la etapa de análisis de herramientas no se dispone de un único entorno de trabajo. Consiste principalmente en la elección del CMS a utilizar para el desarrollo.
- Configuración del sitio web de alojamiento del proyecto.

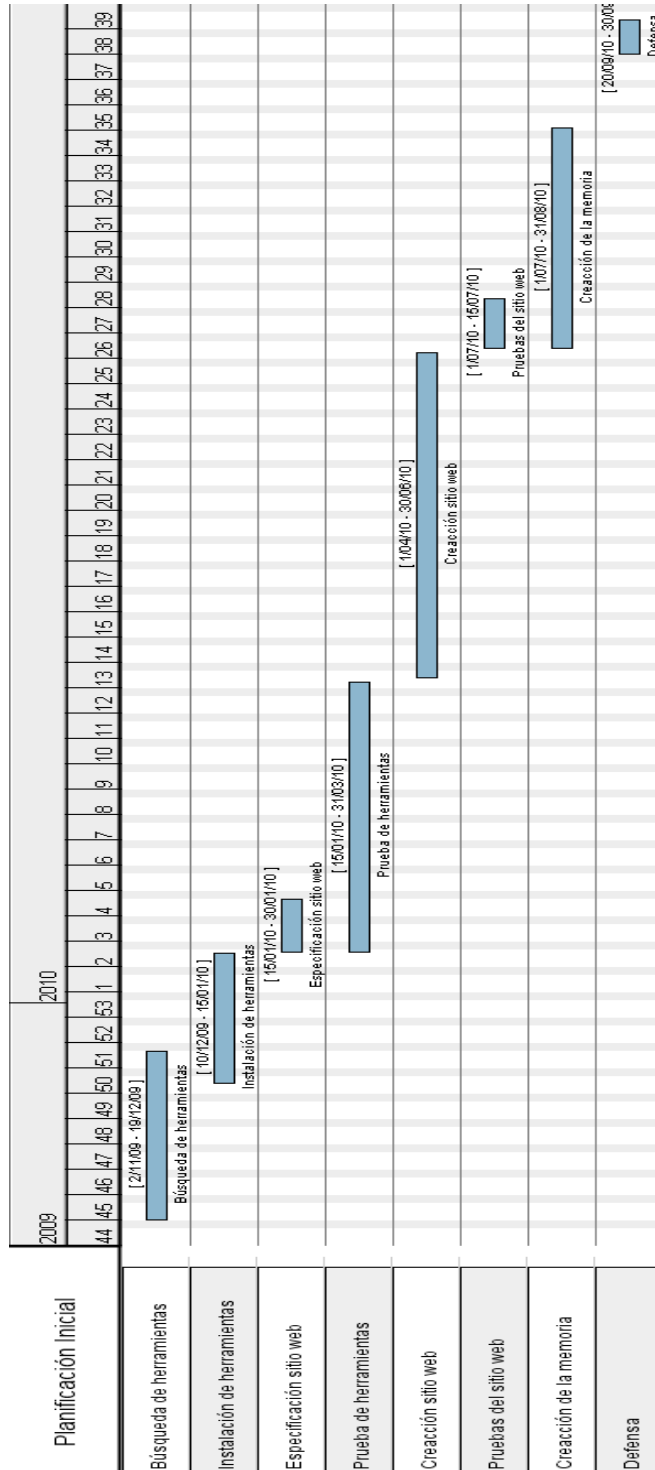


Figura 1. Planificación inicial del proyecto.

1.4.1.3. Seguimiento del proyecto

El seguimiento general del proyecto se muestra en el diagrama de Gantt de la *figura 2* que se puede encontrar en la página siguiente. En él se puede observar que las fechas establecidas inicialmente (ver *figura 1*) han sufrido algunos retrasos, sobre todo debido a que en verano se ha perdido más tiempo del inicialmente previsto por motivos personales y a que a comienzos de 2010 se tardó más de lo inicialmente esperado en disponer del artículo que debíamos seguir para efectuar el análisis. Donde más cambio temporal se aprecia es en la parte relacionada con el sitio web debido a que al iniciar el análisis de herramientas se decide dejarlo de lado para así centrar el trabajo únicamente en las herramientas. Por esto, no se inicia todo lo que tiene que ver con la web hasta finalizar el análisis de las herramientas y la parte de la memoria correspondiente a dicho análisis.

A continuación se va a detallar más en profundidad el seguimiento temporal de la fase que se ha denominado de análisis de herramientas y que engloba lectura y comprensión del artículo, traducción del mismo, y búsqueda, instalación y análisis de las herramientas. Al comienzo del proyecto los plazos se cumplen, e incluso se acortan ligeramente en la etapa de búsqueda de herramientas ya que la mayoría de ellas resultan más sencillas de obtener de lo esperado, aunque algunas se resisten y quedan pendientes para más adelante, siendo necesario incluso contactar con los autores para ver si es posible disponer de la herramienta en cuestión. A la hora de la instalación aparecen ciertos problemas, que unidos a los retrasos ya previstos por los exámenes en Enero de uno de los autores del proyecto, hacen que no se pueda dar por cerrada la etapa de instalación hasta comienzos del mes de Marzo de 2010, cuando se obtiene CodeClinic, aunque se finaliza la instalación del grueso de las herramientas a finales del mes de Enero. Tras la instalación de todas las herramientas disponibles hasta enero, surge el problema de que el artículo que marca la base del análisis a efectuar a cada una de dichas herramientas aún no ha sido finalizado por sus autores por lo que no se puede comenzar el análisis y hay un periodo de espera. El artículo se recibe a mediados de Febrero y aparece un nuevo inconveniente que no se tuvo en cuenta en la planificación inicial, el artículo se encuentra en inglés y dado que contiene gran cantidad de palabras técnicas y que es la principal guía para la elaboración de este proyecto, es necesario realizar una cuidadosa traducción del mismo. Todo esto provoca que la etapa de análisis de herramientas se inicie ya entrado el mes de Marzo, con tres meses de retraso respecto a lo inicialmente previsto. La etapa se finaliza a principios del mes de Julio y se inicia la elaboración detallada de la memoria en la parte correspondiente a esta fase. Como ya se ha comentado, diversos motivos personales obligan a que durante el mes de Julio y los primeros días del mes de Septiembre apenas se pueda avanzar, por lo que la creación de la parte de la memoria correspondiente a lo realizado hasta ese momento se retrasa hasta finales de Septiembre. Tras esto, se inician las tareas relacionadas con la elaboración del sitio web y su parte de la memoria correspondiente.

En la *figura 3* se ve de forma detallada la parte de desarrollo del sitio web. Se incluye una imagen específica para esta parte debido a que contiene gran cantidad de hitos no incluidos en el seguimiento general. La fase comienza con la especificación de requisitos web a comienzos del mes de Octubre y se da por finalizada a mediados del mismo mes. Tras esto, se da un tiempo de espera para obtener la validación de dichos requisitos por parte de los tutores de este trabajo y mientras se decide cómo se va a realizar el sitio web. Tras decidir usar un CMS, concretamente Joomla, se inicia la creación del sitio web a finales del mes de Octubre y se prolonga hasta mediados de Diciembre.

El proceso de desarrollo lleva tiempo debido a que es necesario estudiar el funcionamiento de Joomla y llevar a cabo su instalación en el servidor a utilizar, así como los módulos necesarios para

el funcionamiento de la web, como pueden ser el del Login de usuarios y el de los comentarios. Además hay que escribir los artículos y realizar las tablas de cada una de las herramientas analizadas. Las pruebas del sitio web se inician unos días antes de que acabe la fase de creación ya que desde un primer momento la web está operativa y va recibiendo contenido según se crea, y se dan por finalizadas con la llegada de las Navidades.

Durante todo el tiempo que pasa desde el inicio del proceso de creación de la web hasta la finalización de las pruebas de funcionamiento se va creando también la parte de memoria correspondiente a toda la fase de desarrollo de la página web.

Tras la finalización de todas las tareas principales del proyecto, se inicia la revisión de la memoria con la entrada del 2011 para presentarsela a los tutores y posteriormente proceder a la defensa del mismo.

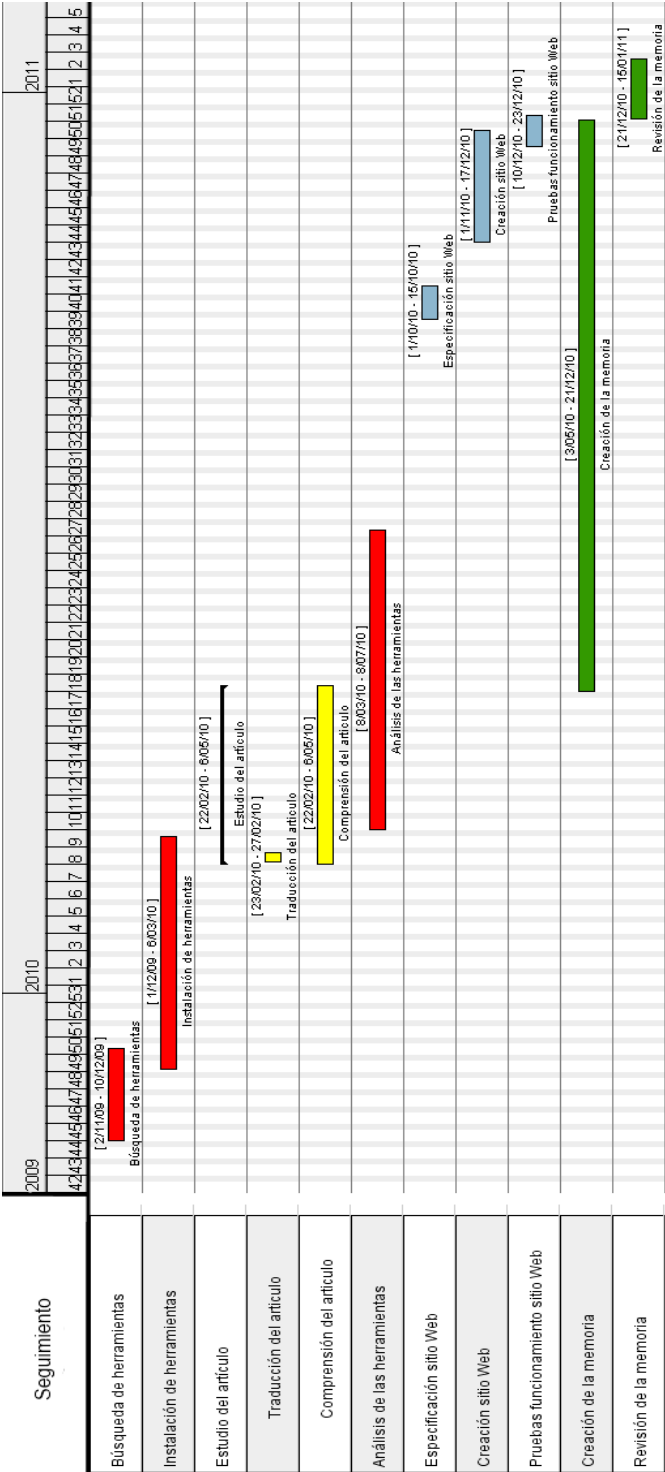


Figura 2. Seguimiento general del proyecto.

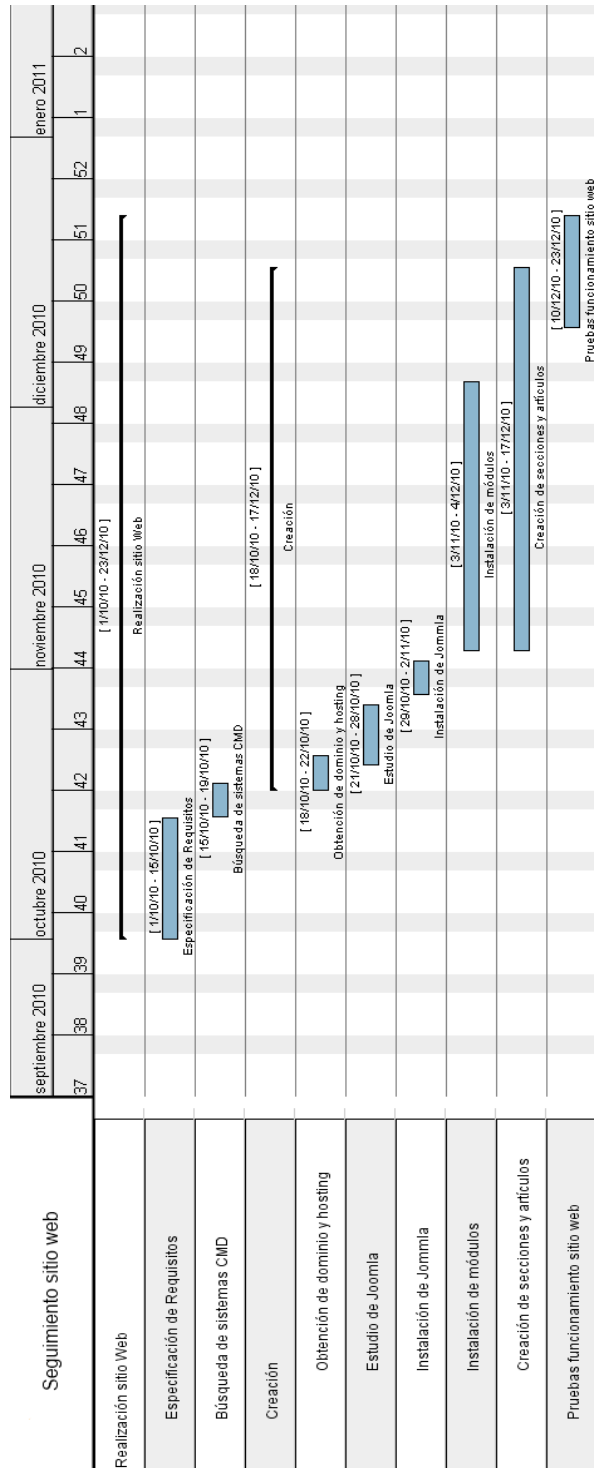


Figura 3. Seguimiento específico de la parte de desarrollo del sitio web.

Capítulo 2

TAXONOMÍA

2.1. Introducción

El análisis de las herramientas se basa en el artículo de Javier Pérez, Nahuel Moha, Tom Mens y Carlos López, *A classification framework and survey for design smell management* [1], que fue proporcionado para caracterizar cada una de las herramientas según unos criterios y características que, a modo de referencia, y para que este proyecto sea autocontenido, se plasman a continuación de una forma similar a como aparece en la fuente original.

2.2. Taxonomía

“La taxonomía que se propone para la gestión de smells permite agrupar actividades, herramientas, técnicas o formalismos basándose en sus puntos en común. Para describir estos puntos en común, se usa una clasificación multidimensional, que permite agrupar y comparar los diferentes enfoques, basados en los criterios de interés. Cada una de las siguientes subsecciones investigará un criterio importante con respecto a la gestión de smells que puede utilizarse para agrupar los enfoques que satisfacen este criterio” [1].

Dentro de la clasificación, no se consideran las propiedades que pueden aplicarse a cualquier tipo de herramienta. Es decir sólo se consideran aquellas específicas de las herramientas destinadas a la detección y corrección de defectos de diseño.

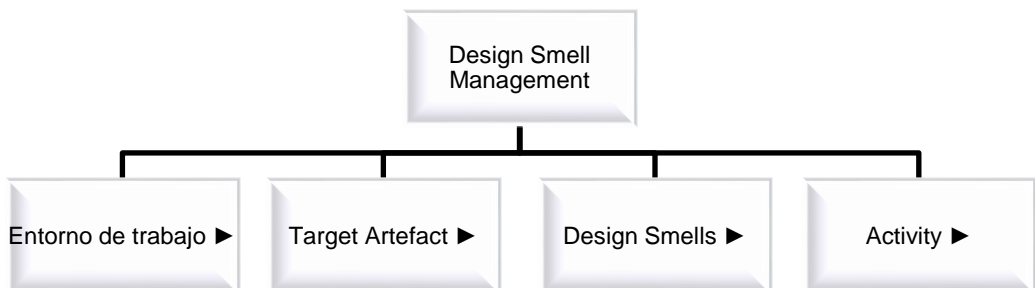


Figura 4. Raíz del diagrama de características.

Para presentar la taxonomía, se comienza por la descripción de las características de nivel superior (ver figura 4). Luego se descenderá por el modelo para describir cada una de las subcaracterísticas.

La característica raíz del modelo es "*Design Smell Management*". Representa cualquier enfoque o herramienta relacionada con la gestión de *smells*. Una instancia del modelo representa la encarnación de un enfoque o herramienta existente, o incluso una no existente que podría ser factible e interesante para desarrollar. Las características a nivel inferior de "*Design Smell Management*" se muestran en la figura 4 y se explican a continuación. Éstas características son las que posteriormente sirven para dividir el análisis en tres fases principales ya que las tablas en las que se centra el trabajo expuesto en esta memoria se basan en cada una de estas tres subcaracterísticas.

2.2.1. Entorno de trabajo

Esta es una característica que no se encuentra en el estudio previo facilitado y que ha sido añadida porque se considera que puede ser una característica importante a tener en cuenta a la hora de elegir una determinada herramienta para analizar o corregir código. No es lo mismo una herramienta independiente que una que actúa como plugin de algún IDE como es Eclipse ya que si no se dispone de dicho IDE instalado el usuario se verá obligado a su instalación y, posteriormente, a la instalación del plugin. Por otro lado, la mayoría de las herramientas independientes requieren una única instalación aunque algunas ni siquiera lo necesitan, siendo posible ejecutarlas desde un sitio web.

Durante el análisis también aparece detallado el Sistema Operativo bajo el que trabajan puesto que también puede ser una característica determinante a la hora de elegir herramienta ya que si únicamente se dispone de por ejemplo un sistema UNIX, es lógico que se intente elegir una herramienta adecuada a este sistema para así evitar la necesidad de instalación de un nuevo Sistema Operativo, con todo el coste (en tiempo y tal vez en dinero) que esto requiere.

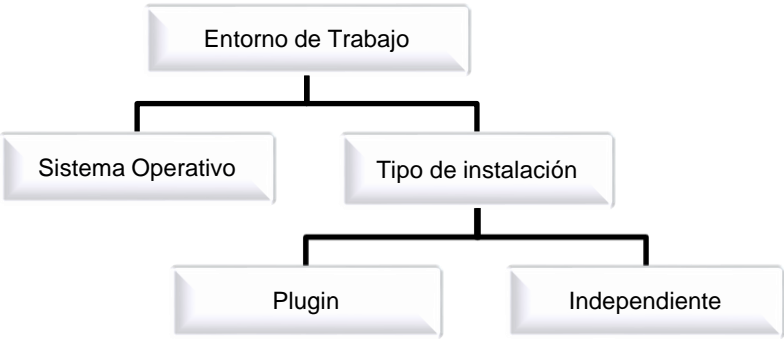


Figura 5. Característica Entorno de Trabajo y sus sub-características.

A la hora de representar esta característica, y dado que su contenido es mínimo, se decide incluirla dentro de la tabla que representa la característica *Target Artefact*, ya que aunque no tenga nada que ver con la definición de la característica, esta tabla es la que hace una presentación de la herramienta en cuanto a que indica qué tipo de código analiza y que lenguajes de programación soporta, lo que unido al Sistema Operativo y el tipo de instalación influirá mucho a la hora de elegir la herramienta que más se adecúa a las necesidades de cada usuario en cada momento.

2.2.2. Target Artefact

“Propiedad común y obligatoria para todos los enfoques de gestión de smells. Cualquier planteamiento que aborde la gestión de los malos olores de diseño debe centrarse en, al menos, un tipo de artefacto software. Por lo tanto esta es una característica obligatoria” [1]. En esta memoria se define a este tipo de artefacto como el *"Target Artefact"* (Artefacto objetivo) del enfoque. Todas las propiedades que un enfoque o una herramienta deben incluir con el fin de dar soporte a un determinado *Target Artefact* son subcaracterísticas de ésta.

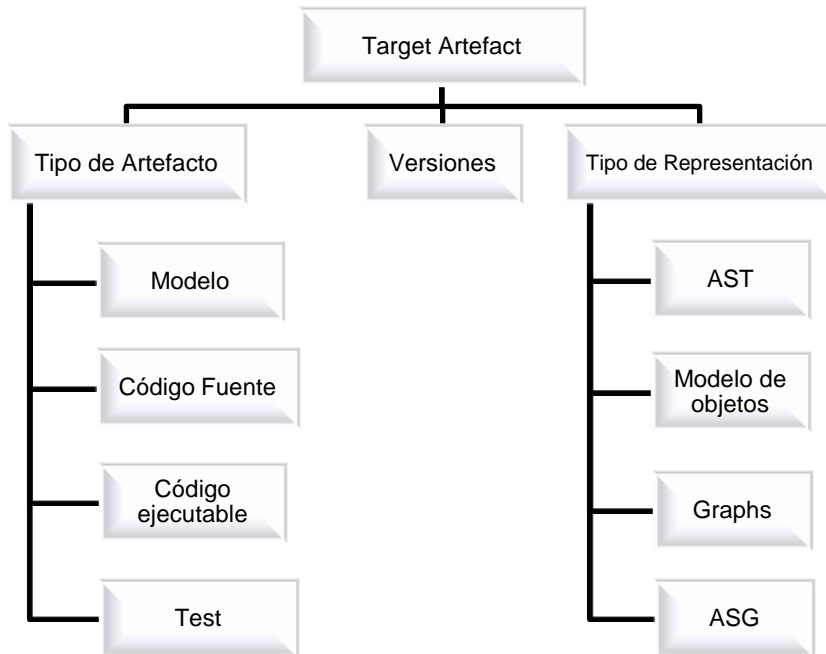


Figura 6.Característica Target Artefact. Subcaracterística de *Design Smell Management* y sus subcaracterísticas.

Entre estas subcaracterísticas se encuentran, como puede verse en la *figura 6*, Tipo de artefacto, Versiones y Tipo de representación. A continuación se explica brevemente cada una de ellas:

- Tipo de Artefacto:

Se denomina Tipo de Artefacto a lo que el programa necesita que se le facilite para que lo analice. Esta es una característica obligatoria ya que *“cualquier herramienta de las que se analiza debe ser capaz de hacer frente a los smells en al menos un tipo de artefacto software”* [1]. Una herramienta normalmente soporta un sólo tipo de artefacto aunque no se puede descartar pensar en desarrollar una herramienta de apoyo a muchos tipos diferentes de artefactos software de forma simultánea (a la hora de realizar esta memoria no se tiene conocimiento de la existencia de ninguna herramienta así, aunque no se puede descartar que no exista ya o que no sea posible su creación). *“El tipo de artefacto se descompone en un conjunto de subcaracterísticas que se pueden considerar de tipo OR”* [1], es decir, la herramienta debe de ser capaz de analizar o corregir al menos un tipo de artefacto software. Los tipos de artefacto más comunes son modelos (*Model*), código fuente (*Source Code*), código ejecutable (*Executable Code*) o pruebas (*tests*).

Algunas de las herramientas disponibles tienen por objetivo la gestión de *smells* en el código fuente. *“Checkstyle [19], por ejemplo, carga código Java y busca violaciones de normas de código”* [1]. Muchos otros lenguajes de programación también pueden soportar la búsqueda de *Design Smells*, como hacen por ejemplo Reek [20] y Roodi [21], que buscan problemas de diseño en código fuente Ruby. FxCop [22] y StyleCop [23] encuentran desviaciones de los convenios de código en código C #, etc.

Otro tipo de artefacto comúnmente soportado es código ejecutable (Código binario, *bytecode*, etc.). Este tipo de enfoques permiten no sólo la comprobación de código estático, sino también el análisis dinámico. RevJava [24] es un ejemplo de herramienta que analiza Java *bytecode*.

“Una nueva tendencia en algunos campos de software, es analizar también las pruebas (test), en concreto las de secuencia de comandos” [1].

- Versiones:

“Una herramienta de gestión del diseño puede beneficiarse de la información adicional que puede ser extraída de un repositorio, que almacena varias versiones del artefacto objetivo en estudio” permitiendo ver la evolución entre las diferentes versiones. *“Algunos bad smells, como Shotgun Surgery son más fácilmente identificables por el análisis del historial de cambios del sistema” [1].* Ésta es una subcaracterística opcional que no es necesaria para la detección y corrección y que puede implicar a cualquier tipo de artefacto

Las herramientas basadas en esta característica pueden tener diferentes versiones del artefacto objetivo como entrada. Las herramientas que soportan dicha característica a menudo incluyen el apoyo de repositorios de versiones software (por ejemplo, CVS, SVN, etc.). El análisis de las múltiples versiones puede claramente ayudar a afinar bien las estrategias de detección de *smells*.

- Tipo de Representación:

“A fin de analizar y operar con el artefacto objetivo, cualquier enfoque debe utilizar una representación interna de él. Una manera común de representar un artefacto software es por su árbol de sintaxis abstracta (AST). Este tipo de representación es especialmente frecuente en herramientas y enfoques que analizan código ejecutable. Una representación típica AST mantendrá la información completa disponible en el artefacto examinado, pero mediante distintos enfoques se puede simplificar el AST para mantener sólo la información necesaria o incluso aumentarla con detalles adicionales a fin de facilitar las tareas de gestión de smells” [1]. Algunos ejemplos son jDeodorant [30] o PMD. Durante el análisis de las herramientas incluidas en esta memoria aparece una técnica similar al AST y que no se comenta en el artículo base (Pérez *et al.* [1]). Se trata del “gráfico de semántica abstracta” (ASG, *Abstract Semantic Graph*), que se construye normalmente a partir de un árbol de sintaxis abstracta (AST) mediante un proceso de enriquecimiento y abstracción. *“El enriquecimiento puede ser, por ejemplo, la adición de punteros desde los bordes de un identificador de nodo donde se utiliza una variable a un nodo que representa la declaración de esa variable. La abstracción puede implicar la retirada de detalles que sólo son pertinentes en el análisis, no para la semántica” [31].*

Otras herramientas usan una representación interna basada en un *Modelo de Objetos (Object Model)*. Este tipo de representación se utiliza sobre todo para simplificar el artefacto objetivo ya que sólo se extrae de él la información que necesita la herramienta. *“El análisis y manipulación del Target Artefact se realiza por medio de procedimientos programados, haciendo uso de una amplia variedad de lenguajes de programación, e incluso auto-desarrollando lenguajes de dominio específico” (DSL, Domain Specific Languages) [1].*

Los enfoques basados en grafos (*Graph*) también pueden ser empleados para analizar el artefacto objetivo en la búsqueda de *Design Smells*. La teoría de grafos puede ayudar a analizar los artefactos en busca de defectos y técnicas para aplicar correcciones.

Las *fórmulas lógicas (Logic Formulas)* ofrecen apoyo para representar sistemas orientados a objetos. Este tipo de representaciones permite utilizar las técnicas basadas en la lógica para gestionar *smells*.

“Estos diferentes tipos de representación se pueden combinar. Un artefacto representado como un AST puede ser analizado con algoritmos de grafos (si se formaliza con grafos) o con fórmulas lógicas. Un modelo puede ser almacenado en una base de datos relacional con el fin de proporcionar una forma fácil y eficiente para acceder a él consultando la base de datos, etc.” [1].

Esta característica de las herramientas, al tratarse de una representación interna del artefacto, es muy difícil de ver por lo que cuando se realiza el análisis hay que confiar en lo que sus autores indican en la documentación ya que de lo contrario, para averiguarlo, habría que analizar miles de líneas del código fuente de las herramientas con lo que el trabajo se prolongaría demasiado en el tiempo. Además, no en todas las herramientas es posible disponer del código fuente y no todas están escritas en un lenguaje conocido por los autores de esta memoria.

2.2.3. Design Smells

La característica *Design Smells*, representada en la figura 7, modela los *smells* que una herramienta puede abordar. Esta es una característica muy importante para efectuar el estudio ya que la mayoría de las aplicaciones analizadas están especializadas bien en un determinado tipo de *smell*, bien en *smells* que pueden detectarse con un indicador en particular, o bien en *smells* que pertenecen a un tipo específico de entidad de programa. Esta característica también depende de la técnica particular utilizada por la herramienta.

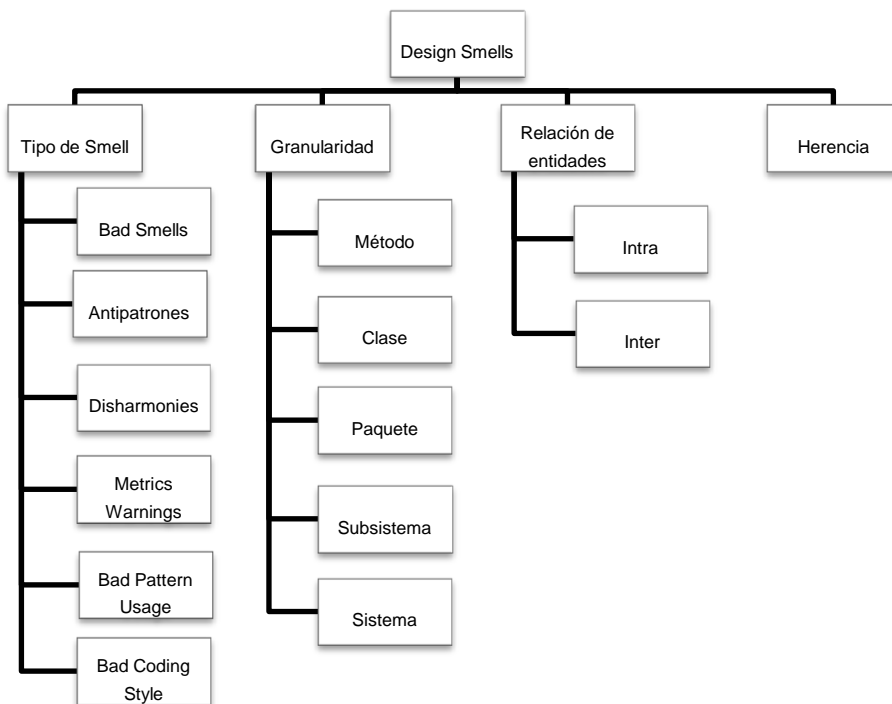


Figura 7. Característica Design Smell. Subcaracterística de *Design Smell Management* y sus subcaracterísticas.

Para representar el *Design Smell*, “se divide la característica en cuatro ramas de funciones agrupadas, y una función opcional que especifica si la relación de herencia es relevante para el *smell* o no” [1]. Esta clasificación de *Design Smells* está basada en las de [2] y [32].

a. Tipo de Smell

Esta característica describe la naturaleza del *smell* a abordar. Puede abarcar desde simples advertencias de métricas (*Metrics Warnings*) a problemas de diseño descritos en detalle en los catálogos de *smells*. “Los problemas en el diseño del sistema se especifican en términos de simples advertencias o violaciones cada vez que una métrica determinada excede un umbral determinado. Las herramientas que utilizan este tipo de definición de *smell* suelen producir listas de filtrado de las entidades que violen de manera relativa o absoluta el umbral de las métricas” [1]. Algunos ejemplos son: RefactorIt [1, 33], Eclipse Metrics Plugin [1, 34], JRefactory [35].

La mayoría de las herramientas estudiadas están dirigidas a corregir los *Bad Smells* que se especifican en el catálogo de Fowler [6]. La herramienta Analyst4j [36] se utiliza para buscar antipatronos. La herramienta de detección de *smells* iPlasma [37] ha sido desarrollada para detectar *disharmonies*.

Algunas herramientas usan patrones de diseño para la búsqueda de los *smells*, ya sea para buscar oportunidades para aplicar un patrón de diseño o para detectar la mala aplicación de los patrones, identificado dentro de nuestras subcaracterísticas como *bad pattern usage*.

“Algunos *Design Smells* están relacionados con problemas que incluyen el mal estilo de codificación (como el código no utilizado, un único órgano de la declaración de una sentencia *if* que no está encerrado entre llaves, etc.) o la violación de las convenciones de estilo o normas de codificación (como la guía de convenciones de código de Sun para Java, o las convenciones internas establecidas por una empresa particular, comunidad o proyecto)” [1]. Estos problemas se denominan como *bad coding style*. Algunas de las herramientas relacionadas con este tipo de problemas son PMD, XRefactory [38] y Checkstyle. Estas herramientas suelen permitir la definición de nuevas reglas de detección de problemas basados en la métrica y las expresiones regulares.

b. Granularidad:

“Esta característica se utiliza para describir los distintos tipos de entidades que participan en los *smells* soportados. Por ejemplo, para el código fuente de Java, los diferentes niveles de granularidad podrían ser sistema, subsistema, paquete, clase y método” [1]. Para otro tipo de código esta subcaracterística no se tiene en cuenta ya que no se sigue la misma jerarquización de Java y no se puede detallar un nivel de granularidad por desconocimiento del lenguaje.

c. Relación entre entidades:

Se refiere a si el *smell* se produce entre entidades del mismo tipo (*inter*) o entre entidades de distintos tipos (*intra*).

d. Herencia:

Las herramientas que soportan la herencia, lo hacen como una característica opcional y tratan la gestión de *smells* que implican relaciones de herencia entre las entidades consideradas.

2.2.4. Actividad

La característica *Actividad*, mostrada en la *figura 8*, representa un gran punto de variación entre la distinta gestión de los *Design Smells* de las herramientas. Se ha descompuesto esta característica en cinco subcaracterísticas que todo enfoque o herramienta debería tener: *Tipo de actividad*, *Técnica*, *Resultado*, *Formato de Salida* y *Automatización*. Cada una de estas subcaracterísticas se explica a continuación.

a. Tipo de Actividad

“El diseño del proceso de gestión de smells puede ser descompuesto en diferentes tipos de actividades que necesitan ser apoyados por un enfoque particular o herramienta. Un enfoque o una herramienta se dice que soporta un cierto tipo de actividad cuando se toma en cuenta explícitamente, independientemente del grado de automatización” [1].

- Especificación:

Esta característica aparece implementada en las herramientas que ofrecen a los desarrolladores el apoyo necesario para ampliar o adaptar a sus necesidades particulares, mediante la definición o la personalización, las normas utilizadas para la detección, corrección, etc. Varios enfoques permiten definir de manera escrita por el usuario las normas de detección o de corrección, como PMD [29]. La capacidad de definir o ajustar las reglas de detección es común a los enfoques que se ocupan de los smells sobre la base de las advertencias métricas. Algunos de los enfoques ofrecen además apoyo explícito en cuanto a cómo ha de ser especificada la norma de corrección o detección.

- Artefact Analysis:

Esta actividad se refiere a la extracción de información relevante sobre el artefacto objetivo, que será utilizada para otras actividades. Se incluyen la extracción de un modelo adecuado del artefacto objetivo para construir su representación interna, el cálculo de la métrica, etc.

- Detección:

La gran mayoría de las herramientas existentes se centran en la actividad de detección de *Design smells*, para la cuál se presta un apoyo explícito. Algunas herramientas en las que se presenta esta característica pueden ser jDeodorant y DECOR [39].

- Visualización:

La actividad de visualización, que normalmente requiere la intervención humana para ser mostrada, produce algún tipo de representación gráfica del artefacto de destino, permitiendo la identificación rápida y fácil de algunas de sus propiedades. Muchas herramientas presentan la visualización en el contexto de la gestión de *Design Smells*. Una herramienta de visualización también puede proporcionar otros tipos de información. Así, puede ayudar al desarrollador a decidir cuáles son las mejores modificaciones para eliminar un determinado *smell* o puede ser utilizada para evolucionar o explicar las causas y los impactos de los *smells*. También se suele proporcionar un resumen visual de las propiedades y características del sistema. Algunos ejemplos de visualizaciones en herramientas son Structure101 [65] y STAN [40].

- Corrección:

Las herramientas o enfoques de apoyo a la actividad de corrección, proporcionan una forma de modificar el artefacto objetivo (o sugieren cómo hacerlo), a fin de mejorar su diseño. El soporte a la actividad de corrección en la actualidad está menos desarrollado que el soporte para la

actividad de detección, por lo tanto existen menos herramientas que lo tengan implementado. Algunas herramientas de apoyo a la corrección de Smells son jDeodorant y TREX [41].

“La mayoría de las herramientas aplican estrategias de refactorización a los smells que han sido detectados previamente como sucede en DECOR. Un enfoque alternativo es sugerir refactorizaciones basándose únicamente en los valores del sistema métrico o en la presencia de ciertos patrones, pero sin identificar de manera explícita el Design Smell” [1].

- Análisis de impacto:

Las herramientas que soportan la actividad de análisis de impacto dan una forma de calcular el impacto de un posible *smell* o de las acciones realizadas para eliminarlo, por ejemplo jDeodorant.

- Verificación:

“Las herramientas que soportan la actividad de verificación ofrecen soporte para analizar y verificar los resultados producidos. Esto podría dar lugar a herramientas que son capaces de mejorar gradualmente sus resultados de forma automática. Esta actividad abarca también los posibles mecanismos para mejorar la herramienta mediante el refinado de forma incremental de los resultados de detección o corrección con la información obtenida en su uso diario. La mayoría de las herramientas necesitan ser asistidas por el usuario para verificar y confirmar sus resultados” [1].

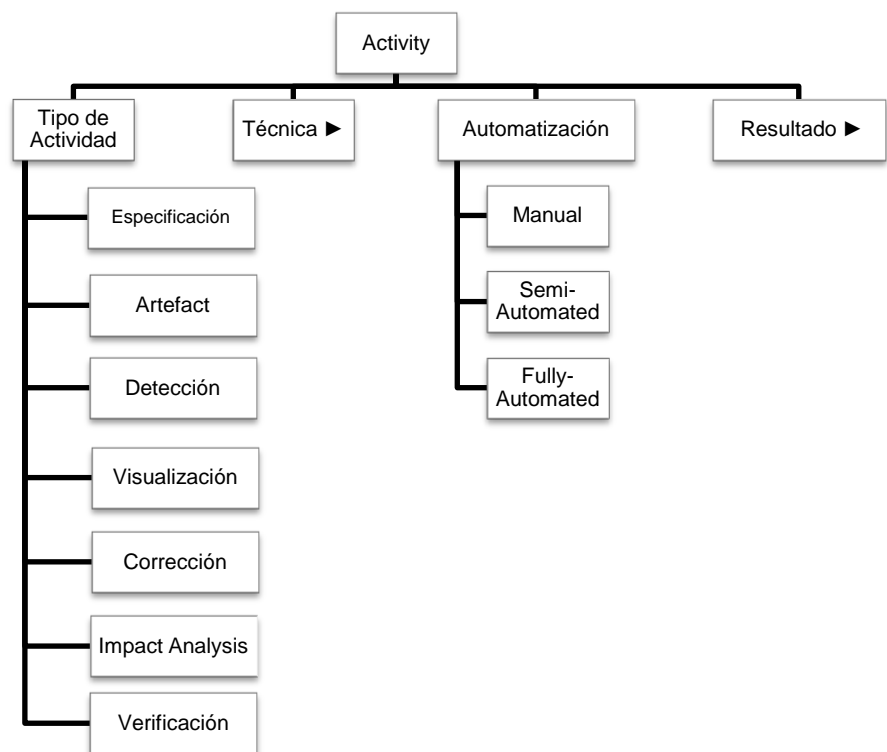


Figura 8. Característica Actividad. Subcaracterística de *Design Smell Management* y sus subcaracterísticas.

b. Técnica

Una segunda subfunción obligatoria de la característica *Actividad* es la técnica en la que el enfoque se basa para soportar una determinada actividad. Esta característica se muestra en la figura 9. Puesto que existe una gran variedad de técnicas que se utilizan, o pueden serlo en el futuro, se deja esta opción abierta.

- Descripción del Proceso:

La mayoría de los enfoques estudiados en [1] incluyen una descripción sobre el proceso de gestión del Design Smell para las actividades que soporta. “*Dependiendo de la formalidad y la integridad de esta descripción, podrán implementarse diferentes subcaracterísticas. Las especificaciones de Design Smells [3], incluyen las directrices generales para corregirlos. El libro de Wake [41] reestructura estas pautas de corrección para escribirlas en forma de recetas. Esto comprende un conjunto de libros de cocina con descripciones de procesos. El libro de antipatrones [42] es un ejemplo de definición de procesos, dados en forma de contraejemplos. Si la herramienta describe explícitamente un proceso sistemático para realizar la actividad de gestión del Design Smell, se implementa la característica modelo de proceso de la subcaracterística process description*” [1].

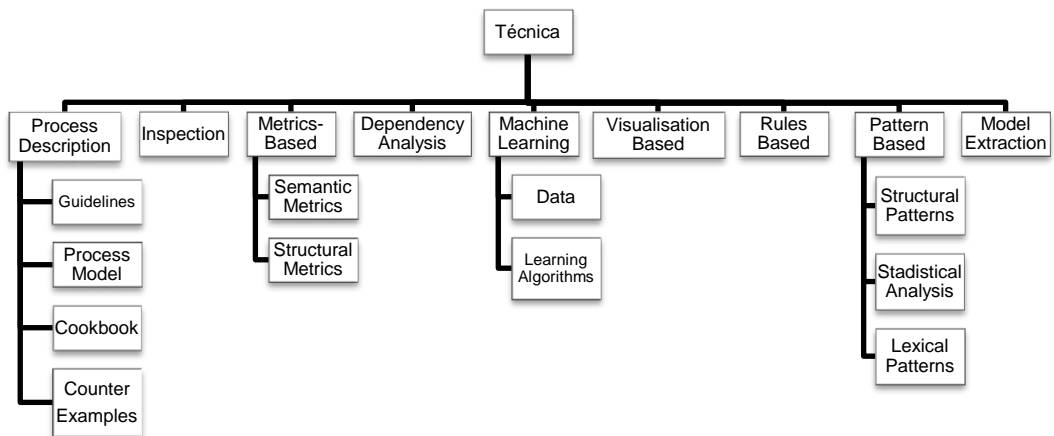


Figura 9. Característica Técnica. Subcaracterística de Actividad y sus propias subcaracterísticas.

- Inspección:

“Las técnicas que se utilizan para explorar manualmente los target artefact” [1].

- Metrics-Based:

La mayoría de los enfoques se basan principalmente en métricas. “Se pueden encontrar dos enfoques diferentes. La gran mayoría de los enfoques y herramientas basados en métricas se basan en los indicadores estructurales, como sucede en TRex [43]. Pero algunos de los enfoques recientes están teniendo en cuenta también parámetros semánticos” [1].

- Machine Learning:

“Las técnicas de la familia Machine Learning se utilizan también para apoyar la gestión de Design Smells. Algunas herramientas pretenden aplicar data mining para la detección de Design Smells [44], o [45,46], y también apoyar la corrección a través de técnicas basadas en algoritmos de aprendizaje” [1].

- Basado en patrones:

“Dar soporte a un modelo basado en la técnica Pattern-Based implica que una actividad se realiza mediante identificación, modificación o creación de instancias de los patrones de algún tipo, que pueden ser patrones estructurales, tales como los patrones de diseño, los modelos de análisis estadístico o patrones léxicos” [1].

- Análisis de dependencias:

Algunos enfoques se centran en detectar *smells* relacionados con problemas de dependencias, como las dependencias cíclicas. Este es el caso de Cultivate.

- Visualisation-Based:

Las herramientas cuya técnica principal para hacer frente a algunas actividades de gestión de *Design Smells* se basa en las técnicas de visualización están representadas por esta función. La detección de *Design Smells* se lleva a cabo mediante su identificación en las representaciones visuales del artefacto objetivo.

- Rule-Based:

La mayoría de las herramientas que soportan la detección o corrección se basan en reglas que pueden ser ejecutados sobre el artefacto de destino o de su representación interna, como ocurre en DECOR [47,48].

c. Automatización

Una subcaracterística de la función de actividad es la medida en que la herramienta proporciona soporte de automatización. Esta característica, representada en parte de la *figura 8*, se puede utilizar para describir que grado de automatización proporciona una herramienta para el tipo de actividad, *“o para describir incluso si un planteamiento no está automatizado, pero podría estarlo en el futuro”*. Se distinguen los siguientes niveles diferentes de automatización:

- **Manual:** La herramienta realiza el análisis gracias a la ayuda del usuario, ya que tiene que configurar ciertos parámetros necesarios para poder realizar dicho análisis.

- **Semiautomática o Interactiva:** La herramienta se encarga de realizar el análisis de forma automática, pero en determinados momentos puede pedir al usuario alguna aprobación relacionada con el análisis realizado para continuar, o sugerir alguna alternativa para que el usuario pueda interactuar.

- **Automático:** La herramienta realiza el análisis de forma automática sin necesidad del que el usuario realice ninguna configuración previa.

d. Resultado

“Esta subcaracterística de la función Actividad, refleja el hecho de que cualquier herramienta de gestión de design smells debe producir algún resultado cuando se aplica a algún artefacto software. Esta característica permite dar información de los resultados obtenidos del modelo. El análisis de una herramienta de acuerdo a esta dimensión es importante para determinar si ésta es útil para resolver un problema particular, o si se puede integrar con otro método o herramienta” [1]. La subcaracterística Resultado se muestra a continuación en la *figura 10*.

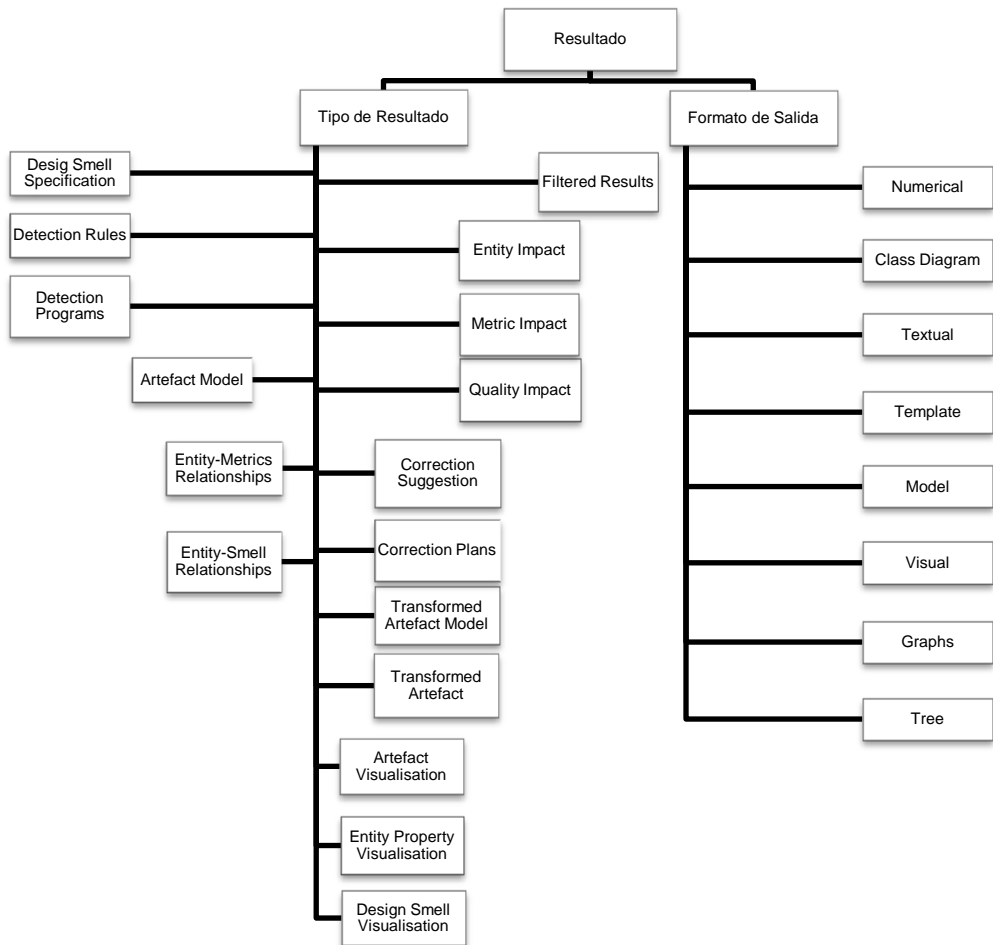


Figura 10. Característica Resultado. Subcaracterística de Actividad y sus propias subcaracterísticas.

- Tipo de Resultado:

Dependiendo de las actividades soportadas por la herramienta, esta será capaz de producir diferentes tipos de resultado. A continuación se explicará brevemente la disposición de algunas de las subcaracterísticas de Tipo de Resultado incluidas en la *figura 10*.

La actividad de **especificación** se refiere a si la herramienta permite modificar las opciones de búsqueda del *Design smell*, bien sea mediante una especificación de éste (*design smell specification*), o bien de cómo realizar su detección (*design smells detection*) o incluso mediante la carga de un programa de detección creado previamente (*detection program*).

La actividad de **análisis del artefacto** siempre produce el mismo tipo de resultado: un modelo del artefacto objetivo, que después será el que la herramienta utilice para el análisis.

La actividad de **detección** muestra, en la mayoría de los casos, *entity metric relationships* o *entity smell relationships* para cada *smell* detectado. Los resultados pueden presentarse en muchos tipos de formas textuales y gráficas.

La mayoría de las herramientas de revisión que se apoyan en la actividad de **visualización** proporcionan diferentes tipos de diagramas, a fin de ayudar a la comprensión del artefacto objetivo. Otros tipos de ayudas gráficas en el diseño de *entity smells* incluyen la visualización de propiedades, el tipo de *smell* y la visualización del diseño del artefacto objetivo.

La actividad de **corrección** puede producir diferentes tipos de resultados en función, principalmente, del grado de automatización que proporciona cada herramienta en particular. Algunas herramientas pueden proporcionar sólo sugerencias de corrección, mientras que otras pueden dar algunos planes de corrección, o las especificaciones de la secuencia de transformación necesaria para mejorar el diseño del artefacto objetivo. *“Otras herramientas pueden aplicar los cambios a su representación computarizada del artefacto interno y por tanto producir un modelo de artefacto transformado. Esto sucede, por ejemplo, con las herramientas totalmente automatizadas, que pueden operar directamente sobre el artefacto objetivo generando un artefacto transformado”* [1].

El resultado de la actividad de **verificación** afecta a los cambios generados por otras actividades, obteniendo así resultados filtrados.

- Formato de salida:

Cada resultado producido por una herramienta puede ser presentado en diferentes formatos de salida. Esta característica también es relevante, ya que revela las posibilidades de comunicación entre las diferentes actividades dentro de la misma herramienta, y por supuesto, entre las diferentes herramientas.

Capítulo 3

ANÁLISIS DE HERRAMIENTAS

3.1. Búsqueda de herramientas

La fase de búsqueda de herramientas consiste en realizar una búsqueda en la web de las herramientas facilitadas en el listado inicial. Como se puede ver en las *tablas 7 y 8*, hay herramientas marcadas como disponibles y otras como no. Se puede observar que también aparecen algunas sombreadas en azul (Herramientas ya analizadas en [1]), en verde (herramientas de tipo *lint* que no se deben tener en cuenta por que van más en la línea de detección de lo que se pueden llamar bugs y no de detección de defectos) y en rojo (Herramientas que han sido descartadas desde un primer momento por los tutores).

Esta primera fase se ha separado en varias búsquedas, una primera superficial y rápida para la totalidad de las herramientas (salvo las tachadas), incluyendo además del nombre de las herramientas, palabras clave como Design, Smell, Analysis, Tool o Source Code, y con la que se descartan Alikacem, Trifu, y otras similares por ser entradas referidas a los trabajos de dichos autores y no herramientas propiamente dichas. A continuación se realiza una segunda búsqueda más exhaustiva con las entradas marcadas como disponibles y que no han aparecido en la primera, contactando con los autores o distribuidores en caso de necesidad. Periódicamente se realizan nuevas búsquedas de las herramientas que no aparecen sombreadas y que no se hayan encontrado (o descartado) todavía para comprobar su disponibilidad.

Como resultado de todas estas búsquedas se obtiene que de las marcadas como no disponibles sólo Cultivate [9] y CodeClinic [10] han pasado a estarlo, la primera por vía web y la segunda facilitada por su autor (Adrian Trifu); mientras que de las marcadas como disponibles son varias las que han dejado de estarlo como es el caso de SemmleCode [11] o CodeCrawler [12]. Debido a la no disponibilidad de esta última y a que su propio autor recomienda el uso de XRay [13] como sustituta, se decide añadir ésta a la lista de herramientas a analizar. Otras herramientas como Optimal Advisor [14] se descartan debido a que son herramientas de pago con un alto coste económico (Aunque esta herramienta en el último acceso parece haber sido abandonada por parte de la compañía desarrolladora, Compuware) y finalmente otras como Goose [15] se descartan al estar obsoletas debido al desarrollo de una nueva herramienta, en este caso Sissy [16].

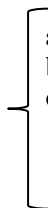
La tabla mostrada a continuación, *tabla 9*, muestra el listado definitivo de herramientas disponibles tras la fase de búsqueda.

Herramienta	Disponible	Observaciones
ArgoUML	SI	
Analyst4J	SI	
Argus CodeWatch	SI	
CodeClinic	SI	Herramienta desarrollada en una Tesis. Facilitada por su autor, Adrian Trifu.
CodeCrawler	NO	DESCARTADO: No disponible para descarga. Ha sido incluida en Moose como Mondrian.
CodeNose	NO	DESCARTADO: Herramienta desarrollada en una Tesis.
CodePro	SI	
CrocoPat	SI	
Cultivate	SI	
Eclipse	SI	
FxCop	SI	
GOOSE	SI	DESCARTADO: Herramienta antigua, se analiza Sissy que es su evolución.
Hammurapi	SI	
inCode	SI	
jCosmo	SI	
jDeodorant	SI	
JRefactory	SI	
M2 Resource Standard Metrics	SI	
OptimalAdvisor	NO	DESCARTADO: Imposible conseguir la herramienta al ser de pago.
PMD	SI	
Prodeoos	NO	DESCARTADO: Herramienta en desuso. La sustituyen herramientas como inCode e iPlasma.
Reek	SI	
RevJava	SI	
Roodi	SI	
SA4J (IBM)	SI	
SemmlCode	NO	DESCARTADO: Herramienta ahora incluida en una nueva disponible sólo para empresas. Pedida y no facilitada.
Shimba	NO	DESCARTADO: Herramienta de la tesis “Reverse engineering environment”, autor Systä T.No facilitada.
Sissy	SI	
Sotograph	NO	DESCARTADO: Imposible conseguir la herramienta, pedido acceso pero denegado a versión de prueba.
STAN	SI	
Structure101	SI	
StyleCop	SI	
Together	SI	
TRex	SI	Herramienta TTCN3 del listado inicial.
XRay	SI	Añadida en sustitución de CodeCrawler
XRefactory	SI	

Tabla 9. Listado de herramientas tras la fase de búsqueda.

3.2. Análisis de herramientas

El análisis de cada una de las herramientas consta de tres partes bien diferenciadas, una de las cuales, la fase de Análisis, se divide en tres subsecciones que vienen marcadas por las tablas que se van a completar en esta memoria y que se definen con las subcaracterísticas de *Design Smell Management* (ver figura 4 y siguientes). Las otras dos partes son una explicación del propósito de la herramienta y un apartado con información sobre la herramienta, necesaria para saber sobre qué versión, dentro de las disponibles, se realiza el análisis y cómo y dónde debe instalarse.

- 
- a. **Propósito.**
 - b. **Información para el análisis.**
 - c. **Análisis.**
 - c.1. **Target Artefact.**
 - c.2. **Design Smells.**
 - c.3. **Activity.**

Antes de entrar en el análisis en profundidad de cada una de las herramientas, se considera necesario explicar varios métodos de instalación que son comunes a bastantes de las herramientas a analizar y a los que luego se hará referencia en el apartado dedicado a la instalación en el análisis de cada una de las herramientas que hagan uso de ellos.

Todas las herramientas que actúan como plugin de Eclipse han sido probadas en Eclipse 3.5. salvo en las que se indique lo contrario. Estas herramientas tienen dos posibles métodos de instalación (Algunas dispondrán de ambos y otras únicamente de uno de ellos):

- Vía Software Updates: Abrir Eclipse y dirigirse a Help→ Install new software → Add → Location. Insertar la url indicada y continuar hasta el final del proceso, tras lo cuál se reinicia eclipse. Cuando se abra de nuevo Eclipse la instalación estará completa.
- Descargando archivos desde la página de la herramienta y extrayéndolos en el directorio de instalación de Eclipse (Combinando las carpetas ya existentes con las que se copian) o bien en la carpeta plugins del mismo directorio de instalación si lo que se descarga es un archivo con extensión *.jar*. Otra opción de instalación de estos archivos es siguiendo los pasos del punto anterior (Vía Software Updates) hasta Add y después seleccionar Local update site y buscar los archivos que previamente se han extraído en la computadora.

La mayoría de las herramientas independientes también tienen dos métodos de instalación comunes:

- Si lo que se descarga es un ejecutable (Si es en *.rar*, *.zip* o similar, primero extraer el ejecutable donde se desee), lanzarlo y seguir las instrucciones que se muestren por pantalla.
- Si se descarga un archivo con extensión *.rar* o *.zip* o similar que contenga un directorio con la instalación ya realizada, extraer donde se desee.

Puede haber otros métodos de instalación que se detallan según aparezcan en las diferentes herramientas.

Analyst4j

3.2.1. Analyst4j

a. Propósito

Analyst4j [36] automatiza la medición de software y permite a los usuarios, con una única búsqueda, realizar un análisis del entorno basado en métricas.

Las métricas de software exponen diversos atributos de calidad del código, que son de gran utilidad para el desarrollo y mantenimiento de software. Analyst4j permite el uso eficaz de las métricas, proporcionando un servicio de búsqueda basado en métricas de software automatizadas. Esta búsqueda permite a los usuarios encontrar defectos que de otro modo sería muy complicado, dado el tamaño del software. Aunque hay búsquedas y análisis predefinidos que dan facilidades al usuario, es posible crear nuevas consultas de búsqueda y, posteriormente, un análisis personalizado basado en dichas consultas. Analyst4j proporciona una interfaz para crear estas búsquedas personalizadas y el posterior análisis.

Además Analyst4j genera informes en formato RTF (Formato compatible con la mayoría de procesadores de texto). Estos informes pueden editarse, permitiendo agregar nuevos valores.

b. Información para el análisis

Versiones disponibles:

- Analyst4j Standalone Version.
- Analyst4j - Eclipse Plugin.

Versión analizada: Analyst4j v.1.5.0 Standalone Version.

Sistema Operativo requerido:

- Como plugin de Eclipse [26], está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).
- Como herramienta independiente ha sido probado en Windows Vista, Windows 7 y sistemas Linux, aunque al ser una modificación de Eclipse, es lógico suponer que funcionará en los mismos Sistemas Operativos que éste.

Instalación:

- Como plugin de Eclipse con la url <http://www.codeswat.com/a4j/update/> o bien extrayendo en el directorio de instalación de Eclipse los archivos que se descargan de <http://www.codeswat.com/a4j/Analyst4j.zip>.
- Como herramienta independiente extrayendo donde se desee la carpeta que se descarga en <http://www.codeswat.com/a4j/Analyst4j-Standalone.zip>.

Código utilizado para el análisis:

- jfreechart 1.0.0.-pre1 [98] y jfreechart 1.0.1. (Este último se encuentra disponible tras la instalación de la herramienta independiente, se instala junto a ella como código de ejemplo)

c. Análisis

Se analiza la herramienta independiente en lugar del plugin de eclipse ya que no hay diferencia alguna entre ambos, con la salvedad de que la herramienta independiente incluye código de ejemplo en su instalación, lo cuál permite un análisis más rápido.

c.1. Target Artefact:

Como ya se ha visto en la información, la herramienta funciona tanto como plugin de Eclipse como de forma independiente.

En la guía de usuario que aparece en la página web de la herramienta [25], se puede leer que “Analyst4j tiene un poderoso motor independiente de análisis estático de código para Java, está diseñado para trabajar en cualquier fragmento de código fuente Java”, por lo que se deduce y posteriormente se comprueba en la propia herramienta que el formato del código que analiza es Código Fuente (Source Code).

La herramienta no admite versiones ya que aunque existe la posibilidad de comparar dos proyectos que se tengan cargados, el resultado es una gráfica que no ofrece ningún tipo de resultado específico en cuanto a comparación, se limita a unir los resultados que se obtienen con un análisis individualizado para cada opción de las disponibles dentro de Analyst4j Searches en una gráfica con ambos análisis, dejando al usuario como encargado de ver si se mejora o no respecto de una versión a otra. No permite tampoco cargar un repositorio de versiones para realizar la comparación. (Para ver la comparación: Pestaña Searches, con el botón secundario se selecciona una de las opciones de la lista y a continuación se hace clic en Across Analyzed Projects).

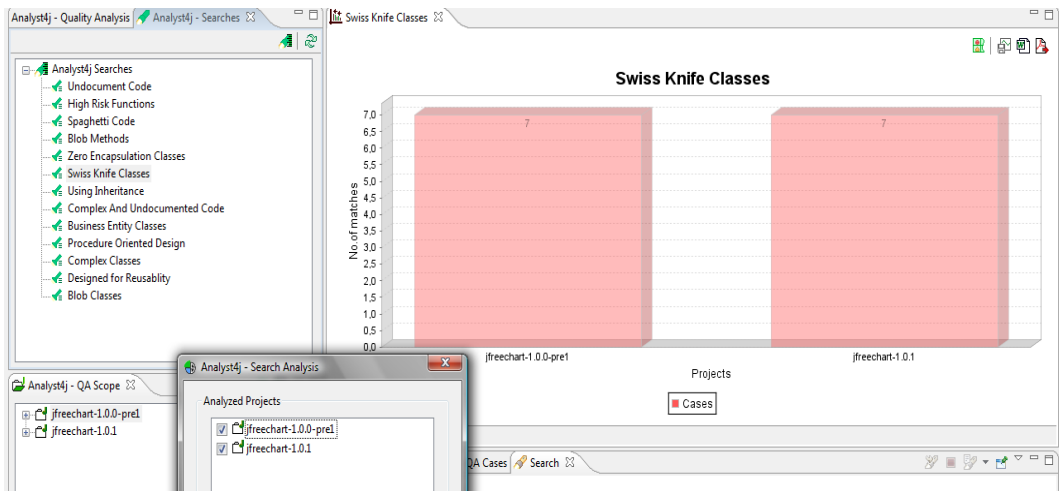


Figura 11. Analyst4j. Comparación manual entre dos versiones.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Analyst4j	Plugin Eclipse/Independiente	Source Code (JAVA)	NO	-

Tabla 10. Analyst4j. Tabla de la característica Target Artefact.

c.2. Design Smells:

En la página web se indican las siguientes características que podrían estar relacionadas con el análisis que se está efectuando acerca de la gestión de *Design Smells*:

- Encuentra Antipatterns rápidamente (*Blob classes*, *Spaghetti Code*, *Swiss Knife Classes*...). (Posible *Antipatterns*)
- Realiza un análisis y una interpretación de una serie de métricas de software automatizadas. (Posible *Metrics Warnings*)

Tras analizar el código fuente en la herramienta se ve que de los posibles *Type of Smell* que se están valorando en esta memoria (ver figura 7) existen:

***Metrics Warnings*:** Éste es un *Metrics Warnings* diferente de lo habitual ya que se indica un posible error mediante un gráfico de comparación de métricas y no mediante el valor de la métrica.

La comparación de cifras ayuda a visualizar un patrón y la tendencia de los datos y posteriormente el código que muestra estas cualidades. Por ejemplo, al comparar la Complejidad Medida de un Método (WMC2) con la profundidad de la herencia de una clase (DIT_CLS) se da una muestra de que incrementando la profundidad de la herencia, la complejidad disminuye, es decir, la complejidad de las clases está bien distribuida (ver figura 12). En un sistema donde hay muy poco uso de la herencia, la complejidad de las clases se concentra y lleva a la aparición de *Blob Classes* o *Swiss Knife Classes*.

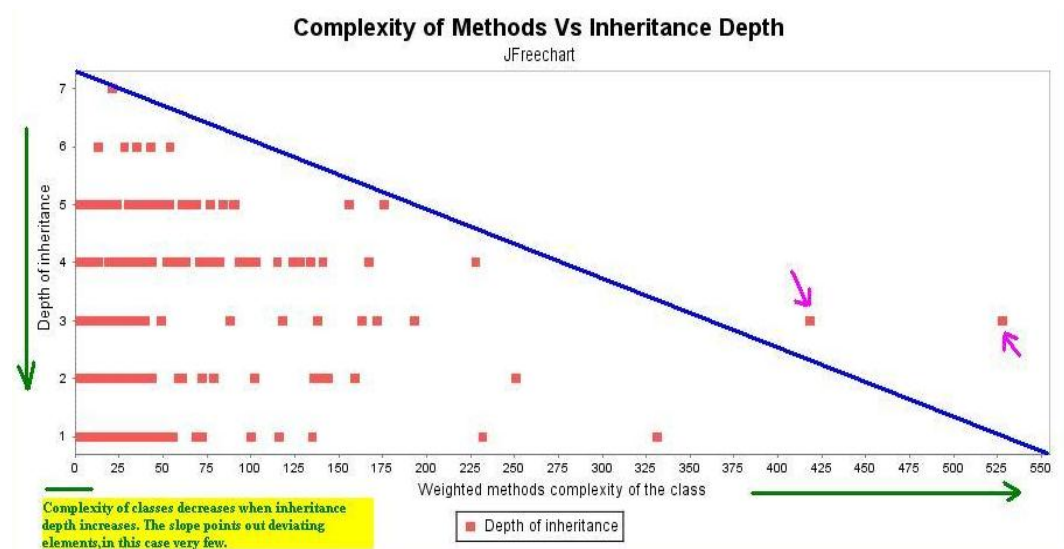


Figura 12. Analyst4j. Ejemplo de comparación de cifras.

***Antipatterns*:** La herramienta se centra en la búsqueda de *Development Antipatterns* (Antipatrones de desarrollo) que afectan principalmente a los desarrolladores de software. Describen situaciones encontradas por el programador cuando resuelve problemas de programación. Para identificar *Antipatterns* en Java, Analyst4j proporciona una búsqueda basada en el cálculo de métricas.

La aplicación de Antipatrones para el control y el mantenimiento, se inicia con la identificación de dichos elementos. Dado el volumen de código, es relativamente difícil encontrar estos elementos de manera manual, por lo que Analyst4j propone un método para identificarlos mediante métricas. Para ello, establece un determinado umbral en las métricas usadas para identificar los síntomas de un determinado antipatrón. Así, Analyst4j identifica tres tipos de antipatrones: *The Blob*, *Spaghetti Code* y *Swiss Knife Classes* [5]. Los umbrales utilizados para los tres antipatrones citados anteriormente se ven en *figura 13*, *figura 14* y *figura 15* respectivamente.

Respecto a la Granularidad (*Granularity*), la herramienta sólo tiene implementadas búsquedas a nivel de clase y de método pero es posible definir operaciones hasta a nivel de paquete.

<i>Metric</i>	<i>Description</i>	<i>Threshold</i>
<i>Coupling between objects (CBO_CLS)</i>	<i>Association with other classes.</i>	<i>>=30</i>
<i>Weighted Methods per Class(WMC1_CLS)</i>	<i>Number of methods of a class</i>	<i>>=60</i>
<i>Lack of Cohesive Methods (LCOM_CLS)</i>	<i>LCOM range[0 to 2] and value nearing one is considered to be alarming level of lack of cohesion.</i>	<i>>=0.8</i>

Figura 13. Analyst4j. Umbral de métricas usado para identificar *The Blob*.

<i>Metric</i>	<i>Description</i>	<i>Threshold</i>
<i>Essential Complexity (EC_MTD)</i>	<i>Measures unstructuredness of code.</i>	<i>>1</i>

Figura 14. Analyst4j. Umbral de métricas usado para identificar *Spaghetti Code*.

<i>Metrics</i>	<i>Description</i>	<i>Threshold</i>
<i>Number of Children(NOC_CLS)</i>	<i>No reuse</i>	<i>=0</i>
<i>Lack of Cohesive Methods(LCOM_CLS)</i>	<i>High unrelated data and methods</i>	<i>>=0.8</i>
<i>Weighted Method Complexity(WMC2_CLS)</i>	<i>Too many complex methods</i>	<i>>=120</i>

Figura 15. Analyst4j. Umbral de métricas usado para identificar *Swiss Knife Classes*.

El análisis que la herramienta lleva a cabo mediante el cálculo de métricas, bien sea para detectar *Metrics Warnings* o bien para detectar antipatrones, tiene relación tanto de tipo inter como intra, provocadas, por ejemplo, en el caso de intra por el antipatrón *Spaghetti Code* y en el caso de inter por algún *Metrics Warnings*.

En cuanto a la casilla de la herencia, se marca con un SI debido a que la herramienta realiza el cálculo de métricas como *Deph Inheritance*, la cual además se usa para mostrar algunos de los *Metrics Warnings*.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Analyst4j	Antipatterns Metrics Warning	NO	NO	SI	SI	SI	SI	SI	SI

Tabla 11. Analyst4j. Tabla de la característica Design Smell.

c.3. Activity:

Una vez realizado el análisis del código con la herramienta, el siguiente paso es ver qué características tiene el programa de las señaladas en la *figura 7* y sus subcaracterísticas representadas en la propia *figura 7* y en la *figura 8* y la *figura 9*.

La opción de Especificación se puede encontrar en la pestaña Quality Analysis. Para definir una nueva búsqueda, se selecciona con el botón secundario del ratón Custom Analysis y después Define Analysis en el menú emergente. Aparecerá una nueva ventana con las opciones posibles ya definidas o con las que haya definido anteriormente el usuario. Para esto último, se incluye la opción Analyst4j Search que aparece en la pestaña Analyst4j del menú superior y en la que se pueden definir nuevas búsquedas, cargar otras que se tengan guardadas o modificar las ya existentes en la herramienta.

Como ya se ha comentado, la herramienta realiza la detección de *Design Smells* basándose en el cálculo de métricas. Los resultados los muestra mediante gráficos (ver *figura 16*).

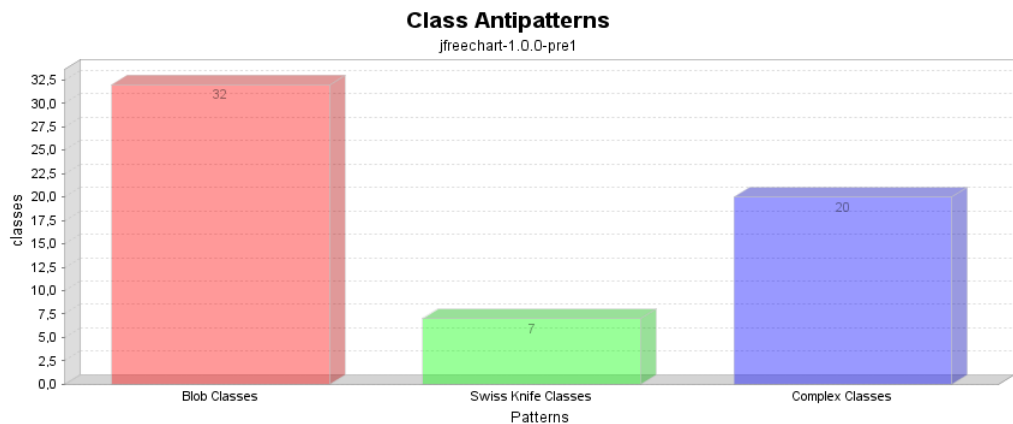


Figura 16. Analyst4j. Actividad Visualización en la que se muestra el resultado de un análisis.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Analyst4j	Especificación	Process description	Manual	Design Smell Specification	Textual
	Detección	Metrics-Based (Structural Metrics)	Interactive	Entity-metrics relationships	Numerical, textual
	Visualización	Diagrams	Fully-automated	Design Smell visualisation	Visual

Tabla 12. Analyst4j. Tabla de la característica Activity.

ArgoUML

3.2.2. ArgoUML

a. Propósito

ArgoUML [17] es una aplicación de diagramado de UML escrita en Java y publicada bajo Licencia BSD. “*Proporciona soporte cognitivo de diseño orientado a objetos y algunas de las características de automatización de una herramienta CASE comercial, pero esta enfocado en características que soportan las necesidades cognitivas de los diseñadores*” [91].

b. Información para el análisis

Versiones disponibles:

- En la web de desarrollo de la herramienta (<http://argouml-downloads.tigris.org/>) se encuentra un repositorio donde se pueden encontrar versiones de la herramienta desde la versión 0.10.1 hasta la última versión disponible a la hora de realizar esta memoria, ArgoUML 0.30.2, datada en Julio de 2010.

Versión analizada: ArgoUML 0.28.1 y ArgoUML 0.30.

Sistema Operativo requerido:

- ArgoUML es una herramienta que puede ser instalada bajo plataforma Windows, Linux o Mac OS X. También, dado que está escrita en Java, puede ser ejecutada sin necesidad de tenerla instalada en el disco duro gracias a la herramienta Java Web Start.

Instalación:

- ArgoUML es una herramienta independiente de licencia gratuita que se puede descargar de la página oficial. Para su instalación, se encuentran a disposición del usuario varias opciones; si se trabaja en Windows existe un archivo ejecutable que una vez descargado y lanzado procederá con una instalación típica en este sistema operativo (Esta opción es la que se ha seguido para el análisis). También existe la posibilidad de descargar los archivos binarios para su instalación en Linux y Mac OS X.

Código utilizado para el análisis:

- Lucene3 [97].

c. Análisis

Antes de comenzar a enumerar las características de la herramienta, es conveniente comentar que ArgoUML ofrece una gran diversidad en cuanto a número de operaciones, como se puede observar en el Manual de ArgoUML [91]. No obstante, en este documento sólo se reflejan las características referentes a la detección de *smells*, y es probable que no queden reflejadas en su totalidad, debido a que esta herramienta ofrece una detección de errores para formatos UML o similares y que no se han podido probar ya que no se dispone de diagramas UML de un tamaño suficiente para hacer un análisis útil. Para el análisis se ha optado por la importación de código fuente ya que esta es otra de las opciones disponibles en ArgoUML. Dicha importación intenta transformar el código en modelos de diagramas de clase, de uso, etc. pero esta característica no está

muy desarrollada en la herramienta ya que no genera los diagramas completos, perdiéndose información, por ejemplo, acerca de las relaciones entre los elementos.

c.1. Target Artefact:

ArgoUML es una herramienta independiente que es capaz de trabajar con modelado UML, Zargo, XMI, XML y también con código fuente escrito en Java, C y C++. No admite versiones, puesto que no se pueden comparar los cambios efectuados entre dos proyectos.

Se desconoce el tipo de representación interna del objeto de entrada que realiza ArgoUML, pero es lógico pensar que se realiza de algún modo gráfico por la cantidad de representaciones de modelos que se hacen y a que parece que los análisis que realiza en busca de defectos de diseño se basan en el modelado previamente realizado del código importado. Pese a esto, no hay información que permita verificarlo por lo que no se indica en su apartado correspondiente.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
ArgoUML	Independiente	Source Code (JAVA, C, C++) Model (UML, Zargo, XMI, XML)	NO	-

Tabla 13. ArgoUML. Tabla de la característica Target Artefact.

c.2. Design Smells:

A esta herramienta, en el análisis a partir de importación de código, sólo se le reconoce un tipo de error, el *bad coding style*, aunque no se pueden descartar otras posibilidades en un análisis que se efectuase a partir de modelos UML completos.

Una vez importado el código fuente, ArgoUML representará dicho código en formato de modelo de diagramas para de esta forma llevar a cabo su análisis. Los errores detectados reflejarán decisiones erróneas respecto a estándares o convenciones de código referentes al correcto modelado de los diagramas.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
ArgoUML	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO

Tabla 14. ArgoUML. Tabla de la característica Design Smells.

c.3. Activity:

ArgoUML posee dos tipos de actividades fácilmente diferenciables para el tratamiento de *smells*. Una de ellas es la detección, que se realiza de forma totalmente automática una vez importado el código fuente. Como se puede observar en la *figura 17*, estos errores aparecen marcados por defecto por orden de prioridad (alta, media y baja). A su lado se mostrará el número total de errores detectados.

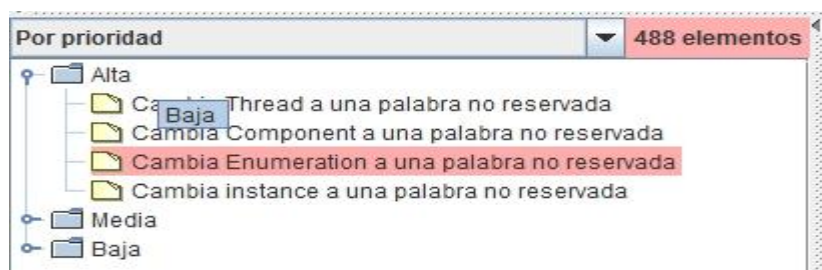


Figura 17. ArgoUML. Elementos problemáticos detectados tras un análisis con código java importado.

La otra actividad encontrada es la de corrección. Esta actividad aparece tras ejecutar la detección. Cuando el usuario hace clic en uno de los errores marcados, en el cuadro de la parte inferior derecha se muestra información acerca del mismo, incluyendo también alguna sugerencia para la corrección, la cual es decisión del usuario si debe realizarse o no (ver *figura 18*).

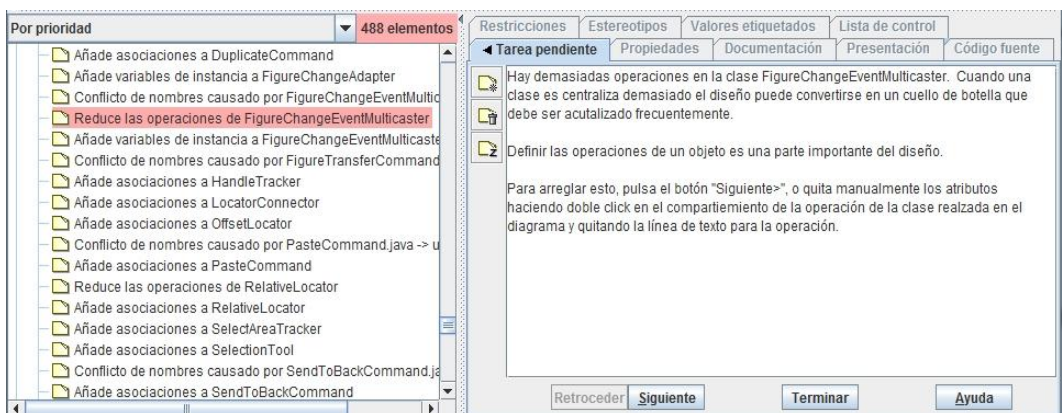


Figura 18. ArgoUML. Sugerencias de corrección mostradas para un determinado problema.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
ArgoUML	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Interactive	Correction Suggestions	Textual

Tabla 15. ArgoUML. Tabla de la característica Activity.

Argus CodeWatch

3.2.3. Argus CodeWatch

a. Propósito

Argus CodeWatch [49] funciona como plugin integrado en Eclipse. El propósito de este plugin es añadir errores y advertencias adicionales a la infraestructura Eclipse, asistiendo a los programadores Java para crear código libre de errores. Problemas como errores de sintaxis son detectados por el compilador Java, y por lo tanto, no es necesario controlarlos con Argus CodeWatch. El plugin intenta detectar las malas prácticas de codificación o violaciones de contrato. Proporciona, además, un marco para añadir nuevos controles que los programadores consideren necesarios o para añadir a los controles ya existentes, los denominados *quickfixes* (soluciones rápidas).

b. Información para el análisis

Versiones disponibles:

- Argus CodeWatch Eclipse plugin

Versión analizada: Argus CodeWatch Eclipse plugin 0.9.0.

Sistema Operativo requerido:

- Como plugin de Eclipse, está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).

Instalación:

- Como plugin de Eclipse con la url <http://arguscodewatch.sourceforge.net/update/> o bien extrayendo en el directorio de instalación de Eclipse los archivos que se descargan de <http://sourceforge.net/projects/arguscodewatch/files/>

Código utilizado para el análisis:

- jfreechart y jHotdraw 5.2. [99]

c. Análisis

c.1. Target Artefact:

Argus CodeWatch funciona, como ya se ha mencionado anteriormente, únicamente como plugin de Eclipse. Este plugin, escrito en Java 5.0, analiza código fuente escrito en java.

La herramienta no admite versiones, ya que no es posible realizar un análisis partiendo de dos proyectos iguales pero que hayan sufrido alguna corrección o modificación entre ellos y se desee observar el progreso efectuado en el código.

En cuanto a su tipo de representación, se desconoce al no tener ningún tipo de información al respecto pero al estar insertado como plugin de eclipse, la herramienta podría no realizar una representación interna propia, sino que se serviría de la que hace Eclipse.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Argus CodeWatch	Plugin Eclipse	Source Code (JAVA)	NO	-

Tabla 16. Argus CodeWatch. Tabla de la característica Target Artefact.

c.2. Design Smells:

La herramienta se centra en detectar únicamente un tipo de *smell*, *Bad Coding Style*. Este tipo de *smell*, como ya se ha comentado en puntos anteriores, se produce cuando se cometen errores o violaciones de las convenciones y/o estándares de código establecidos. Argus CodeWatch se encarga de marcar las posibles violaciones de código de forma que se puedan visualizar en su punto exacto.

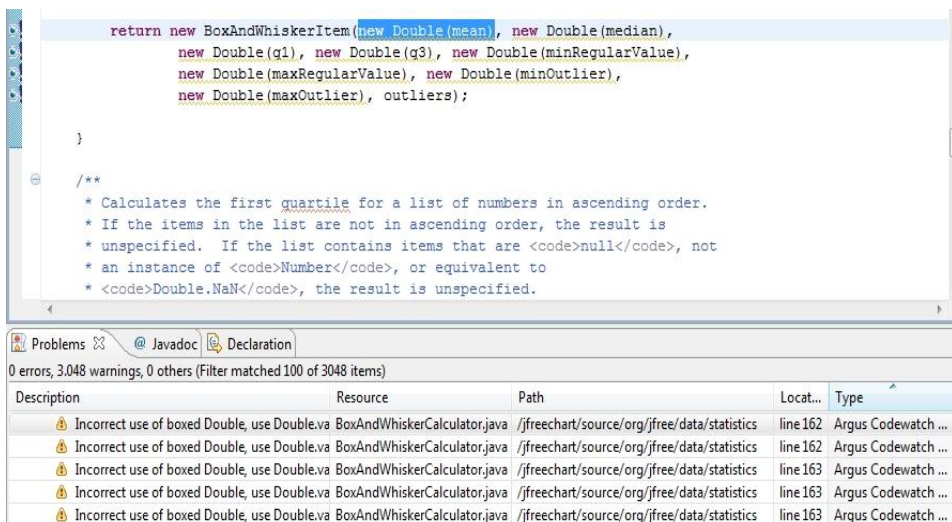


Figura 19. Argus CodeWatch. Detección de errores.

Un aspecto que es necesario comentar de la herramienta es la posibilidad de poder cambiar el tratamiento del *bad coding style*, realizando modificaciones dentro de las preferencias. Estas modificaciones se basan en el carácter que se le quiera dar a posibles violaciones dentro del siguiente rango: error, warning o ignore. También se pueden marcar y desmarcar las diferentes casillas de errores como se aprecia en la figura 19.

Con respecto a la granularidad, al tratarse de una herramienta que solamente trata el *bad coding style*, se puede hablar de actividad a nivel de método y clase, debido principalmente a que se centra en errores de violaciones en código.

No existen relaciones ni de tipo inter ni de tipo intra, debido a que las violaciones se producen en un determinado punto del código y no se relacionan con otros posibles errores de igual o diferente nivel. No trata en ningún caso actividades que tengan relación con la herencia.

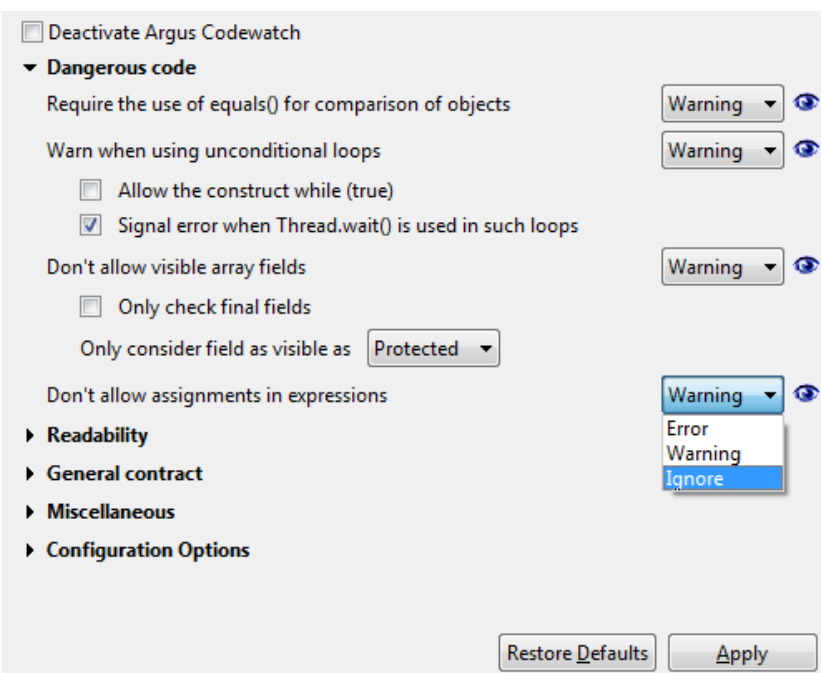


Figura 20. Argus CodeWatch. Preferencias Argus CodeWatch.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Argus CodeWatch	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO

Tabla 17. Argus CodeWatch. Tabla de la característica Design Smells.

c.3. Activity:

Argus CodeWatch posee la actividad de Detección ya que muestra, como se ha visto en el punto anterior, errores de tipo *bad coding style* a lo largo del código analizado, marcando los puntos dónde se encuentra la violación. En la zona inferior, dentro de una ventana denominada *problems*, se puede encontrar una breve descripción del problema (ver *figura 19* y *figura 21*). La técnica para su detección está basada en reglas que vienen dadas por estándares de código.

La segunda actividad que presenta esta herramienta es la de Corrección, lo hace mediante unas sugerencias que aparecen en determinadas violaciones en el código y que se muestran en la *figura 21*. Existen algunos errores para los cuales Argus CodeWatch no es capaz de sugerir soluciones.

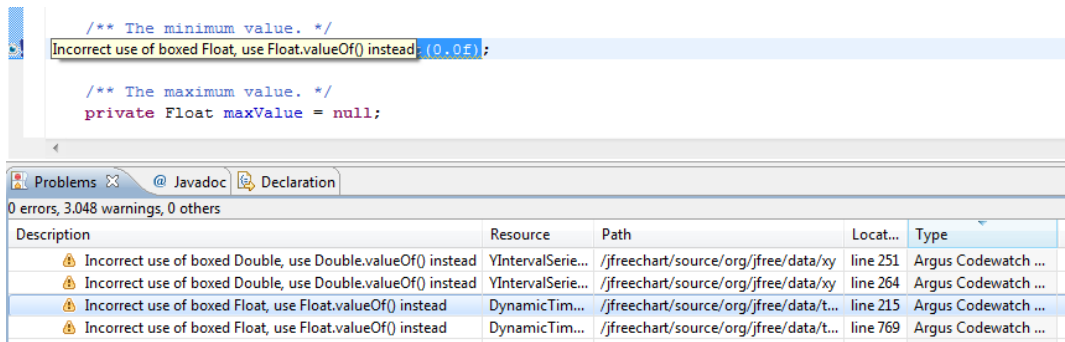


Figura 21. Argus CodeWatch. Sugerecias de corrección.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Argus CodeWatch	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Manual	Correction Suggestions	Textual

Tabla 18. Argus CodeWatch. Tabla de la característica Activity.

CodePro Analytix

3.2.4. CodePro Analytix

a. Propósito

CodePro Analytix [50] es una herramienta de prueba de software Java para desarrolladores de Eclipse. Permite trabajar en la mejora de la calidad y la seguridad del código. Se integra en el entorno de Eclipse, usando un análisis automatizado de código fuente para detectar problemas de calidad y vulnerabilidades de seguridad antes de que el código alcance la fase de control de calidad, o peor aún, la de producción. CodePro Analytix ofrece, además, recomendaciones concretas para arreglar la mayoría de las situaciones problemáticas detectadas.

b. Información para el análisis

Versiones disponibles:

- CodePro Analytix 6.5. - Eclipse Plugin.

Versión analizada: CodePro Analytix 6.5. - Eclipse Plugin.

Sistema Operativo requerido:

- Como plugin de Eclipse [26], está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).

Instalación:

- Como plugin de Eclipse con la url:

<http://download.instantiations.com/CodeProAnalytix/integration/latest/update/3.5/>

o bien usando el auto instalador disponible en

<http://www.instantiations.com/codepro/analytix/download-latest.html>

y que es válido para las versiones de Eclipse que se encuentran entre la 2.1 y la 3.3.

Código utilizado para el análisis:

- Jhotdraw5.2. y proyecto09 [90].

c. Análisis

La versión analizada de la herramienta es una versión de prueba válida durante 30 días desde su activación aunque ésta es totalmente funcional.

c.1. Target Artefact:

Como se ha podido ver en la descripción de la herramienta y como se ve en el sitio web oficial, CodePro Analytix analiza código Java cargado en Eclipse. La herramienta no admite la opción de comparar dos versiones de un mismo código ni cargando ambas en Eclipse ni a través de un repositorio de versiones. Se desconoce el tipo de representación interna que la herramienta realiza del artefacto objetivo.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
CodePro Analytix	Plugin Eclipse	Source Code (JAVA)	NO	-

Tabla 19. CodePro Analytix. Tabla de la característica Target Artefact.

c.2. Design Smells:

En las preferencias de la herramienta (Pestaña Window → Preferences → CodePro dentro de Eclipse) se encuentran multitud de opciones que pueden indicar el tipo de *Design Smell* que CodePro detecta. En la opción Audit aparecen muchas características relacionadas, entre otras, con el smell denominado *Bad Coding Style*, como puede ser *The elements of Java Style*, que aparece en el desplegable Audit Rule Set. Se observa que otra de las opciones es Metrics por lo que se procede a cargar código y analizarlo para verificar que se trata de *Metrics Warnings* y no simplemente de una relación de métricas que el programa calcula para que el usuario las utilice como crea conveniente.

CodePro Analytix permite localizar *smells* relacionados con la herencia tanto en su apartado Audit, con varias opciones relacionadas con la herencia (Casilla Inheritance) como en su apartado Metrics, donde calcula la métrica profundidad de la herencia entre otras.

Algunas métricas se calculan desde nivel de método hasta nivel de paquete, además de los errores o violaciones de convenios de código (*Bad coding style*) que se pueden encontrar en clases o métodos.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
CodePro Analytix	Metrics Warning	NO	NO	SI	SI	SI	SI	SI	SI
	Bad Coding Style								

Tabla 20. CodePro Analytix. Tabla de la característica Design Smells.

c.3. Activity:

De las características señaladas en la *figura 8* y siguientes de la taxonomía, aparece la especificación ya que la herramienta permite al usuario hacer sus propias búsquedas de *smells* señalando nuevas opciones o desmarcando otras en un listado ya facilitado. Otra característica de esta herramienta dentro de las opciones de especificación es que permite, a partir de una serie de reglas ya determinadas en una serie de libros [51], crear búsquedas más restrictivas mediante la unión de estos (ver *figura 22*).

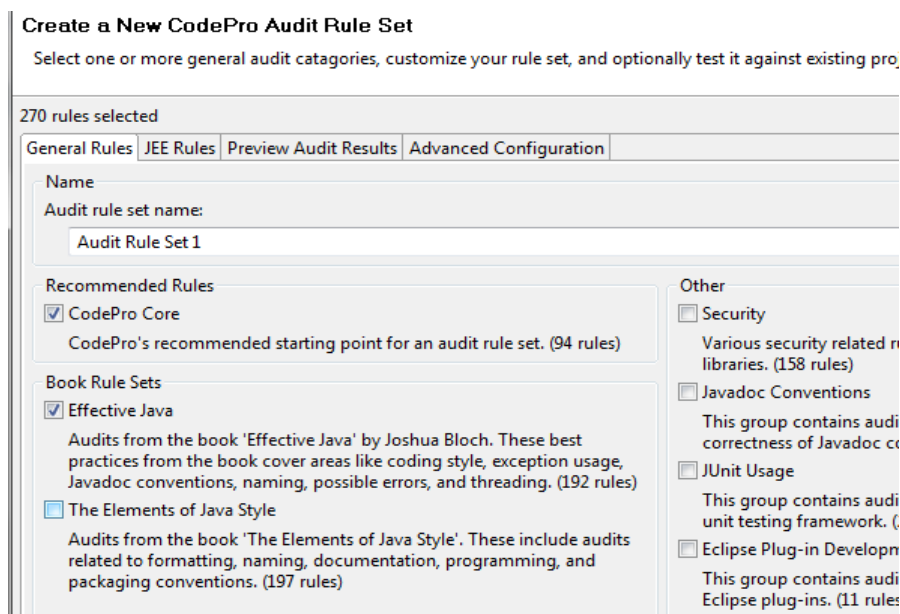


Figura 22. CodePro Analytix. Creación de un nuevo conjunto de reglas a partir de libros.

Otra característica a señalar además de la de detección ya comentada en el anterior apartado, es la de corrección ya que la herramienta sugiere posibles correcciones a determinados errores. Además, se detecta una posible actividad de visualización de los problemas detectados. (No es posible verificar al completo esta característica y adjuntar una figura en la que se vea claramente dicha característica.¹)

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
CodePro Analytix	Especificación	Book rule sets, Process Description	Interactive	Design Smell Specification	Textual
	Detección	Dependency analysis, metric-based	Interactive	Entity-metrics y entity-smells relationships	Numerical, textual
	Corrección	Rules-based	Interactive	Correction suggestions	Textual
	Visualización ¹	Dependency analysis	Interactive	Package dependencies and cycles	Visual

Tabla 21. CodePro Analytix. Tabla de la característica Activity.

¹ En un primer análisis, se detecta una posible actividad de visualización ya que el programa muestra dependencias de manera gráfica pero tras el análisis de otras herramientas, existe la posibilidad de que dichas gráficas no muestren en realidad ningún error por si solas sino que simplemente sean de carácter informativo para que sea el usuario quien determine, viendo estos gráficos, si hay algún tipo de defecto de diseño en el código o no. Debido a que la versión de prueba de que se disponía ha caducado y a que Instantiations, la empresa encargada de CodePro, ha sido adquirida recientemente por Google, la herramienta ha dejado de estar disponible a la hora de realizar un segundo análisis y ésta memoria, por lo que no se ha podido verificar si los gráficos pueden indicar una actividad de Visualización o no.

Cultivate

3.2.5. Cultivate

a. Propósito

Cultivate [9] es un plugin de Eclipse, que sirve de guía, en general, para mejorar el diseño de los proyectos Java. Permite obtener un código mantenible, reutilizable y comprensible mientras que éste se va creando.

Cultivate evalúa y visualiza código detectando *bad smells* y métricas de software. Los resultados son mostrados como una representación textual a través de una variedad de diagramas, centrándose en aspectos de diseño específicos. Durante el desarrollo, las violaciones en la estructura impuestas por una determinada arquitectura a seguir, se indican de una forma característica. También muestra y analiza los conceptos de software subyacente, representados en los nombres de los identificadores.

b. Información para el análisis

Versiones disponibles:

- Cultivate Eclipse Plugin

Versión analizada: Cultivate Eclipse plugin

Sistema Operativo requerido:

Como plugin de Eclipse, está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).

Instalación:

Para instalar Cultivate se necesita instalar *swi-prolog* [52]. Para que funcione correctamente la herramienta es recomendable la versión 5.8.3. Para completar esta instalación hay que añadir la carpeta **bin** del directorio de instalación de *swi-prolog* a la variable **PATH** del sistema si es que el instalador de *prolog* no lo ha hecho. Para añadir la carpeta a la variable PATH (en Windows) se seguirán las siguientes operaciones:

Inicio → clic con botón secundario en Equipo → Propiedades → Configuración avanzada del sistema → Opciones avanzadas → Variables de entorno → Variables de sistema → Seleccionar la variable PATH y clic en editar. A continuación escribir la ruta de la carpeta bin de SWI-PROLOG tras la última ruta existente en la variable PATH, separada por punto y coma (;) y finalizar haciendo clic en Aceptar

Antes de instalar Cultivate, es necesaria la herramienta jTransformer [53], que se empleará para adaptar la estructura interna del proyecto java a analizar al uso de Cultivate. Para instalar jTransformer seguiremos los siguientes pasos:

Iniciar eclipse y seleccionar: Help → Install new software → Add → Location e insertar la url: <http://sewiki.iai.uni-bonn.de/public-downloads/update-site/>. Continuar hasta finalizar correctamente la instalación y reiniciar eclipse, para posteriormente acometer la instalación de Cultivate.

El plugin de Eclipse se instalará como viene siendo habitual en el resto de herramientas en formato plugin para Eclipse analizadas con anterioridad. La url necesaria para su instalación es <http://sewiki.iai.uni-bonn.de/public-downloads/cultivate-site/>.

NOTAS: GEF, Draw2D y Zest serán instaladas con Cultivate en Eclipse si no habían sido instaladas con anterioridad. Estas herramientas complementarias serán de utilidad para Cultivate para proporcionar visionados gráficos.

Para utilizar la característica “Smell Detection in Context Feature” también incluida en la herramienta Cultivate es necesario instalar en Eclipse el plugin Mylyn [54] que se encuentra disponible en la sección de descargas de la web oficial de Eclipse [25].

Código utilizado para el análisis:

- jhotDraw 5.2.

c. Análisis

Se analiza la versión 1.0.0 del plugin para eclipse, que es la primera versión disponible. Se encuentra en la red desde marzo de 2010.

c.1. Target Artefact:

Cultivate funciona únicamente como plugin de Eclipse. Requiere para su correcto funcionamiento jTransformer, que a su vez requiere Prolog. La herramienta ha sido diseñada para analizar código fuente Java.

Cultivate no admite versiones, ya que no es posible realizar un análisis partiendo de dos proyectos iguales pero que hayan sufrido alguna corrección o modificación entre ellos y se quiera observar el progreso del código.

Su tipo de representación viene marcado por la utilización del plugin jTransformer, del que se ha detallado su instalación anteriormente. Gracias a esta herramienta, Cultivate puede llevar a cabo su análisis del proyecto con el que se esté trabajando, ya que se encarga de hacer la transformación interna creando un AST (*árbol sintáctico abstracto*) usando fórmulas lógicas, para posteriormente realizar dicho análisis.

Para analizar un proyecto Java cargado, el primer paso debe ser asignar jTransformer nature al proyecto Java con el que se vaya a trabajar. Para ello, hay que hacer clic con el botón derecho sobre el proyecto y desplazarse hasta *configure*, una vez allí, seleccionar *Assign JTransformer Factbase* en el menú contextual que aparecerá.

Después de esta asignación, se hará algo similar para hacer funcionar Cultivate (ver *figura 23*). En el submenú *configure* se debe seleccionar la opción *convert to cultivate Project*, que asignará las propiedades necesarias para llevar a cabo el análisis del proyecto.



Figura 23. Cultivate. Conversión a proyecto Cultivate.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Cultivate	Plugin Eclipse	Source Code (JAVA)	NO	AST

Tabla 22. Cultivate. Tabla de la característica Target Artefact.

c.2. Design Smells:

Cultivate es una herramienta bastante completa en cuanto a detección de *smells* se refiere, ya que abarca muchos de ellos. En particular hace un análisis de *bad smells* y *disharmonies*, realiza una tarea de detección de *bad coding style* y proporciona unos gráficos de dependencias donde se puede observar el grado de unión que tienen determinadas entidades.

Para comenzar el análisis una vez realizada la transformación a Cultivate project se debe activar la perspectiva Cultivate dentro de Eclipse, para ello se debe seguir la ruta window → open perspective → other → Cultivate. Una vez abierta la perspectiva, se podrán variar las vistas que se deseen de dicha perspectiva, para poder visualizar los diferentes análisis que ofrece la herramienta.

Una de las vistas más interesantes para el análisis de *bad smells*, *disharmonies* y *bad coding style* es *Metric and Smells Results* (En anteriores versiones se denominaba *Metric and Detector Results*, ver figura 24). En esta vista se ven resaltados como errores o warnings aquellos defectos o violaciones que han sido marcados previamente para su visualización. La manera de categorizar estos *smells* se realiza mediante un marcador que ofrece Cultivate. Para llegar a él, hay que hacer clic con el botón derecho en el proyecto que se desee y desplazarse hasta propiedades. Una vez allí seleccionar *Cultivate Markers*.

Como se puede observar en la figura 25, algunos de los defectos más reconocidos por varios autores se encuentran disponibles para detectar con esta herramienta, como pueden ser: *God Class*, *Brain Class*, *Data Class*, *Middleman*, *Iceberg Class*, *Tradionbreaker*, *Refused Parent Bequest*, *Brain Method*, *Feature Envy*, *Shotgun Surgery*, *Dispersed Coupling*, *Intensive Coupling*, *Method Chain*, *Law of Demeter Violation*, etc.

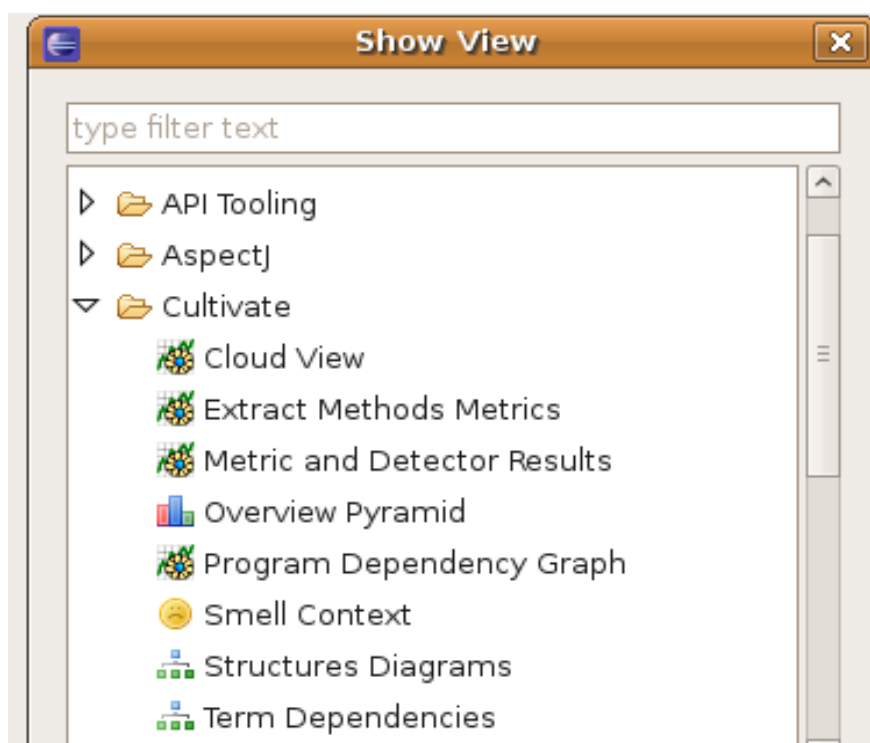


Figura 24. Cultivate. Vistas que incluye la herramienta.

Una vez marcados los *smells* de la forma que el usuario haya elegido para su análisis, se volverá de nuevo a la perspectiva Cultivate para continuar con el estudio.

En cuanto a la granularidad de la herramienta, el análisis llega hasta un nivel de paquete, debido a que la detección se puede iniciar desde el paquete e ir continuando con el análisis de forma descendente hasta nivel de método.

Debido a la detección de *smells*, existen relaciones inter, como sucede en el caso de encontrar *God Class*, donde aparece una relación de tipo inter debido a la posible relación con diferentes tipos de entidades y relaciones de tipo intra, como sucede con *Data Class*, donde la relación intra viene provocada por la relación entre elementos de una misma entidad.

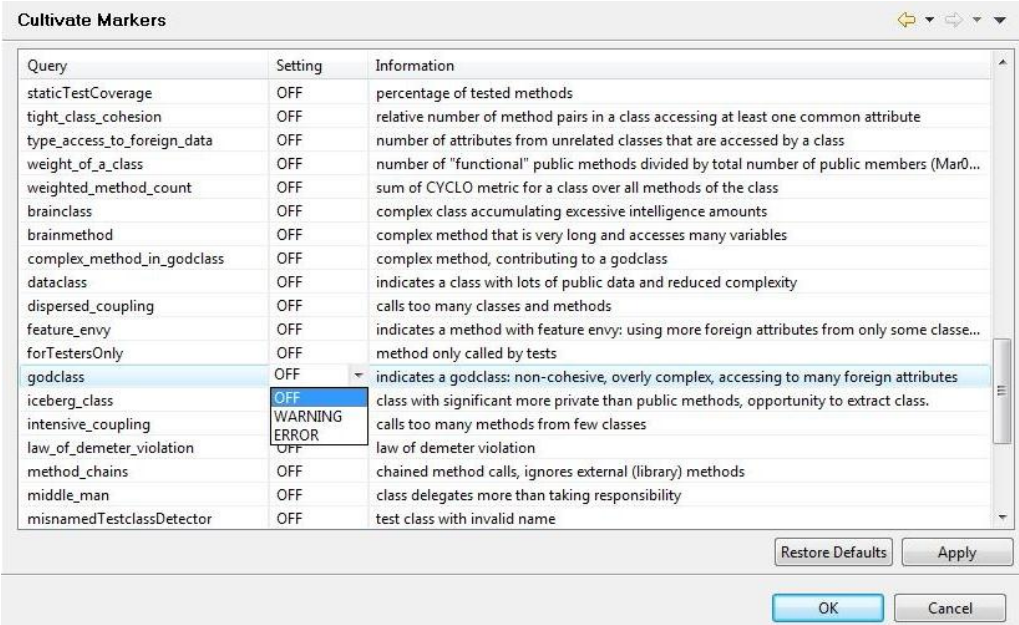


Figura 25. Cultivate. Selección del modo de marcar el error en Cultivate.

Cultivate hace al menos un tratamiento de la herencia ya que es capaz de detectar el *bad smell* *Refused Parent Request* [2], el cual está basado en herencia.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Cultivate	Bad Smells Disharmonies Bad Coding Style Dependency Warnings	NO	NO	SI	SI	SI	SI	SI	SI

Tabla 23 Cultivate. Tabla de la característica Design Smells.

c.3 Activity:

Uno de los tipos de actividad que posee Cultivate es el análisis, aunque realmente no es él quien lo realiza. Como ya se ha comentado anteriormente Cultivate se beneficia del trabajo de otras herramientas para su funcionamiento. En este caso jTransformer realiza el trabajo para la extracción del modelo, creando un AST para que luego Cultivate pueda llevar a cabo el análisis.

La detección se realiza basándose en las métricas obtenidas. Esta actividad se lleva a cabo de forma interactiva, ya que requiere de la acción del usuario para seleccionar los *smells* que queramos detectar.

La última actividad característica de esta herramienta es la visualización. Cultivate proporciona la posibilidad de ofrecer gráficos para el análisis del proyecto con el que se este trabajando. Destaca el gráfico de dependencias entre entidades, que muestra las violaciones de dependencia en color rojo. Además existen dentro de las vistas otros tipos de visualización, como se puede comprobar observando la *figura 25*.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Cultivate	Análisis (jTransformer)	Model Extraction	Fully-Automated	-	-
	Detección	Metric-Based, Dependency Analysis	Interactive	Entity-smells relationships	Textual, Numerical
	Visualización	Dependency Analysis	Interactive	Package dependency graph	Visual

Tabla 24. Cultivate. Tabla de la característica Activity.

Eclipse

3.2.6. Eclipse

a. Propósito

Eclipse [25] es un entorno de desarrollo integrado, de código abierto y multiplataforma, creado para desarrollar lo que el grupo de proyecto encargado de la herramienta llama "Aplicaciones de Cliente Enriquecido", que se encuentran en el lado opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma ha sido usada típicamente para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) o el compilador (ECJ) que se entrega como parte de Eclipse y que son usados para desarrollar el propio Eclipse.

Eclipse dispone de un editor de texto con resaltado de sintaxis. La compilación se realiza en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (*wizards*) para creación de proyectos, clases, tests, etc., y refactorización. Asimismo, a través de *plugins* libremente disponibles es posible añadir diferentes controles o herramientas adicionales para proporcionar funcionalidades al frente de la plataforma cliente, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite el uso de técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de "meta data" en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

b. Información para el análisis

Versiones disponibles:

- En la web de la herramienta se encuentra disponible, nada más acceder al sitio, la versión más reciente de Eclipse (Eclipse 3.6) pero también se ofrece dentro del sitio web la opción de descargar cada una de las versiones anteriores. Como es un entorno de trabajo muy utilizado en la realización de este proyecto, se han utilizado varias versiones antiguas para la comprobación de algunos *plugins* de detección de *smells*, pero para la realización de su análisis individual se ha utilizado la última versión a fecha de la realización de esta parte de la memoria.

Versión analizada: Eclipse Classic 3.6.0

Sistema Operativo requerido:

Eclipse es un entorno de trabajo multiplataforma. Puede ser instalado en sistemas operativos como Windows, Linux y Mac OS X.

Instalación:

Para su instalación es necesario haber descargado de su web oficial, <http://www.eclipse.org>, el paquete que contiene la herramienta. Una vez descargada se extraerá todo el contenido del archivo comprimido a una carpeta vacía. En el interior de la

carpeta donde se descomprime, se encontrarán todos los archivos que componen la herramienta, uno de ellos es un archivo ejecutable que lanzará Eclipse.

Una vez iniciada la ejecución y antes de que se complete totalmente, Eclipse pedirá que se seleccione una carpeta que por defecto la denomina *workspace*, donde se guardará el código de los proyectos que se efectúen con él.

Código utilizado para el análisis:

- jhotDraw5.2.

c. Análisis

Se analiza la versión de Eclipse Classic 3.6.0 que es la última disponible hasta la fecha de realización de esta memoria.

c.1. Target Artefact:

Eclipse funciona como un entorno de desarrollo integrado de manera independiente, sin la necesidad de otra herramienta que condicione su trabajo como ocurre con muchas otras de las herramientas analizadas en este trabajo.

En Eclipse se puede desarrollar y analizar código en diferentes lenguajes de programación como pueden ser Java, C, C++ y Python. La herramienta también permite trabajar con lenguajes para procesamiento de texto como LaTeX, aplicaciones en red como Telnet y un sistema de gestión de base de datos. Estas últimas no atañen al análisis que se está realizando, por lo que no se las incluirá en la tabla comparativa.

Admite versiones, ya que dispone de una aplicación que implementa un sistema de control de versiones CVS que mantiene el registro de todo el trabajo y los cambios en los ficheros (código fuente principalmente) que forman un proyecto y permite que distintos desarrolladores colaboren.

En cuanto a su tipo de representación, Eclipse utiliza un objeto de modelo basado en un AST según se puede extraer de una de las secciones que componen la web oficial de la herramienta [93].

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Eclipse	Independiente	Source Code (Java, C, C++,Python)	SI	AST

Tabla 25. Eclipse. Tabla de la característica Target Artefact.

c.2 Design Smells:

Eclipse tiene como funcionalidad principal el desarrollo de aplicaciones y sólo se le reconoce un tipo de *smell* capaz de detectar: *Bad Coding Style*. Es bastante lógico que lo incorpore ya que es un tipo de error que se suele detectar en el momento de desarrollo y que intenta evitar errores de codificación o violaciones de algún estándar o convenio de código. En la *figura 26* se puede observar como en la pestaña *problems* que eclipse despliega se definen este tipo de errores que se han comentado.

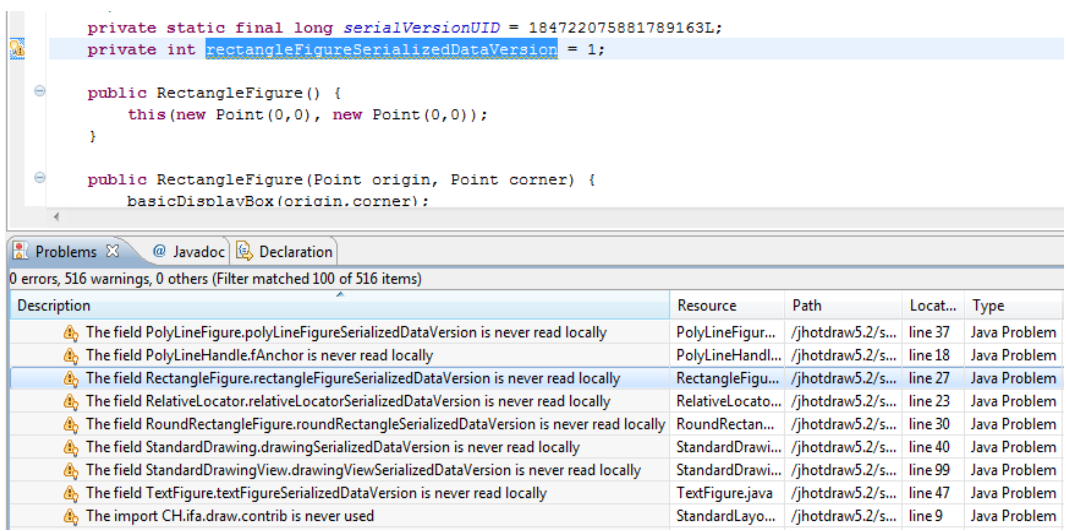


Figura 26. Eclipse. Bad Coding Style en Eclipse.

La granularidad está condicionada por el tipo de error que detecta. Al tratarse de *bad coding style* su campo de acción llega como máximo hasta nivel de clase, centrándose principalmente en nivel de método.

No existen relaciones de tipo inter e intra, debido a que las violaciones se producen en un determinado punto del código los cuales no se relacionan con otros posibles errores del mismo o diferente nivel. Tampoco analiza elementos relacionados con la herencia.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Eclipse	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO

Tabla 26. Eclipse. Tabla de la característica Design Smells.

c.3 Activity:

Eclipse realiza dos tipos de acciones relacionados con el tipo de error que muestra. La primera de ellas es la detección, que está basada en reglas provenientes de estándares y convenios de códigos. La herramienta realiza esta operación de detección de manera totalmente automática.

La segunda de las actividades es la de corrección que, al igual que la detección, y como es lógico, también está basada en reglas. Para llevar a cabo la corrección se necesita interactuar con Eclipse. Para ello hay que posicionarse con el ratón en uno de los problemas que haya detectado Eclipse (ver figura 26) y hacer clic sobre uno de ellos con el botón derecho; se desplegará un menú y se seleccionará *Quick Fix* para acceder al cuadro de corrección (ver figura 27).

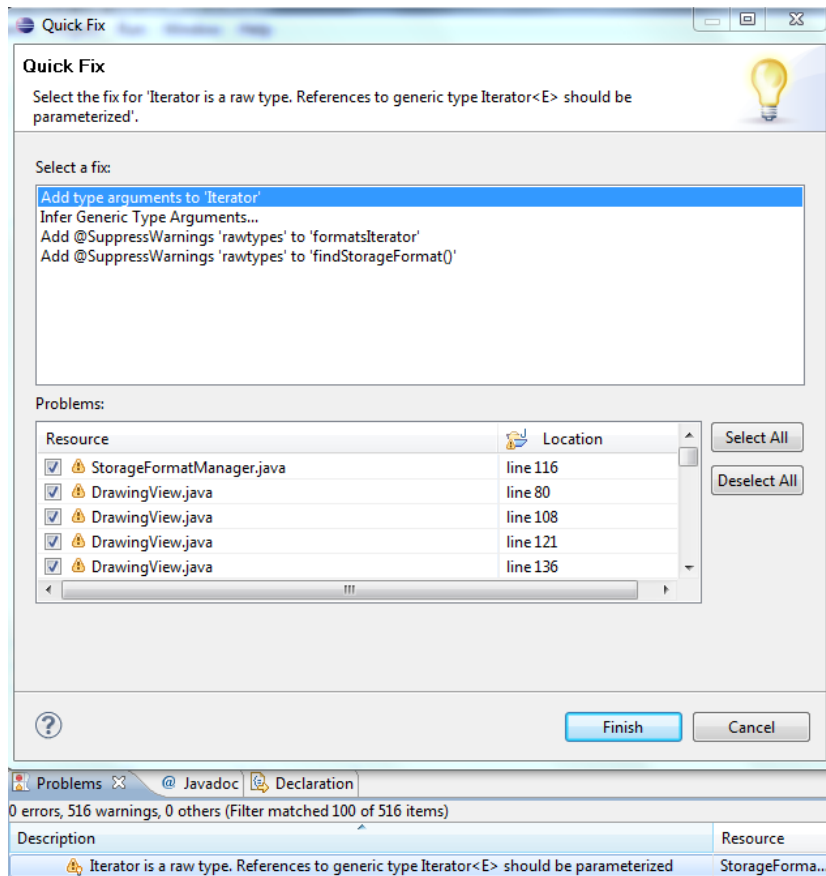


Figura 27. Eclipse. Cuadro de corrección en Eclipse.

En la ilustración se puede observar cómo el tipo de resultado que Eclipse tiene para la actividad de corrección es *Correction suggestions* ya que se proponen una serie de sugerencias para intentar solucionar el error del código seleccionado.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Eclipse	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Interactive	Correction Suggestions	Textual

Tabla 27. Eclipse. Tabla de la característica Activity.

FxCop

3.2.7. FxCop

a. Propósito

FxCop [22] es un programa que analiza ensamblados de código administrado, *Managed Code Assemblies*. El código administrado es el código del programa que se ejecuta como parte de la “Common Language Runtime” en un entorno .NET y devuelve información acerca del ensamblado, el posible diseño, la localización, el rendimiento y posibles mejoras de seguridad. La mayoría de los problemas se refieren a violaciones de las reglas de programación y diseño establecidas en las directrices de diseño (*Design Guidelines*), que son las directrices de Microsoft para escribir código robusto y fácil de mantener utilizando .NET Framework.

FxCop está destinado a desarrolladores de bibliotecas de clases pero también puede resultar útil para cualquiera que quiera crear aplicaciones que deban cumplir con las mejores prácticas de .NET Framework. FxCop también es útil como una herramienta educativa para las personas que son nuevas en .NET Framework o que no están familiarizadas con las directrices de diseño de .NET.

FxCop se distribuye bien como una aplicación completamente funcional que cuenta con una interfaz gráfica de usuario (FxCop.exe) para el trabajo interactivo y con una herramienta de línea de comandos (FxCopCmd.exe) adecuada para su uso como parte de la construcción de procesos automatizados o bien integrada con Microsoft Visual Studio .NET como una herramienta externa.

b. Información para el análisis

Versiones disponibles:

- FxCop 10.0 Application.
- Como plugin de Visual Studio. La versión 10.0 viene ya incluida en Visual Studio 2010, las versiones anteriores se integran con Microsoft Visual Studio .NET como una herramienta externa.

Versión analizada: FxCop 10.0 Application.

Sistema Operativo requerido:

- FxCop 10.0 ha sido probado con éxito en Windows XP, Windows Vista y Windows7, aunque según la página oficial funciona en todos los sistemas Windows desde el 2000.

Instalación:

- Para instalar la versión de FxCop 10.0 es necesario instalar el Kit de desarrollo Microsoft Windows SDK para Windows y Microsoft .NET Framework 4.0. Al finalizar la instalación, hay que ejecutar el setup de FxCop que se encuentra en
%ProgramFiles%\Microsoft SDKs\Windows\v7.1\Bin\FxCop\FxCopSetup.exe

Código utilizado para el análisis:

- QuickGraph [100].

c. Análisis

Pese a que se realizó un primer análisis sobre la versión 1.36. de la herramienta, posteriormente se lanzó una nueva versión, la 10.0, que es la que finalmente se ha analizado. No se han detectado más cambios que la compatibilidad de esta nueva versión con Microsoft .NET Framework 4.0. (De hecho, como se ha comentado anteriormente, sin esta versión no se puede instalar FxCop 10.0).

c.1. Target Artefact:

Tras la instalación de la herramienta, se procede a cargar código. FxCop está diseñado para el análisis de conjuntos de código (*Code assemblies*) en .NET 1.x, .NET 2.0, .NET 3.x y .NET 4.0 conformes a Microsoft .NET Framework. Un proyecto que se construye utilizando un código que no incluye ensamblados .NET no funciona en esta herramienta. Éstos ensamblados forman código de tipo .dll o .exe y han sido desarrollados en lenguaje C#, C++, CLI o Visual Basic .NET. Para realizar el análisis, FxCop utiliza un motor de análisis que desarma los ensamblados mediante meta-data API's y construye gráficos de llamadas a partir del ensamblado y gráficos de control de flujo para los métodos. FxCop implementa una combinación de análisis MSIL, análisis estático, y técnicas de análisis "*call-graph*" para identificar y reportar defectos de código:

- Análisis MSIL: "*FxCop incorpora un analizador para verificar código en Lenguaje intermedio de Microsoft (MSIL) que incluye el conjunto de las instrucciones CPU independientes para cargar, almacenar, inicializar y llamar a métodos en los objetos, así como instrucciones para operaciones aritméticas y lógicas, control de flujo, acceso directo a memoria, manejo de excepciones, y otras operaciones*" [56].

- Análisis estático de datos: FxCop analiza los distintos métodos de las *managed code assemblies* mediante la aplicación de normas definidas y de normas personalizadas para el proyecto a analizar. Usando estos análisis, la herramienta identifica los defectos de código que no se adhieren a las directrices de diseño .NET Framework. Los mensajes informativos ayudan a guiar al desarrollador para comprender las anomalías y hacer las correcciones necesarias en el código subyacente [56].

- Análisis Call-graph: FxCop internamente genera gráficos que representan las relaciones en forma de llamada entre varios métodos en las *managed code assemblies*. Estos gráficos de llamadas se utilizan para detectar las anomalías en la ejecución del programa o la violación de las pautas recomendadas. [56]

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
FxCop	Independiente / Plugin Visual Studio	Microsoft .NET Assemblies	NO	Object Model, Graphs

Tabla 28. FxCop. Tabla de la característica Target Artefact.

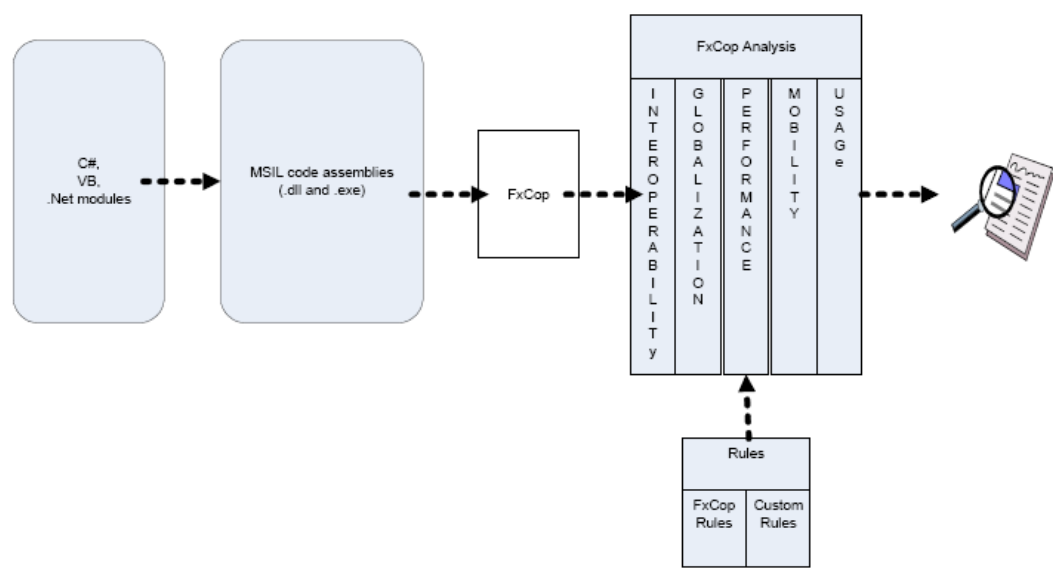


Figura 28. FxCop. Análisis de programa usando la herramienta FxCop.

c.2. Design Smells:

Como ya se ha podido observar en el propósito de la herramienta y en el apartado anterior, FxCop se encarga de buscar violaciones de los estándares de código establecidos por Microsoft en .NET Framework, por tanto, muestra el tipo de error que se ha denominado *Bad Coding Style*.

En lo que se refiere a la granularidad, el tipo de relación entre entidades a la hora de mostrar errores y al tratamiento de la herencia, no se puede comentar nada al respecto debido al desconocimiento del lenguaje que la herramienta analiza.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
FxCop	Bad Coding Style	-	-	-	-	-	-	-	-

Tabla 29. FxCop. Tabla de la característica Design Smells.

c.3. Activity:

La primera actividad que se detecta es la de Análisis, ya que como se ha comentado en el apartado c.1., FxCop crea una serie de gráficos para analizar el código. Es algo que la herramienta hace de manera automática cuando se carga código y que se realiza de forma interna, por lo que se desconoce si existe algún resultado relativo a esta actividad en la pantalla.

Como también se ha visto anteriormente, la herramienta permite seleccionar de una lista de reglas las que se desee que utilice para el análisis. Además, se pueden cargar otras reglas que se

tengan, siempre que se encuentren en formato FxCop Rule assemblies (.dll). Estas reglas son las que luego usará para detectar los errores que se producen en el código y en las que se basará para proponer sugerencias como la mostrada en la *figura 29*. Dichas sugerencias se pueden ver tanto en el apartado inferior de la pantalla principal del programa como haciendo doble clic sobre el error o advertencia de la parte de código que se quiera corregir.

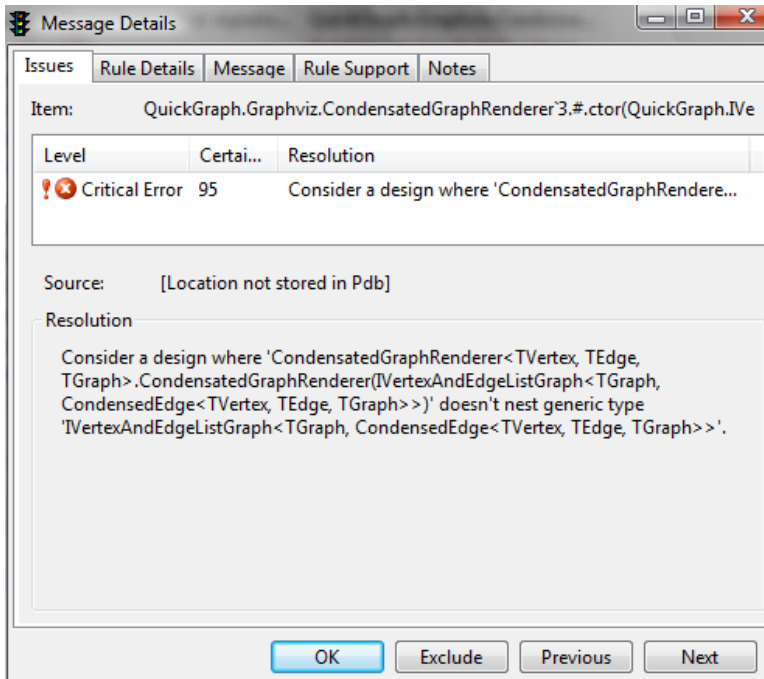


Figura 29. FxCop. Sugerencia de corrección mostrada al hacer doble clic en un error.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
FxCop	Especificación	Process Description	Interactive	Design Smell Specification	Textual
	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Rules-based	Interactive	Entity-smells relationships	Textual
	Corrección	Rules-based	Manual	Correction suggestions	Textual

Tabla 30. FxCop. Tabla de la característica Activity.

inCode

3.2.8. inCode

a. Propósito

inCode [56] es un plugin de Eclipse que ayuda al usuario a comprender, evaluar y mejorar la calidad del diseño de su proyecto. Está totalmente integrado, es discreto, y trabaja a medida que se escribe. inCode ahorra la tarea de revisión de código demasiado largo y la necesidad de sesiones de reestructuración, permitiendo detectar y corregir problemas de diseño de forma continua y ágil.

Como se comenta posteriormente en el apartado de análisis de la herramienta, inCode destaca por realizar sus tareas de detección de defectos de diseño conocidos (*Code Smells*) sobre “la marcha”, como la duplicación de código (*Code Duplication*), las clases que rompen la encapsulación (*Data Class, God Class*), o métodos que se encuentran en una clase equivocada (*Feature Envy*). La detección se basa en métricas orientadas a objetos.

b. Información para el análisis

Versiones disponibles:

- inCode 2.0.6 (Versión de demostración).

Versión analizada: inCode 2.0.6 (Versión de demostración).

Sistema Operativo requerido:

- Como plugin de Eclipse [26], está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).

Instalación:

- Como plugin de Eclipse con la url <http://www.intooitus.com/incode>.

Código utilizado para el análisis:

- jHotdraw5.2, ganttproject2 [101] y jfreechart.

c. Análisis

Antes de entrar en el análisis, es necesario comentar que la versión que se analiza de inCode es una versión de demostración (Demo). Esta Demo es una versión que la compañía pone a disposición de cualquier usuario para que vea su funcionamiento y posibilidades, y si está interesado adquiriera una versión personalizada. Por tanto, no es descartable que algunas de las características que aparecen aquí no aparezcan en una versión personalizada de la herramienta o que incluso aparezcan algunas nuevas características que la empresa pueda añadir en la versión personalizada a petición del propio usuario.

Como ya se ha comentado en el propósito de la herramienta, y según sus autores, la herramienta de demostración va a realizar la detección de *Disharmonies* mediante el cálculo de métricas. Debido a esta información, el análisis se inicia comprobando que efectivamente esto sea cierto.

c.1. Target Artefact:

inCode es una herramienta que se integra como plugin en Eclipse y como tal, analiza código fuente Java. No admite la comparación de versiones. Se desconoce el tipo de representación interna que la herramienta realiza para el análisis, por lo que se marca como desconocido mediante un guión.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
inCode	Plugin Eclipse	Source Code (JAVA)	NO	-

Tabla 31. inCode. Tabla de la característica Target Artefact.

c.2. Design Smell:

inCode, como se puede ver en las preferencias de análisis de la herramienta (Window → Preferences → inCode → Analysis preferences, mostrado en *figura 30*), analiza el código en busca de *Disharmonies* (Catalogadas así por los autores de la herramienta, aunque algunas aparecen también en el catálogo de *Bad Smells* [5]).

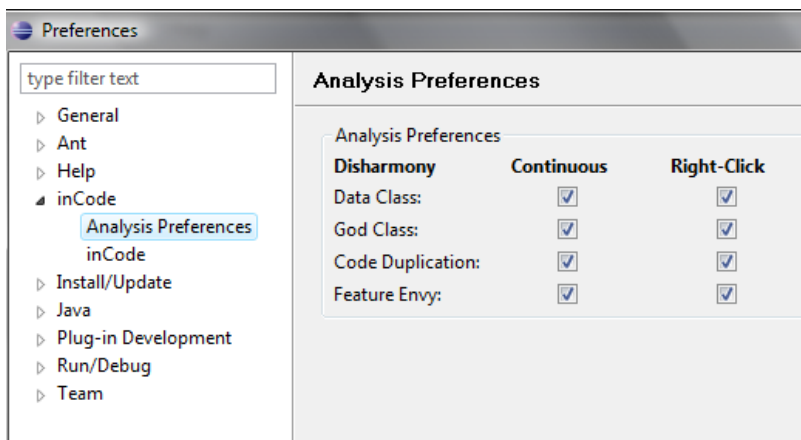


Figura 30. inCode. Preferencias de Análisis en las que se ve la lista de Disharmonies que soporta.

En la opción inCode dentro de Help → Help Guide → inCode user Guide → Visualisations, se puede ver también a que nivel realiza la herramienta las búsquedas de los errores de diseño para mostrarlos en una serie de gráficos (ver *figura 31*).

Además, en la búsqueda de *Disharmonies* la herramienta realiza acciones de tipo Inter, como puede ser en la detección de *Data Class*, en las que analiza distintas clases, y de tipo Intra, como por ejemplo en *Feature Envy*, donde mira distintos métodos y clases, buscando métodos que accedan de manera directa o indirecta a una gran cantidad de datos de otras clases.

Además, en la opción inCode Overview (Clic con el botón derecho en el proyecto a analizar y seleccionar inCode) se ve que aparecen mediciones métricas como *Average Width* (aparece en verde pero si hay un error se muestra en rojo, ver *figura 33*) en las que influye la herencia. La

aparición de la herencia también se ve mientras la herramienta carga la vista inCode Overview (ver figura 32).

inCode Visualizations

A picture is worth of thousand words ... and a thousand words take lots of time to utter.

In order to save your time, in inCode we propose 4 visualizations that capture many essential aspects of the programming elements on which they are defined, namely:

- [System Complexity \(system, packages\)](#) - size, complexity and distribution of design problems over classes
- [Class Blueprint \(classes\)](#) - interactions between methods and attributes of a class
- [Method Interaction \(methods\)](#) - interaction of a method with attributes (i.e. accessing them) and other methods (i.e. calling or being called)
- [Attribute Usage \(attributes\)](#) - who and how uses an attribute of a class

All these visualization techniques provide ways to uncover and utilize information about software systems.

In particular we use a special technique called *polymetric view* to visualize the software artifacts that we analyze using metrics. A *polymetric view* is a metrics-enriched visualization of software entities and their relationships. Their main benefit is that they can visually render numbers in a simple, yet effective and highly condensed way which is directly interpretable by the viewer.

Figura 31. inCode. Niveles a los que la herramienta trabaja para realizar gráficos.

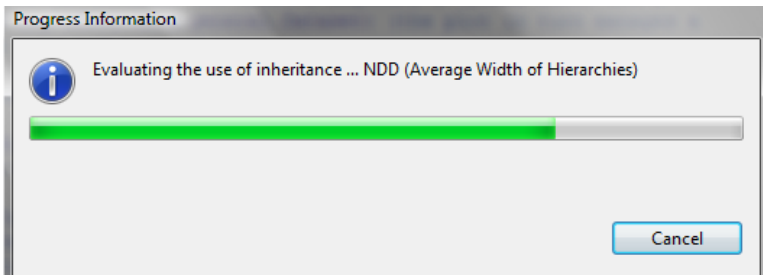


Figura 32. inCode. Uso de la herencia mientras se carga inCode Overview.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
inCode	Bad Smells / Disharmonies Metrics Warning	SI	SI	SI	SI	SI	SI	SI	SI

Tabla 32. inCode. Tabla de la característica Design Smells.

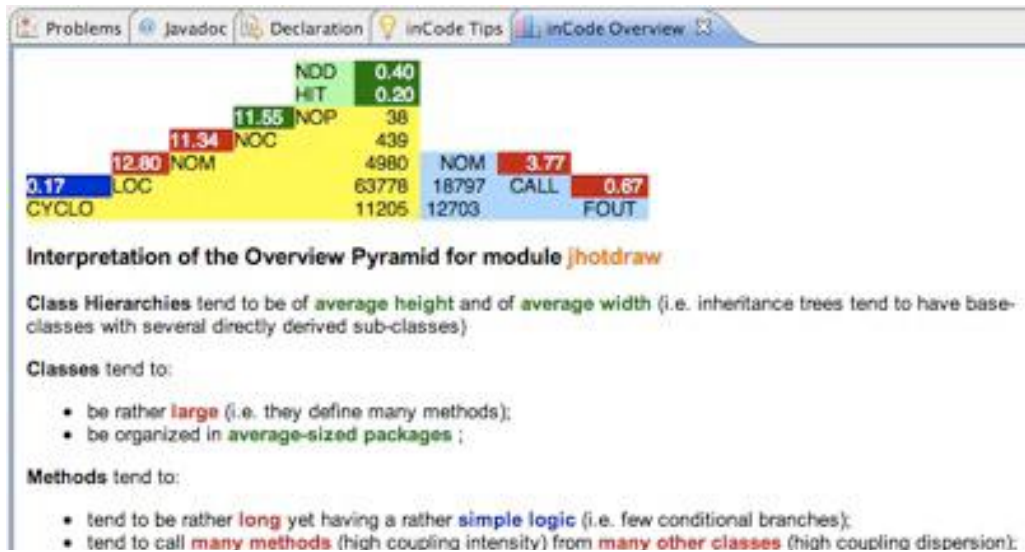


Figura 33. inCode. Vista inCode Overview en la que se ve las métricas y los errores.

c.3. Activity:

inCode es una herramienta que trabaja en un segundo plano de manera automática, aunque es posible hacerla trabajar también de manera interactiva. Los *Bad Smells* o *Disharmonies* que la herramienta identifica son *Data Class*, *God Class*, *Feature Envy* y *Code Duplication*, como ya se ha podido observar en la figura 30. Otro tipo de error que la herramienta detecta está relacionado con las métricas, reflejado en la figura 33.

En su trabajo en segundo plano, inCode va señalando los posibles problemas de diseño en el código con el uso de marcadores rojos junto a la línea de código que presenta el problema. Situando el ratón sobre estos marcadores o haciendo clic en ellos, se muestra el posible error y una serie de sugerencias para su posible solución (ver figura 35).

Otra opción que la herramienta incluye es en la que el usuario puede solicitar todos los errores detectados del proyecto, lo cual se realiza seleccionando la vista inCode Overview, en la que hay un apartado específico para las *Disharmonies* (ver figura 34). Seleccionando un error se muestran también sus sugerencias de corrección.



Figura 34. inCode. Listado de errores que aparecen en el proyecto ganttproject2.

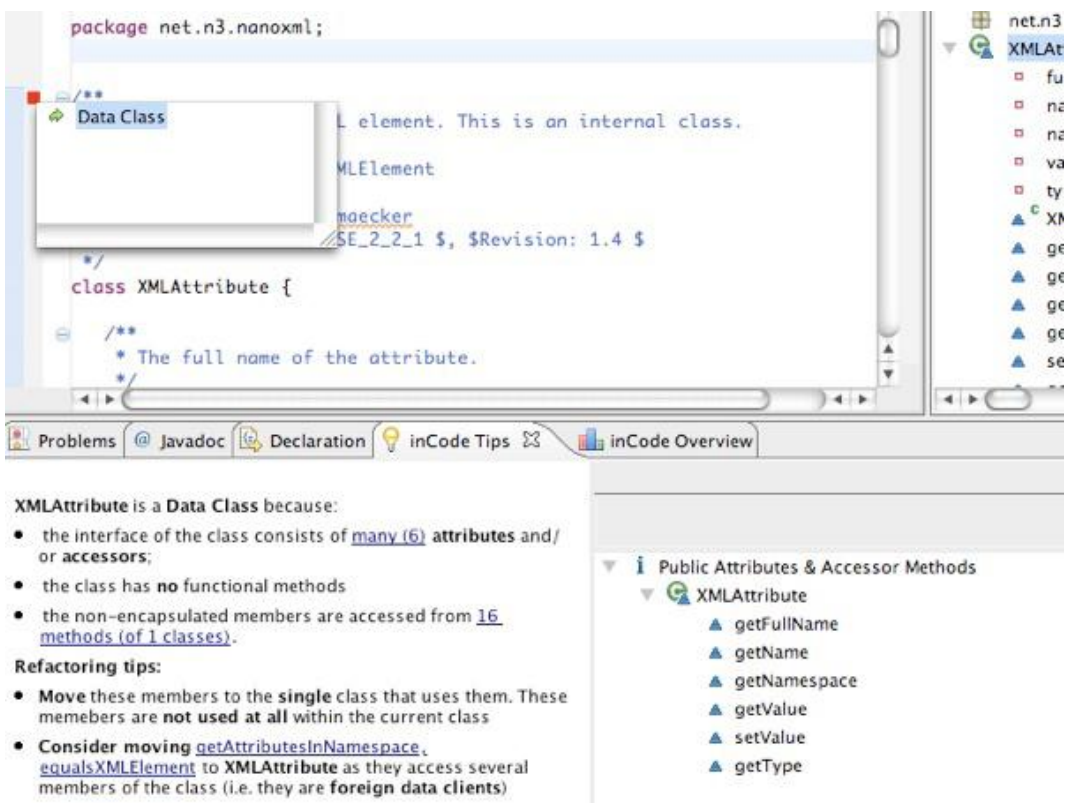


Figura 35. inCode. Detección de Disharmonies de manera automática. Sugerencias de corrección.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
inCode	Detección	Metric-Based	Interactive / Fully-automated	Entity-smells relationships	Textual
	Visualización	Visualisation-based	Interactive	Polymetrics views	Visual
	Corrección	Refactoring Suggestions	Interactive	Refactoring actions	Models

Tabla 33. inCode. Tabla de la característica Activity.

jDeodorant

3.2.9. jDeodorant

a. Propósito

jDeodorant [30] es un plugin de Eclipse que identifica problemas de diseño de software, conocidos como *bad smells* y los resuelve mediante la aplicación de refactorizaciones apropiadas. jDeodorant emplea nuevas metodologías con el fin de identificar en el código *smells* y sugerir las refactorizaciones convenientes para resolverlos. En la actualidad, la herramienta identifica cuatro tipos de *bad smells*: *feature envy*, *type checking*, *long method* y *god class*.

b. Información para el análisis

Versiones disponibles:

- jDeodorant Eclipse plugin.

Versión analizada: jDeodorant Eclipse plugin

Sistema Operativo requerido:

- Como plugin de Eclipse, está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en el apartado 3.2.6. Eclipse de esta memoria).

Instalación:

- Como plugin de Eclipse con la url <http://java.uom.gr/~jdeodorant/update/> o bien extrayendo en el directorio de instalación de Eclipse los archivos que se descargan de http://java.uom.gr/~jdeodorant/components/com_stalytics/filetrack.php?~jdeodorant/files_JDeodorant/JDeodorant.zip

Código utilizado para el análisis:

- lucene1.9 y Proyecto 09.

c. Análisis

Para realizar el análisis se revisa la versión 1.0.0 y la versión 4.0.0 (disponible desde el 12 de julio de 2010) que incluye la novedad de la detección del *bad smell god class*.

c.1. Target Artefact:

jDeodorant solamente funciona como plugin de Eclipse y analiza código escrito en Java. La herramienta no admite versiones, ya que no es posible realizar un análisis partiendo de dos proyectos iguales pero que hayan sufrido alguna corrección o modificación entre ellos y se quiera observar el progreso del código.

En cuanto a su tipo de representación, se hace mención en el artículo de *J. Pérez et al.* [1] a que el modelo utilizado para la representación interna del código del proyecto a analizar es un AST. Una explicación más detallada de dicho modelo aparece en varios artículos dedicados a jDeodorant, [57, 58, 59].

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
jDeodorant	Plugin Eclipse	Source Code (Java)	NO	AST

Tabla 34. jDeodorant. Tabla de la característica Target Artefact.

c.2. Design Smells:

jDeodorant es una herramienta que realiza el análisis centrándose en un único tipo de error, los *bad smells*. La herramienta no se encarga de detectar la totalidad de los *bad smells* catalogados, si no que focaliza su trabajo en encontrar algunos de ellos. La antigua versión de jDeodorant detectaba *Feature Envy*, *Type Checking* y *Long Method*, pero la nueva versión, jDeodorant 4.0.0, añade a los nombrados anteriormente la detección de *God Class*.

Para cada *bad smell* detectado, jDeodorant lo resuelve de un modo específico. *Feature envy* se soluciona mediante la refactorización denominada *move method*. Los problemas de *type checking* son resueltos mediante reemplazos condicionales con polimorfismo y códigos de reemplazo utilizando refactorizaciones *state/strategy*. Para *long method* se aplican refactorizaciones *extract method*. Y por último para *god class*, el tipo de refactorización que se seguirá para solucionar el *smell* será *extract class*.

La granularidad de la herramienta vendrá indicada principalmente por el análisis de los cuatro *bad smells* que detecta. Tres de ellos (*feature envy*, *type checking* y *long method*) analizarán el código a nivel de método, aunque no únicamente ya que *feature envy*, por ejemplo, utilizará dicho análisis para comprobar la distancia a la clase a la que pertenece y comprobar la distancia a otras clases del sistema por lo que se estará hablando también de un análisis a nivel de clase, ya que se tiene en cuenta esta entidad para resolver el *smell*. En cuanto a *god class* su análisis se realiza a nivel de clase.

Existen relaciones de tipo inter debido a que se analizan elementos de diferentes entidades como se ha comentado antes con *feature envy*. Existen también relaciones de tipo intra buscando conexiones entre elementos de la misma entidad.

jDeodorant no maneja análisis de ningún tipo relacionado con la herencia.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
jDeodorant	Bad Smells	NO	NO	NO	SI	SI	SI	SI	NO

Tabla 35. jDeodorant. Tabla de la característica Design Smells.

c.3 Activity:

La herramienta presenta tres tipos de actividades apreciables como son detección, corrección y análisis de impacto.

jDeodorant realiza la actividad de detección mediante una técnica basada en métricas. Para llevar a cabo dicha detección es necesario interactuar con la herramienta de la siguiente forma: Teniendo instalado correctamente el plugin en Eclipse, deberá aparecer en la barra de pestañas de la parte superior, un apartado con nombre “Bad Smells”. Si se selecciona se desplegarán los *bad smells* que la herramienta detecta. Si por ejemplo se quiere detectar si un determinado proyecto tiene *type checking*, la forma de proceder para su detección sería la siguiente:

Seleccionar la pestaña de “Bad Smells” → clic en *type checking* y se abrirá una nueva vista en la parte inferior. A continuación seleccionar el código fuente del proyecto a analizar del *package explorer* y en la vista abierta anteriormente aparecerá en su propia barra de herramientas una “i” (*identify bad smells*) donde se hará nuevamente clic. Se muestran entonces en la vista todos los *smells* detectados relacionados con este tipo de error. En la figura siguiente se puede observar gran parte de lo comentado.

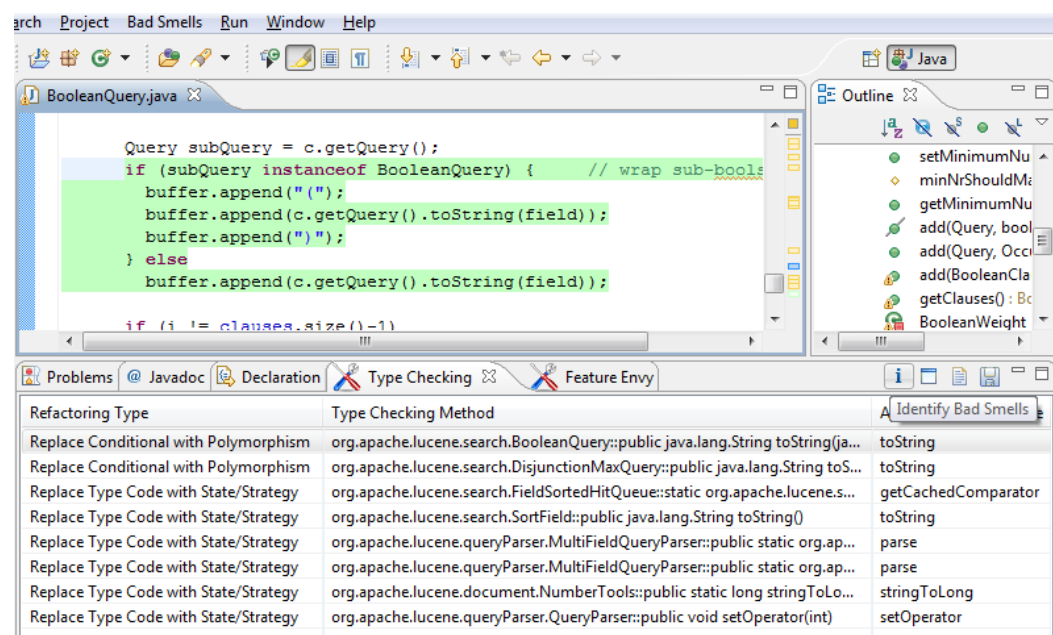


Figura 36. jDeodorant. Detección de *type checking*.

La actividad de corrección aplica técnicas de refactorización para llevar a cabo su función. Como sucede en la detección, es necesario interactuar con la herramienta para ejecutar la corrección. Para ello, y usando el ejemplo anterior de la detección, se mostrará la posible corrección del error visto en la *figura 36*.

En la barra de herramientas de la vista, justo a la derecha de la “i” (*identify bad smells*) se puede apreciar un icono simulando una ventana (*apply refactoring*). Si se hace clic en el icono teniendo seleccionado un *smell* de los detectados se desplegará una ventana emergente (ver *figura 37*) con la porción de código erróneo en un margen, y en el otro la porción de código de la refactorización a realizar. Por tanto si se hace clic en “OK” se llevará a cabo la corrección.

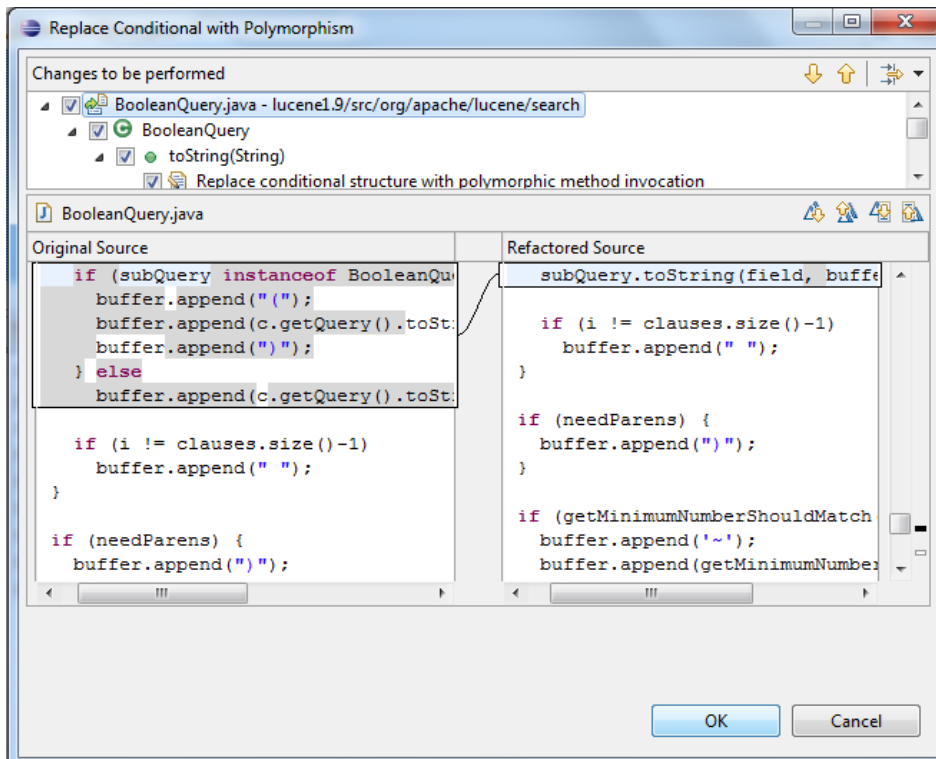


Figura 37. jDeodorant. Corrección mediante refactorización.

Como actividad característica de jDeodorant se encuentra el análisis de impacto. Esta actividad sólo está disponible para el tipo de *bad smell feature envy*. Con la vista abierta de *feature envy* se puede observar como existe un campo denominado *entity emplacement*. Este campo toma diferentes valores en función de los *smells* detectados y su cálculo se realizará teniendo en cuenta las distancias medias entre las clases. Los valores que se obtienen por cada *bad smell* indican el impacto que puede provocar una refactorización de manera virtual, y por tanto dan al usuario un criterio para aplicar las refactorizaciones más eficaces. jDeodorant muestra los candidatos a refactorización en orden ascendente con el fin de evidenciar cual sería la solución más efectiva.

Refactoring T...	Source Entity	Target Class	Entity Placement
Move Method	Negocio.Pedido::actualizar_lineas(java....	Negocio.LineaPedido	0.8750744415532554
Move Method	Negocio.Pedido::getCantidadLin(int):int	Negocio.LineaPedido	0.8774644510718175
Move Method	Negocio.Pedido::getTotal():int	Negocio.LineaPedido	0.879485673921173
Move Method	Negocio.Pedido::getSubtotalLin(int):int	Negocio.LineaPedido	0.879485673921173
Move Method	Negocio.ControladorNegocio::pedir_fe...	Negocio.Pedido	0.8797723591634892
Move Method	Vista.loginJFrame::jButtonSalirActionP...	Negocio.ControladorNego...	0.8800124493546384
Move Method	Vista.MenuJFrameProduct::jButtonSalir...	Negocio.ControladorNego...	0.8804991968416567

Figura 38. jDeodorant. Análisis de impacto.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
jDeodorant	Detección	Metrics-Based	Interactive	Entity-smells relationships	Textual, numerical
	Corrección	Refactoring Suggestions	Interactive	Refactoring actions	Textual, Model
	Análisis de impacto	Metrics-Based	Interactive	Metric-Impact	Numerical

Tabla 36. jDeodorant. Tabla de la característica Activity.

Nota: Es recomendable aumentar la memoria dedicada a Eclipse para el correcto funcionamiento de esta herramienta.

JRefactory

3.2.10. JRefactory

a. Propósito

JRefactory [60] es una aplicación que engloba una serie de herramientas (mostradas mediante pestañas en la aplicación), algunas de las cuales están relacionadas con la gestión de *Design Smells*, y que, o bien se pueden encontrar de manera independiente (la herramienta *Find Bugs*), o bien han sido creadas a partir de algunas ya existentes, como sucede con *Coding Standards*, que está creada a partir de PMD. También incluye una herramienta JRefactory, de desarrollo propio y que sirve para, después de cargar en ella el código, llegar a un diagrama de clases en el que se pueden aplicar las refactorizaciones y ver las métricas del proyecto.

Entre las opciones que la aplicación contiene aparece la denominada por sus autores como *Pretty Printer/Beautifier*. Es una herramienta que permite limpiar la sangría y el formato del código fuente Java. Para simplificar la escritura de comentarios javadoc, algunos métodos tienen comentarios generados automáticamente basándose en el nombre del método.

Además, la aplicación, en algunas de sus versiones, incluye opciones para visionado de diagramas UML y su impresión (*UML Diagrams* y *Printing*) entre otras y que sirven para, por ejemplo, mostrar los diagramas de clases en los que poder hacer las refactorizaciones.

b. Información para el análisis

Versiones disponibles:

- Se encuentran disponibles varias versiones desde la 2.6 hasta la 2.9.19. Están disponibles para diversos IDE y también de manera independiente.

Versión analizada: JRefactory 2.8. Versión independiente disponible desde la Web.

Sistema Operativo requerido:

- La versión analizada funciona mediante Java Web Start, por lo que es de suponer que funcione en cualquier sistema operativo que permita la instalación Java Web Start. Ha sido probada con éxito en Ubuntu, Windows Vista y Windows 7.

Instalación:

- La herramienta se ejecuta mediante un enlace disponible en la web [57], aunque para ello es necesario tener instalado el JDK de Java y Java Web Start.

Código utilizado para el análisis:

- Ganttproject 2.0.1., PMD Source Code [29] y jFreechart.

c. Análisis

Se opta para efectuar el análisis por la versión de JRefactory disponible desde el sitio web de la herramienta [60] y que funciona con Java Web Start debido a que no requiere instalación ya que durante la instalación de la versión independiente surgen diversos fallos.

En el sitio web oficial se comenta que la herramienta se ha dejado de desarrollar a partir de la versión 2.9.19 para incluirse en JavaStyle, por lo que se prueba a instalar dicha herramienta en jEdit y NetBeans siguiendo las instrucciones indicadas en la web pero sólo se encuentran disponibles las últimas versiones de JavaStyle y estas ya no incluyen la opción de JRefactory. En la instalación del archivo JavaStyle que se puede descargar desde la web para estos dos IDE's surgen problemas que debido al desconocimiento de jEdit en unos casos y a la no existencia de soluciones para el caso de NetBeans no son posibles de solucionar. Además, las instrucciones de instalación de la aplicación en ellos, que deberían aparecer en la página web, no se encuentran disponibles al completo.

En definitiva, la herramienta ha caído en desuso ya que su última versión disponible data de 2004 y posteriormente se integró en JavaStyle pero en las nuevas versiones ha desaparecido.

c.1. Target Artefact:

Como ya se ha comentado, la herramienta se encuentra disponible para diferentes entornos de desarrollo integrados (IDE's), los cuales aparecen indicados en el sitio web, y también de forma independiente, bien a través del enlace web o bien mediante la descarga del archivo correspondiente y su posterior ejecución desde línea de comandos.

La herramienta analiza código fuente Java, aunque hay algunas herramientas internas como *Coding Standards* que admiten archivos en formato .jar. No existe ninguna opción de comparación ni similar entre dos proyectos por lo que la herramienta no admite versiones.

Dentro de la herramienta existe una pestaña que permite ver la representación AST del código que cargamos, lo que lleva a pensar que la herramienta usa este tipo de representación interna, aunque no se ha podido verificar debido a la poca información disponible en relación con JRefactory.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
JRefactory	Independiente / Plugin jEdit / Plugin NetBeans 3.6. / Plugin jBuilderX	Source Code (JAVA) Executable Code (Java bytecode)	NO	AST

Tabla 37. JRefactory. Tabla de la característica Target Artefact.

c.2. Design Smells:

Como ya se ha comentado, la herramienta cuenta con una pestaña denominada *Coding Standards* (Se muestra en la *figura 39*). Esta pestaña sirve para identificar las violaciones de los estándares de código Java que pueda haber en el proyecto cargado. Es lo que se viene denominando en esta memoria como *Bad Coding Style*. Para cada una de las violaciones de código, muestra una sugerencia de refactorización que después el usuario tendrá que desarrollar, ya que no proporciona una solución, sino donde está ubicado el problema. (Por ejemplo sugiere renombrar una clase para cumplir los estándares de código pero no dice que nombre se debería poner o que características debe tener para cumplir dichos estándares).

Aunque, como se puede ver en la *figura 40*, JRefactory calcula métricas, éstas no muestran ningún tipo de error, es decir, la herramienta proporciona unos valores y el usuario se encarga de

interpretarlos. Debido a esto, no se ha considerado oportuno incluir un campo en Tipo de Error que sea *Metrics Warnings*.

Otra de las características de JRefractory es la posibilidad de detectar errores *Cut & Paste*, los cuales están incluidos en los listados de Antipatrones que se están utilizando (ver *tablas 1 y 2*). Dichos errores los busca tanto a nivel de clase como de método y también de manera conjunta.

Debido a los errores que detecta, no es posible que haya herencia ni relaciones de ningún tipo. La búsqueda de errores en estándares de código condiciona que no se localice una granularidad superior a nivel de clase relacionada con el *Bad Coding Style*.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
JRefractory	Bad Coding Style Antipatterns	NO	NO	NO	SI	SI	NO	NO	NO

Tabla 38. JRefractory. Tabla de la característica Design Smells.

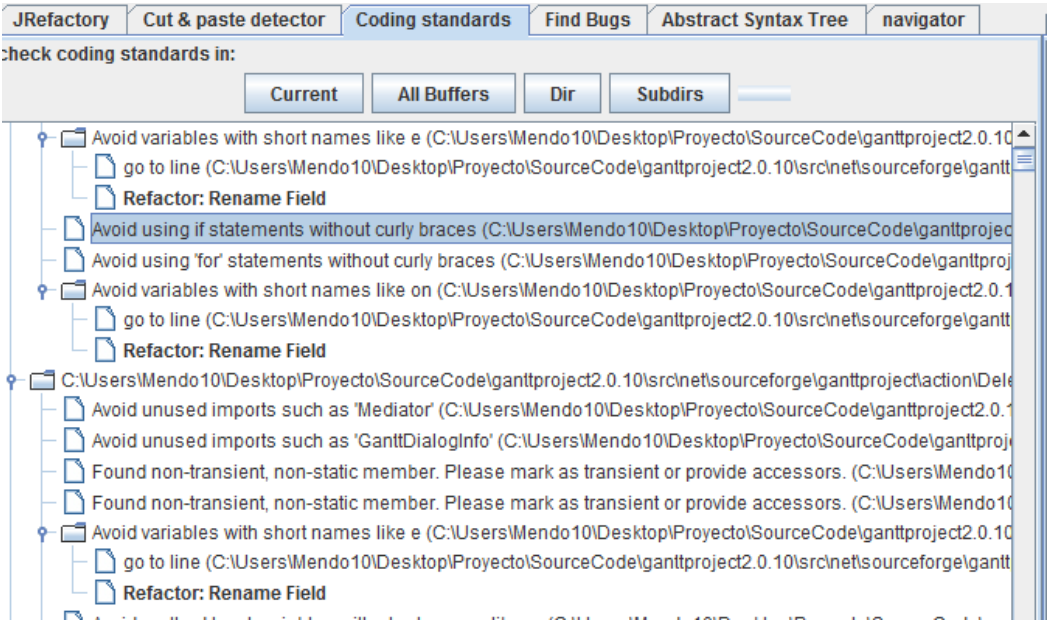


Figura 39. JRefractory. Coding Standards. Posibles errores en código y sugerencias de refactorización.

c.3. Activity:

Aunque en la *figura 39* se observa que entre los errores aparece un campo Refactor, éste sólo aparece en algunos de ellos y no aparece una refactorización como tal, sino una ligera sugerencia de lo que se puede hacer (En ningún momento se da una solución ni en la sugerencia se indica cómo llevarla a cabo). Por estos motivos, no se ha incluido un apartado de corrección dentro de las

actividades que la herramienta realiza. Pese a ello, la herramienta sí incluye algunas opciones de refactorización, que aparecen al hacer clic con el botón derecho sobre el Diagrama de Clases que, como ya se ha comentado antes, aparece desde la pestaña JRefactory (Concretamente con doble clic en una clase). Como se puede ver, estas refactorizaciones son bastante sencillas y de nuevo la herramienta no proporciona ninguna sugerencia, por lo que es el usuario el que debe encargarse de la tarea.

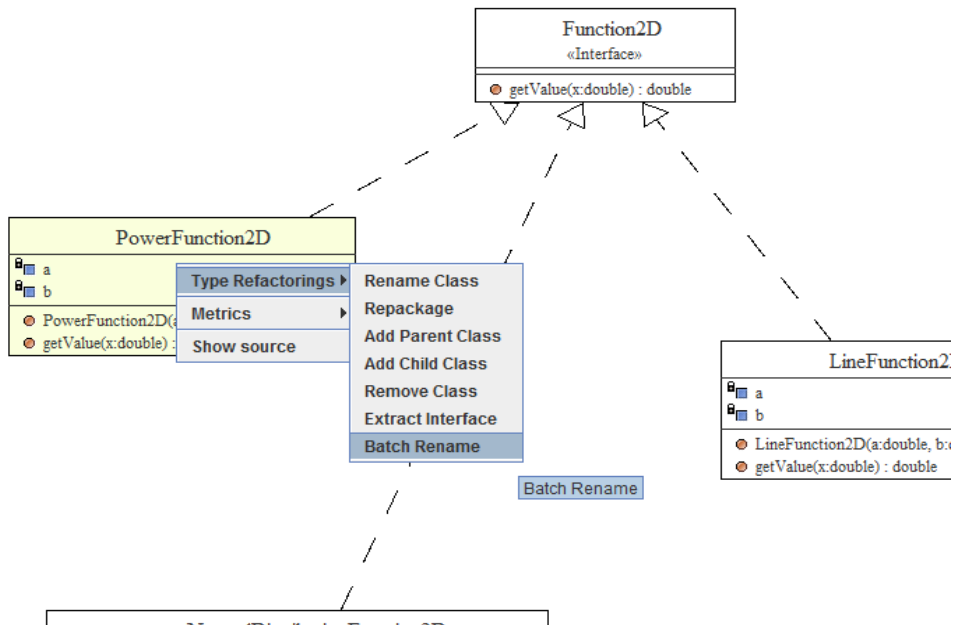


Figura 40. JRefactory. Opciones de refactorización.

Una de las características que es de resaltar es el Análisis (Aunque todas las herramientas lo hacen, no en todas es destacable debido a que lo realizan de manera totalmente interna y en la mayoría de ocasiones ni siquiera se conoce el cómo) ya que JRefactory muestra el AST del código que se tenga cargado en el Buffer en ese momento.

También aparece la opción de detección, la cual se realiza mediante reglas de manera interactiva y que se muestran, como se ha visto en la *figura 39*, de manera textual.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
JRefactory	Análisis	Model Extraction	Fully-automated	-	Visual
	Detección	Rules-Based	Interactive	Entity-smells relationships	Textual

Tabla 39. JRefactory. Tabla de la característica Activity.

M2 Resource Standard Metrics

3.2.11. M2 Resource Standard Metrics

a. Propósito

M2 Resource Standard Metrics, RSM [61] es una herramienta que analiza la calidad del código fuente y sus métricas. RSM proporciona un método estándar para el análisis de C, ANSI, C++, C# y código fuente Java. Ofrece a los usuarios de la herramienta la posibilidad de estandarizar la medición de la calidad del código fuente y el análisis de métricas a través de su organización. RSM proporciona métodos fáciles, rápidos y flexibles para ayudar en la medición de la calidad del código y el cálculo de métricas.

b. Información para el análisis

Versiones disponibles:

- Independiente, Visual Studio plugin, .NET, jBuilder, Eclipse (Como External Tool) y otras IDE's conocidas.

Versión analizada: Resource Standard Metrics Wizard 7.70 (independiente)

Sistema Operativo requerido:

- La versión independiente que es la que se ha analizado está disponible para ser instalada bajo plataformas Windows, Linux y Mac OS-X

Instalación:

- La versión independiente de RSM se obtiene en la sección de descargas del sitio web de la herramienta: http://msquaredtechnologies.com/m2rsm/rsm_demo.php

El archivo comprimido descargado tendrá en su interior un ejecutable que llevará a cabo su instalación de forma automática una vez lanzado.

Código utilizado para el análisis:

- lucene1.9.

c. Análisis

El análisis esta realizado sobre la herramienta independiente de RSM, debido a que se ha encontrado su instalación y manejo más práctico y rápido para el usuario. La versión que se puede encontrar en la web para su descarga es la 7.75 en el momento de realizar la memoria, que viendo el log, no parece presentar muchos cambios respecto a la versión analizada (7.70).

c.1. Target Artefact:

RSM puede funcionar como herramienta independiente, pero también puede estar integrada en varios entornos de desarrollo como son Eclipse, Visual Studio 6, Visual Studio .NET, jBuilder, UltraEdit/Studio y Return. Puesto que se analiza la versión independiente de RSM, no se especifica como sería la instalación y manejo de la herramienta en los diferentes IDE's en los que puede

funcionar. La información necesaria para proceder a la integración de la herramienta en los diferentes entornos mencionados anteriormente se podrá encontrar en la dirección http://msquaredtechnologies.com/m2rsm/docs/ide_menu.htm.

Como ya se ha dicho en la presentación de la herramienta, RSM analiza código fuente escrito en C, C++, C# y Java. No admite la comparación de distintas versiones de código ya que no es posible realizar un análisis comparativo entre dos o más versiones de un mismo trabajo.

En cuanto a su representación interna, no se ha encontrado información sobre la herramienta que especifique cómo se lleva a cabo, aunque según la estructura de la herramienta y su modo de trabajar es posible pensar que se trate de un *object model*.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
M2 Resource Standard Metrics	Independiente Plugin de: Eclipse/ Visual Studio 6/ Visual Studio .NET/ jBuilder/ UltraEdit-Studio/ Return	Source Code (C,C++,C#, Java)	NO	Object Model

Tabla 40. M2 Resource Standard Metrics. Tabla de la característica Target Artefact.

c.2 Design Smells:

RSM proporciona diferentes informes que están basados en métricas. La gran mayoría de ellas son datos que describen de forma numérica el comportamiento y la estructura del proyecto analizado. En este análisis sólo interesan las métricas de “emergencia” que son aquellas que miden, detectan y en algunos casos, dependiendo de la herramienta, marcan o resaltan los valores que sobrepasan los establecidos por la herramienta o por el usuario. En este caso, ha costado encontrar dichos *metrics warnings* debido a la forma de presentarlos que tiene RSM. A la vez que se muestran los *metrics warnings*, también se observa la aparición de otro tipo de problema o defecto de diseño. Este *smell* detectado es el *bad coding style*, que atendiendo a una serie de estándares o convenciones de código muestra aquellas líneas de código que no los cumplen.

A continuación se explica brevemente como realizar el análisis para la detección de los *smells* permitidos por la herramienta, ya que de primeras el interfaz de la aplicación M2 Resource Standard Metrics puede resultar algo complejo y confuso para interactuar con él si no se está familiarizado:

Se ejecuta la versión independiente de M2 Resource Standard Metrics. Para realizar las dos operaciones que en este estudio interesan (*Bad Coding Style* y *Metrics Warnings*), se comienza eligiendo los ficheros que se quieren analizar. Para ello se debe hacer clic en el botón de la parte superior, *choose or modify files*. Al hacerlo se activa la zona izquierda de la interfaz de la herramienta. A continuación se pulsa en el botón *set path* y se abre una ventana para que se seleccionen los ficheros del proyecto que se desean analizar. Una vez seleccionados se pulsa en el botón *acquire files* para visualizar el contenido de las carpetas a analizar. A su derecha también se debe hacer clic sobre el botón *files to list* para importar todas las clases que intervienen en el proyecto y que se pueden observar en la parte inferior izquierda.

El siguiente paso es seleccionar el formato de salida del análisis, para ello se pulsa en *load or name output files*. Se activa entonces la parte superior derecha de la interfaz de la herramienta donde se elige un nombre y un formato para el informe de resultados.

A continuación se pulsa en *select/create reports*, activando así la parte inferior derecha de la interfaz de la herramienta donde se debe marcar el tipo de análisis que se quiere realizar. El que interesa para el análisis de este trabajo de toda la lista que se presenta es *Total quality profile*. Para finalizar se pulsa *execute RSM* y en la parte inferior se hace clic en *execute* para que de forma automática se cree el informe con el análisis de los ficheros seleccionados en el formato de salida elegido (Para la realización de esta memoria se ha elegido el formato html).

En la figura siguiente se muestra el interfaz de la herramienta, donde se pueden observar todos los pasos explicados anteriormente para llegar a realizar el análisis.

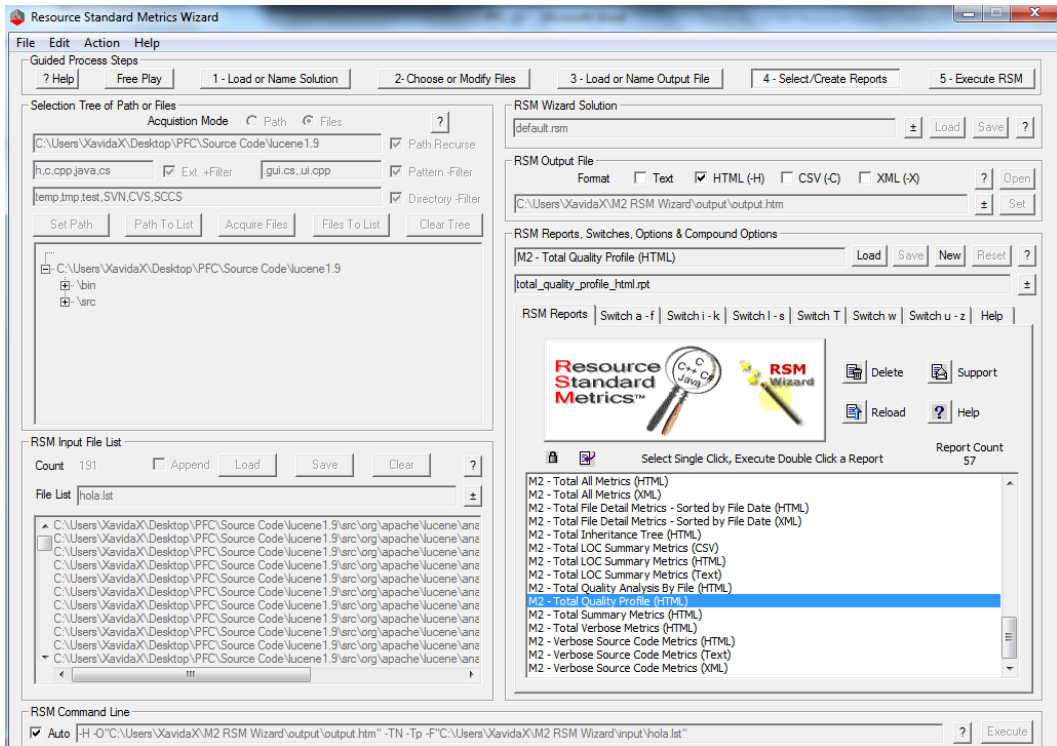


Figura 41. RSM. Interfaz de M2 Resource Standard Metrics.

Respecto a la granularidad, siempre que se efectúe un análisis de código Java, se puede considerar que llega hasta un nivel de paquete, que es hasta donde las métricas llegan indicando un posible *smell*. A partir del nivel de subsistema, las métricas que RSM muestra sólo son cálculos de medias y sumas totales de los niveles inferiores.

Existen relaciones inter e intra, ya que las métricas calculadas intervienen en *smells* de entidades tanto del mismo como de diferente nivel.

RSM proporciona una opción para realizar un informe denominada *Total Inheritance tree*, pero se desconoce si dicho informe hace mención a algún tipo de *smell* relacionado con la herencia. Pese a ello, la herramienta permite efectuar operaciones relacionadas con la herencia por lo que se marca la casilla como un si.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
M2 Resource Standard Metrics	Metrics Warnings Bad Coding Style	NO	NO	SI	SI	SI	SI	SI	SI

Tabla 41. M2 Resource Standard Metrics. Tabla de la característica Design Smells.

c.3. Activity:

RSM sólo tiene un tipo de actividad reconocido, la detección. Esta actividad se realiza una vez que se ejecuta el tipo de análisis seleccionado, que como se ha visto con anterioridad debe ser *Total quality profile* para que muestre los diferentes tipos de error. La detección de esos *smells* está basada en métricas y se realiza de manera interactiva con el usuario. En la *figura 42* se observa como se presenta el informe en formato textual (html) del proceso de detección realizado.

En la documentación disponible en su web se puede leer el siguiente fragmento, “*The configuration file allows the user to customize the degree of source code analysis.*” [61], que hace pensar que tenga una actividad de especificación, pero que no se ha podido corroborar debido a lo complejo de la herramienta.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
M2 Resource Standard Metrics	Detección	Metrics-Based	Interactive	Entity-metrics relationships	Textual, numerical

Tabla 42. M2 Resource Standard Metrics. Tabla de la característica Activity.

```

      ~~ Total Metrics For 10 Files ~~

-----

      ~~ Project Quality Profile ~~

Type  Count Percent  Quality Notice
-----
1      225      5.67  Physical line length > 80 characters
2        4      0.10  Function name length > 32 characters
17      27      0.68  Function comment content less than 10.0%
18       2      0.05  Function eLOC > maximum 200 eLOC
20       1      0.03  File comment content < 10.0%
27      11      0.28  Number of function return points > 1
28       1      0.03  Cyclomatic complexity > 10
29       4      0.10  Number of function parameters > 6
30     3483     87.71  TAB character has been identified
31       1      0.03  Class/Struct comments are < 10.0%
35       1      0.03  Class specification contains public data
38      13      0.33  Exception Handling "try"- "catch" has been identified
46      33      0.83  Function/Class Blank Line content less < 10.0%
47       1      0.03  File Blank Line content < 10.0%
48       7      0.18  Function lLOC <= 0, non-operational function
49      49      1.23  Function appears to have null or blank parameters
50      12      0.30  Variable assignment to a literal number
51      85      2.14  No comment preceding a function block
52       9      0.23  No comment preceding a class block
119     1      0.03  Return is not a function
121     1      0.03  class name not proper cased

```

Figura 42. RSM. Informe originado tras ejecutar la opción de detección.

PMD

3.2.12. PMD

a. Propósito

PMD [29] es una herramienta de código abierto (*Open source*). Es un analizador estático de código fuente, concretamente analiza código fuente Java basado en unas normas de evaluación que se han habilitado durante la ejecución. La herramienta viene con un conjunto predeterminado de reglas que pueden ser utilizadas para descubrir errores comunes de desarrollo tales como variables que nunca se utilizan, objetos que son innecesarios, etc. Además, PMD también permite a los usuarios ejecutar análisis personalizados que les permitan elaborar nuevas normas de evaluación de una forma cómoda. La herramienta viene con una interfaz de línea de comando relativamente fácil de usar y, al mismo tiempo, también se puede integrar con diversos entornos de desarrollo populares, tales como Eclipse. PMD actualmente sólo es compatible con Java.

b. Información para el análisis

Versiones disponibles:

- Se encuentran disponibles versiones para diversos entornos de desarrollo. A fecha de realización de esta memoria su versión de desarrollo es la 4.2.5.

Versión analizada: PMD 4.2.5. Eclipse plugin.

Sistema Operativo requerido:

- Como plugin de Eclipse, está restringido a los Sistemas Operativos en los que éste funciona (Ver Sistema Operativo requerido en 3.2.6. Eclipse).

Instalación:

- Como plugin de Eclipse con la url <http://pmd.sourceforge.net/eclipse>.

Código utilizado para el análisis:

- Proyecto 09, jHotDraw5.2.

c. Análisis

En el análisis de PMD se ha utilizado la versión disponible como plugin de Eclipse ya que su instalación es más sencilla que la versión independiente, al tener que añadir en esta última valores a la variable \$PATH y tener que trabajar desde línea de comandos. Según se observa en la web, no debería haber diferencias en la forma en la que PMD trabaja ni en como muestra sus resultados, sólo varía la interfaz.

c.1. Target Artefact:

PMD trabaja generando y recorriendo un árbol de sintaxis abstracta (AST). Se comentará su funcionamiento mucho más detallado en el apartado **c.3. Activity**.

Hay tres formas diferentes para la utilización de PMD: como una línea de comandos (herramienta independiente), como un plugin de un IDE (por ejemplo Eclipse), o como un

elemento Ant (la lista completa de entornos de desarrollo y herramientas con los que PMD puede trabajar, a día de realizar esta parte de la memoria, además de la herramienta independiente, es JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, y Emacs). Como ya se ha comentado, se ha centrado el análisis sobre la versión diponible como plugin de Eclipse.

La herramienta incluye un detector de *Cut & Paste* que no sólo admite Source Code como formato de entrada como sucede con el resto de opciones de PMD sino que también admite JSP, C, C++, Fortran, Ruby y PHP code.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
PMD	Independiente / Plugin Eclipse / Plugin jEdit / Plugin Net Beans / Otros IDE	Source Code (JAVA) CPD admite: Java, JSP, C, C++, Fortran, Ruby y PHP code	NO	AST

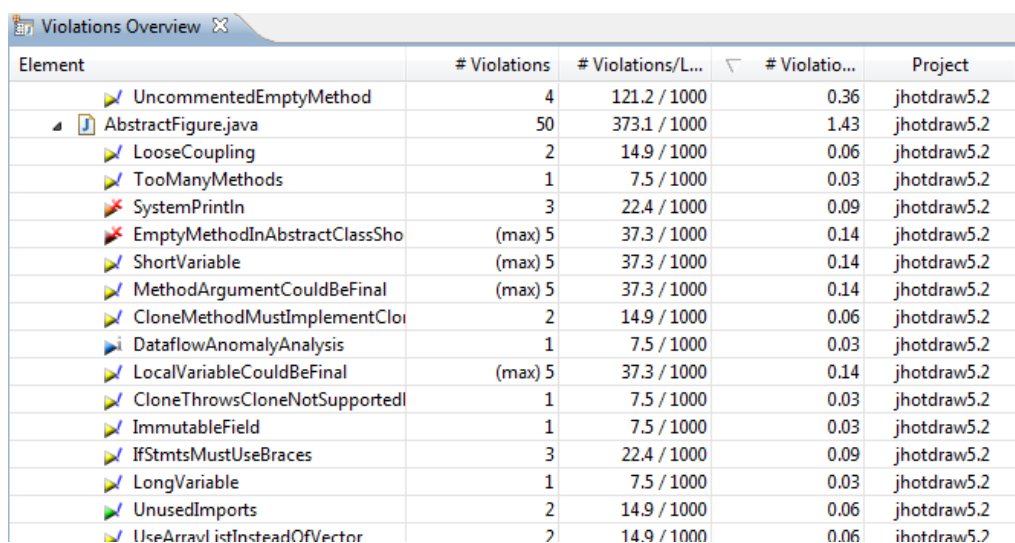
Tabla 43. PMD. Tabla de la característica Target Artefact.

c.2. Design Smells:

Tras realizar un análisis con la herramienta, lo primero que se observa es que en el explorador de paquetes los archivos con violaciones están señalados con marcas de error, pero las marcas de error son un poco confusas, ya que son idénticas a las marcas de errores de compilación que el propio Eclipse muestra. En los editores del código fuente, las violaciones también se muestran con marcadores. Hay una ventana general que tiene por objetivo proporcionar un resumen de las violaciones, permitiendo cambiar los niveles de gravedad que se muestran. Todas estas marcas de errores o posibles violaciones están dentro de lo que se viene definiendo en esta memoria como *Bad Coding Style*. Además, como se puede ver en la *figura 43*, y ya se ha comentado anteriormente, PMD analiza el código fuente de Java basándose en unas normas de evaluación que se han habilitado durante la ejecución (lo que confirma el tipo de error *Bad Coding Style*). Al igual que pasaba con JRefactory, PMD sólo analiza el código en busca de problemas a nivel de clase y de método y lo hace con relaciones tanto de tipo inter como de tipo intra y sin tener en cuenta la herencia. Como se acaba de comentar, PMD incluye un detector de *Cut & Paste*, lo cual se tiene catalogado como un antipatrón.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
PMD	Bad Coding Style Antipatterns	NO	NO	NO	SI	SI	SI	SI	NO

Tabla 44. PMD. Tabla de la característica Design Smells.



Element	# Violations	# Violations/L...	▽ # Violatio...	Project
UncommentedEmptyMethod	4	121.2 / 1000	0.36	jhotdraw5.2
AbstractFigure.java	50	373.1 / 1000	1.43	jhotdraw5.2
LooseCoupling	2	14.9 / 1000	0.06	jhotdraw5.2
TooManyMethods	1	7.5 / 1000	0.03	jhotdraw5.2
SystemPrintln	3	22.4 / 1000	0.09	jhotdraw5.2
EmptyMethodInAbstractClassSho	(max) 5	37.3 / 1000	0.14	jhotdraw5.2
ShortVariable	(max) 5	37.3 / 1000	0.14	jhotdraw5.2
MethodArgumentCouldBeFinal	(max) 5	37.3 / 1000	0.14	jhotdraw5.2
CloneMethodMustImplementClon	2	14.9 / 1000	0.06	jhotdraw5.2
DataflowAnomalyAnalysis	1	7.5 / 1000	0.03	jhotdraw5.2
LocalVariableCouldBeFinal	(max) 5	37.3 / 1000	0.14	jhotdraw5.2
CloneThrowsCloneNotSupportedException	1	7.5 / 1000	0.03	jhotdraw5.2
ImmutableField	1	7.5 / 1000	0.03	jhotdraw5.2
IfStmtsMustUseBraces	3	22.4 / 1000	0.09	jhotdraw5.2
LongVariable	1	7.5 / 1000	0.03	jhotdraw5.2
UnusedImports	2	14.9 / 1000	0.06	jhotdraw5.2
UseArrayInsteadOfVector	2	14.9 / 1000	0.06	jhotdraw5.2

Figura 43. PMD. Vista general en la que se muestran las violaciones de código detectadas.

c.3. Activity:

PMD usa, como ya se ha hablado, un árbol AST para realizar la actividad de detección de las violaciones de código. Con base en la información obtenida de la página web oficial del proyecto PMD en sourceforge.com [29] y en el documento [62], el funcionamiento interno de la herramienta (Es decir, su fase de análisis) se puede resumir de la siguiente manera: (1) El usuario suministra la ubicación del código fuente que tiene que ser analizado junto con la norma o normas que le gustaría ejecutar, (2) la herramienta abre un flujo de datos de lectura para leer el código fuente y transferirlo a un analizador de código Java que a su vez genera un árbol de sintaxis abstracta (AST), (3) el AST se devuelve a PMD, que a su vez se lo da a la capa de la tabla de símbolos, que identifica los ámbitos, las declaraciones, y los diversos usos; (4) si una norma en particular (que está habilitada) implica el análisis de flujo de datos, PMD facilita el AST a la capa del autómata finito determinista (DFA) que a su vez genera gráficos de control de flujo y los nodos de flujo de datos. (5) Con todos estos datos obtenidos, cada regla atraviesa el árbol de sintaxis abstracta como sea necesario y detecta los problemas sobre la base de este de recorrido, las normas también pueden utilizar las tablas de símbolos y los nodos dentro del DFA generado; (6) los problemas identificados en el paso 5 se imprimen por pantalla [62]. Todo este proceso PMD lo realiza de forma automática nada más que se carga el código fuente.

Si en la barra de menús de Eclipse se selecciona Window → Preferences → PMD → Rules Configuration, se apreciarán todas las reglas que aparecen cargadas por defecto en el programa y las opciones para editarlas, importar las que ya se tengan creadas o incluso crear una nueva (botón *Rule Designer*). Debido a esto, se ha separado en dos la opción de especificación, ya que hay una clara diferenciación entre que un usuario pueda modificar las reglas ya existentes o crear una nueva con todo el código que ello implica (el código de esa nueva regla formará un programa en sí, encargado de buscar las violaciones en el artefacto objetivo). Sea cual sea el método mediante el que se obtenga una nueva regla, ésta se añadirá al listado del apartado Rules que se observa en la figura 44 y por tanto, se mostrará de forma textual.

Una vez que finaliza el proceso de especificación, se puede pasar a la detección de violaciones de código, la cual, como es lógico, se realiza basándose en las reglas anteriormente especificadas. Para ello, basta con hacer clic con el botón secundario del ratón en el proyecto que se desea analizar y seleccionar: PMD → Check Code With PMD. Tras unos segundos efectuando el análisis, se abre una pestaña en Eclipse (como la que se ha podido observar en la *figura 43*) en la que se muestran los resultados del análisis.

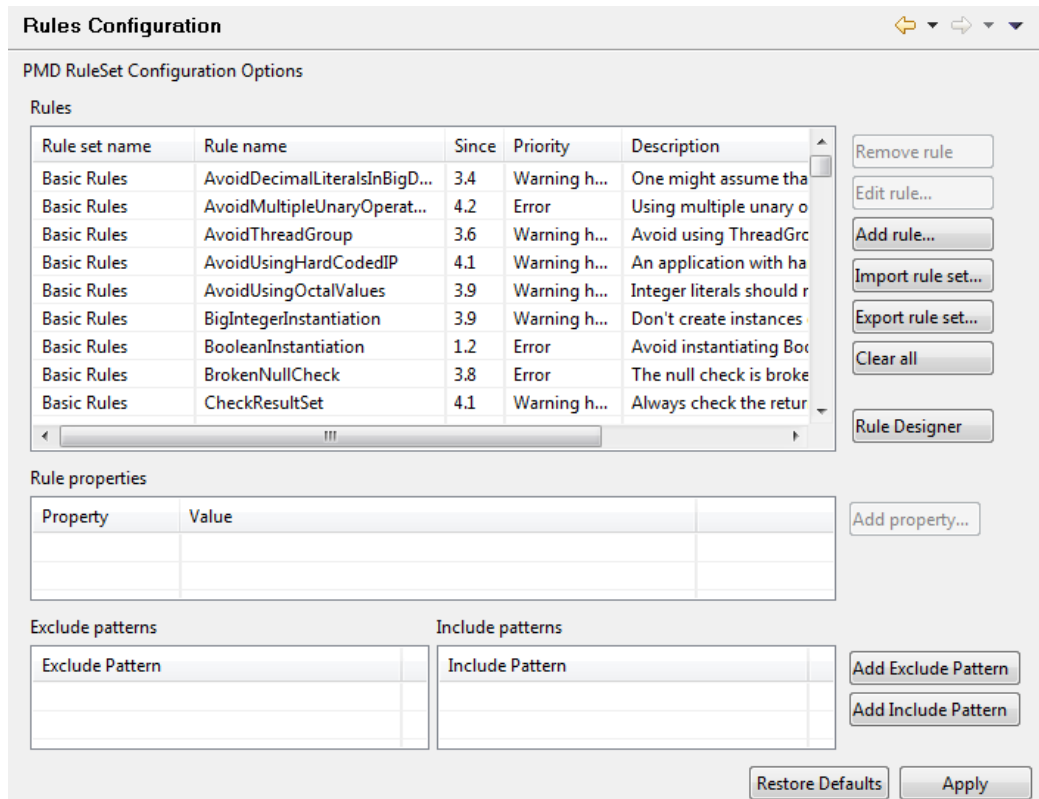


Figura 44. PMD. Opción de Especificación donde se ven las diversas posibilidades.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
PMD	Especificación	Process Description	Interactive	Detection Rules	Textual
			Manual	Detection Programs	Textual
	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Rules-Based	Interactive	Entity-smells relationships	Textual / Visual

Tabla 45. PMD. Tabla de la característica Activity.

Reek

3.2.13. Reek

a. Propósito

Reek [20] es una herramienta desarrollada por Kevin Rutherford que examina clases, módulos y métodos para informar de posibles *smells* que pueda contener el código analizado. La herramienta trabaja con código escrito en Ruby.

b. Información para el análisis

Versiones disponibles:

- Existen varias versiones disponibles en la web <http://github.com/kevinrutherford/reek> en el apartado *switch tags*, donde se puede descargar desde la más antigua, la 0.3.0, hasta la más actual a fecha de realizar esta memoria, la 1.2.8.

Versión analizada: Se realiza el análisis sobre la versión 1.2.8

Sistema Operativo requerido:

- Funciona únicamente en Linux.

Instalación:

Para el correcto funcionamiento de la herramienta se deberá tener actualizado el paquete de mejoras para el manejo de código fuente escrito en Ruby, que es el lenguaje que Reek es capaz de analizar. Dado que se ha tenido que efectuar la instalación de este paquete, de Ruby y de la propia herramienta para llevar a cabo el análisis, a continuación se detallan estas instalaciones.

- Instalación de Ruby:

Se va a instalar desde la línea de comandos la versión disponible en código fuente de Ruby, ya que la versión que está en los repositorios de Ubuntu no incluye el ejecutable para instalar las gemas e instalar el paquete `gem1.9`, lo que puede traer problemas.

Se comienza instalando las dependencias necesarias para compilar Ruby 1.9:

sudo apt-get build-dep ruby1.9

Se descarga el código fuente de la última versión de Ruby 1.9 (al momento de escribir esta memoria era Ruby 1.9.1-p0) y se descomprime en la localización que se desee, se abre un terminal, se va al directorio donde se encuentran los archivos que han sido descomprimidos y se teclea:

./configure --program-suffix=19

Para compilarlo e instalarlo se debe introducir la siguiente línea:

make && sudo make install

De esta manera se tendrá Ruby 1.9 instalado en el sistema, pero como ruby19. Para acceder a él de manera más sencilla, como Ruby, lo que se puede hacer aquí es crear unos enlaces simbólicos (*symlinks*), como por ejemplo:

```
sudo ln -fs /usr/local/bin/ruby19 /usr/local/bin/ruby
sudo ln -fs /usr/local/bin/gem19 /usr/local/bin/gem
sudo ln -fs /usr/local/bin/irb19 /usr/local/bin/irb
```

- Instalación de Reek:

Una vez instalado el intérprete Ruby, se procede a la instalación de la herramienta en cuestión. Reek examina clases, módulos y métodos, e informa de los posibles *bad smells* encontrados. Para la instalación en Linux se deben seguir los siguientes comandos:

```
gem install reek
```

Una vez instalado, se ejecuta de la siguiente forma:

```
reek [options] [dir_or_source_file]*
```

Para obtener información sobre las opciones de Reek, útiles para el manejo de la herramienta, se puede ejecutar el comando de ayuda:

```
reek --help
```

Código utilizado para el análisis:

- Carpeta Ruby source code, incluida en el cd que acompaña a la memoria.

c. Análisis

c.1. Target Artefact:

Reek funciona como herramienta independiente bajo plataforma Linux. Solamente es capaz de analizar código fuente escrito en Ruby, lo que la hace más exclusiva y particular con respecto al resto de herramientas que se analizan en este documento. No admite versiones que permitan comparar dos proyectos diferentes del mismo código. De su tipo de representación, hay que destacar que trabaja con un AST que le permite acceder a los módulos de manera más simple para poder realizar el análisis.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Reek	Independiente (Linux)	Source Code (Ruby)	NO	AST

Tabla 46. Reek. Tabla de la característica Target Artefact.

c.2. Design Smells:

Reek es capaz de reconocer dos tipos de errores de los descritos en el listado de *design smells* que se está utilizando, como son *bad smells/disharmonies* y *bad coding style*. El primero de ellos

engloba a varios de los *smells* catalogados y que Reek es capaz de detectar como son: *Feature Envy*, *Utility Function*, *Nested Iterators*, *Control Couple*, *Large Class*, *Long Method*, *Data Clump*, *Code Duplication* y *Long Parameter List*. También existen otros tipos de *bad smells* (Catalogados así por su autor) que no están incluidos en [5] y en el listado usado y que son detectados por Reek: *Attribute*, *Class Variable*, *Irresponsible Module* y *Simulated Polymorphism*. De todos estos *smells* nombrados se puede encontrar algo de información en el apartado *code smells* de la web dedicada a Reek, <http://github.com/kevinrutherford/reek/wiki/code-smells>.

El otro tipo de *smell* que la herramienta es capaz de detectar es lo que se denomina *Uncommunicative Name*, error incluido dentro de los *bad coding style*, por estar basado en normas o estándares de código.

En la figura que se muestra a continuación se puede observar como Reek rastrea un directorio en busca de código escrito en Ruby (.rb) para ser analizado, mostrando en la línea de comandos los problemas detectados.

```
Archivo Editar Ver Terminal Ayuda
Redmine::Hook::ManagerTest::TestHook3#view layouts base html head refers to context more than self (LowCohesion)
Redmine::Hook::ManagerTest::TestHookHelperController has no descriptive comment (IrresponsibleModule)
Redmine::Hook::ManagerTest::TestHookHelperView has no descriptive comment (IrresponsibleModule)
Redmine::Hook::ManagerTest::TestLinkToHook has no descriptive comment (IrresponsibleModule)
^CError:
mendol0@ubuntu:~$ ^C
mendol0@ubuntu:~$ reek /home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/boot.rb -- 7 warnings:
  Rails#preinitialize calls preinitializer_path twice (Duplication)
  Rails::Boot has no descriptive comment (IrresponsibleModule)
  Rails::GemBoot has no descriptive comment (IrresponsibleModule)
  Rails::GemBoot#load rubygems calls exit(1) twice (Duplication)
  Rails::GemBoot#load rubygems calls rubygems_version twice (Duplication)
  Rails::GemBoot#load rubygems has approx 6 statements (LongMethod)
  Rails::VendorBoot has no descriptive comment (IrresponsibleModule)
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environment.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/demo.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/development.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/production.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/test.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/test_pgsql.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/environments/test_sqlite3.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/10-patches.rb -- 8 warnings:
  ActionView::Helpers::DateHelper#distance of date in words doesn't depend on instance state (LowCohesion)
  ActionView::Helpers::DateHelper#distance of date in words refers to distance in days more than self (LowCohesion)
  ActionView::Helpers::DateHelper#distance of date in words refers to locale more than self (LowCohesion)
  ActionView::Helpers::DateHelper#distance of date in words refers to to_date more than self (LowCohesion)
  ActiveRecord::Base has no descriptive comment (IrresponsibleModule)
  ActiveRecord::Errors has no descriptive comment (IrresponsibleModule)
  ActiveRecord::Errors#full messages contains iterators nested 4 deep (NestedIterators)
  ActiveRecord::Errors#full messages has approx 8 statements (LongMethod)
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/20-mime_types.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/30-redmine.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/40-email.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/backtrace_silencers.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/bigdecimal-segfault-fix.rb -- 1 warning:
  BigDecimal has the name 'BigDecimal' (UncommunicativeName)
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/inflections.rb -- 0 warnings
/home/mendol0/Escritorio/CodigoRuby/redmine-0.9.4/config/routes.rb -- 0 warnings
```

Figura 45. Reek. Muestra de *smells* detectados en un directorio con archivos escritos en Ruby.

Al tratarse de una herramienta que analiza código fuente Ruby, no se puede hablar de una granularidad con una jerarquía de tipo Java como la que se está siguiendo en las tablas que componen esta memoria, pero si se puede decir que el alcance del análisis que Reek efectúa, como se ha comentado en el propósito y como su propio autor indica en la web de la herramienta, es hasta nivel de clase, debido a los tipos de error que detecta y a que se conoce que Ruby dispone al menos de los tipos de jerarquía Clase y Método. No se entra a valorar Sistema, Subsistema y Paquete ya que se desconoce si el código Ruby implementa estas jerarquías y además, como se ha comentado, los tipos de error detectados llegan hasta nivel de clase, por lo que en caso de existir se

deberían marcar con un “NO”. Algo similar ocurre con el tipo de relación que afecta a la detección de *smells*.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Reek	Bad Smells/Disharmonies	-	-	-	SI	SI	-	-	NO
	Bad Coding Style	-	-	-	SI	SI	-	-	

Tabla 47. Reek. Tabla de la característica Design Smells.

c.3. Activity

En este apartado Reek tiene dos tipos de actividades, el análisis y la detección. El primero de ellos viene dado por la transformación del modelo a un árbol sintáctico abstracto (AST) con el que la herramienta es capaz de realizar sus operaciones de forma más eficiente. Esta actividad se realiza de forma totalmente automática una vez que se ejecuta Reek para analizar cualquier proyecto escrito en código Ruby.

La detección es la actividad principal de la herramienta. Muestra por pantalla qué error se ha producido dentro de un determinado archivo o directorio, indicando el método donde se ha producido. Este proceso se realiza de forma automática una vez se ha ejecutado la herramienta, indicando previamente (en la línea de comando, junto a la ejecución de Reek) el archivo o directorio que se quiere examinar. Esta actividad está basada tanto en métricas como en reglas, dependiendo del tipo de *smell* que se localiza.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Reek	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Metrics/Rules-Based	Fully-automated	Entity-smells relationships	Textual

Tabla 48. Reek. Tabla de la característica Activity.

RevJava

3.2.14. RevJava

a. Propósito

RevJava [24] es una herramienta que funciona con código compilado Java y comprueba si los sistemas software están realizados conformes a unas normas de diseño específicas. Su meta-modelo define todos los conceptos relevantes de un sistema software Java (paquete, clase o método) y las asociaciones entre ellos (definición de herencia, llamada al método o variable de acceso).

“Esta herramienta presta soporte para múltiples lenguajes de programación. El meta-modelo está diseñado para modelar programas en Java, pero es lo suficientemente general como para captar los conceptos básicos de sistemas software escritos en otros lenguajes de programación basados en clases orientadas a objetos” [24].

b. Información para el análisis

Versiones disponibles:

- La última versión disponible en su web oficial es la 0.8.5. En el momento de realizar la memoria, la descarga de la aplicación ha dejado de estar disponible.

Versión analizada: RevJava versión 0.8.5.

Sistema Operativo requerido:

- Es una herramienta independiente con instalación para Windows.

Instalación:

- Ejecución del archivo .exe descargado del sitio web oficial. Durante la instalación se elige la ubicación de la herramienta ya instalada, donde estará ubicado el archivo ejecutable de la aplicación, *revjava.bat*.

Código utilizado para el análisis:

- Proyecto 09.

c. Análisis

El análisis de RevJava se realiza sobre la última versión disponible. En el momento de realizar esta memoria, al comprobar de nuevo los links de descarga se ha observado que estos no están disponibles. Se desconoce si volverá a estar disponible una nueva versión, ya que la 0.8.5 data de febrero de 2003 por lo que parece que la herramienta ha sido abandonada.

c.1. Target Artefact:

RevJava es una aplicación independiente que no depende de ninguna otra herramienta para su funcionamiento. Analiza código Java ejecutable (java *bytecode*). No compara diferentes versiones de un mismo proyecto, por lo que no admite versiones.

El código ejecutable introducido en la herramienta para su análisis es transformado en un tipo de representación modelada por un AST, en el que se basará el funcionamiento del análisis de RevJava para detectar los posibles *smells*.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
RevJava	Independiente	Executable Code (Java <i>bytecode</i>)	NO	AST

Tabla 49. RevJava. Tabla de la característica Target Artefact.

c.2. Design Smells:

RevJava es capaz de tratar tres tipos de errores de los que se están buscando en esta memoria. Para reconocer estos *smells* se debe cargar de forma correcta el archivo java *bytecode*. Para ello una vez abierta la aplicación, en la barra de menús dentro de *file* se debe seleccionar (*re*)load clases. Se desplegará una ventana donde se tiene la opción de buscar el archivo java *bytecode* del proyecto que se desee analizar con RevJava.

Una vez abierto, se muestra en la ventana principal una relación de los *critics* que el código contiene. *Critics* es como el programa denomina a los posibles tipos de error que reconoce. En la figura 46 se puede observar el número de *critics* que detecta del java *bytecode* correspondiente al código de Proyecto 09.

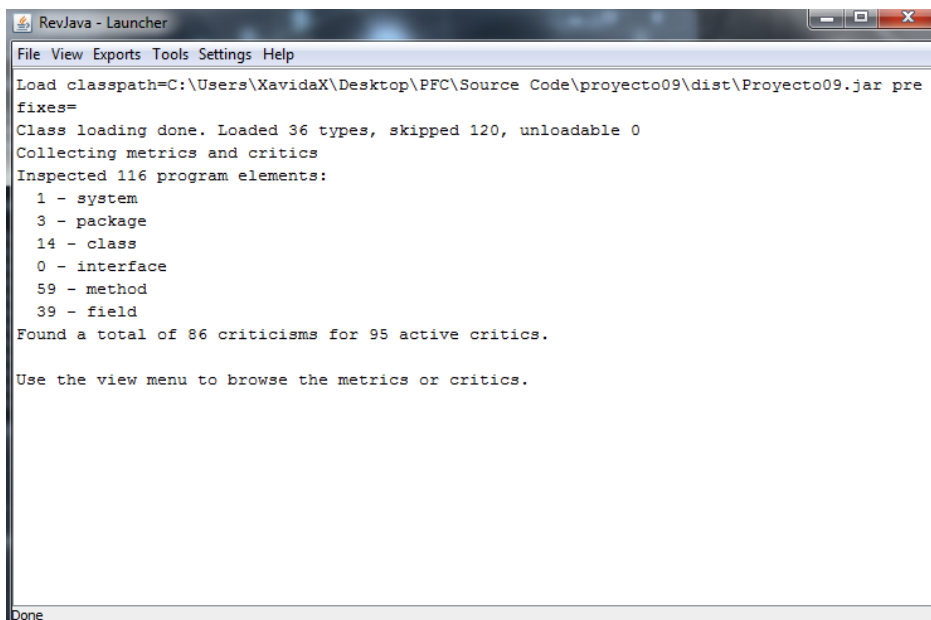


Figura 46. RevJava. Ventana principal con el código de proyecto09 cargado.

Para conocer que tipo de *critics* tiene el código a analizar y dónde se producen, el usuario debe desplazarse nuevamente hasta la barra de menus. En *View*, se selecciona *critics*, y se desplegará una nueva ventana. Se puede observar que RevJava es capaz de detectar *smells* en diferentes niveles (*system*, *package*, *class*, *interface*, *method* y *field*), los cuales se muestran en la

ventana de *critics*. Estos niveles indican la granularidad de la herramienta. Si se selecciona cualquiera de los niveles que se han mencionado anteriormente se desplegará en el apartado de *critics* la lista completa de posibles *smells* encontrados a ese nivel. Si existe algún caso de *bad smell* en la lista facilitada por la entidad, se marcará a su lado con un número que indicará la cantidad de errores producidos del mismo tipo. Para el nivel de sistema, se encuentra un *bad coding style* relacionado con problemas existentes en la entrada y salida, y en las conexiones con la base de datos. El nivel de paquete se centrará en detectar *smells* relacionados con problemas de dependencias entre ellos, como pueden ser dependencias cíclicas o la alta ocultación. Para el resto de niveles los tipos de error que RevJava detecta variarán en *bad coding style*, *bad pattern usage* (*Illegal instantiated singleton*, *incomplete singleton*,...) y *Bad smell / disharmonies* (*God class*).

En la figura siguiente se observa cómo es la interfaz de la ventana *critics*, donde se podrán visualizar algunos de los aspectos comentados anteriormente.

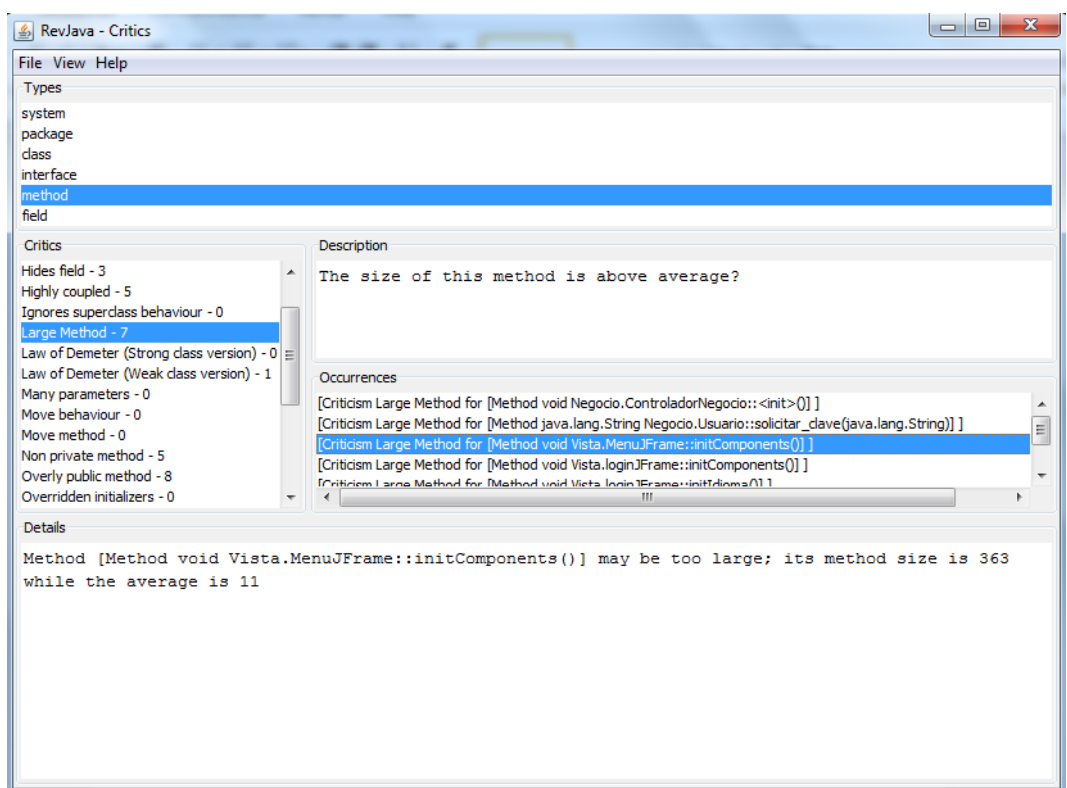


Figura 47. RevJava. Ventana Critics.

A pesar de que la aplicación posee una herramienta para el cálculo de métricas, no se puede hablar del tipo de error *metrics warnings* debido a que no facilitan ningún tipo de advertencias métricas, no existen valores marcados como “peligrosos” que ayuden a la buena mantenibilidad del código analizado.

Existen relaciones intra e inter entre las diferentes entidades que RevJava es capaz de analizar. En cuanto al aspecto de la herencia, la aplicación realiza un análisis a nivel de clase denominado

Collection inheritance que comprueba y detecta si existe un posible problema de comportamiento en las colecciones de herencia al ser invocadas.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
RevJava	Bad Coding Style								
	Bad Pattern Usage	SI	NO	SI	SI	SI	SI	SI	SI
	Bad Smells / Disharmonies								

Tabla 50. RevJava. Tabla de la característica Design Smells.

c.3. Activity:

A RevJava se le reconocen dos tipos de actividades. La primera de ellas es el análisis. Como ya se ha dicho con anterioridad, esta herramienta realiza un trabajo interno que transfiere el código del proyecto a analizar por la herramienta a un analizador de código que a su vez genera un AST, con el que RevJava se ayudará para realizar las operaciones que ofrece al usuario.

El otro tipo de actividad que posee esta herramienta es la detección de *smells*. Como se ha visto en la sección *design smells*, RevJava es capaz de encontrar tres tipos de errores incluidos en el estudio que venimos realizando: *bad coding style*, *bad pattern usage* y *antipatterns / disharmonies*. Para llevar a cabo la detección de estos *smells* la herramienta aplica técnicas basadas en métricas y reglas, calculadas por ella misma, para detectar posibles casos de *bad coding style* y técnicas basadas en métricas y patrones para el resto de posibles *smells* mencionados anteriormente.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
RevJava	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Metrics/Rules/Pattern-Based	Interactive	Entity-smells relationships	Textual

Tabla 51. RevJava. Tabla de la característica Activity.

Roodi

3.2.15. Roodi

a. Propósito

Roodi, “Ruby Object Oriented Design Inferometer”, se encuentra disponible para el lenguaje Ruby. Analiza el código Ruby y advierte sobre los problemas de diseño basándose en los controles que se han configurado al inicio de su ejecución (Tiene valores predeterminados por defecto en caso de que no se configure).

b. Información para el análisis

Versiones disponibles:

- Únicamente está disponible para su descarga o instalación la versión 2.0.1.

Versión analizada: Roodi 2.0.1.

Sistema Operativo requerido:

- Linux. Ha sido probado en Ubuntu 10.

Instalación:

- Su instalación desde cero es como la de Reek por lo que el primer paso sería tener instalado el interprete Ruby. Ahora el comando de instalación de Roodi es ***gem install roodi***.

Una vez instalado, se ejecutará de la siguiente forma

roodi [options] [dir_or_source_file]*

Para obtener mayor información sobre las opciones de Roodi se puede introducir el comando de ayuda para la herramienta, **roodi -help**.

Código utilizado para el análisis:

- Se ha probado con todo el código contenido en la carpeta “Ruby source code”.

c. Análisis

c.1. Target Artefact:

Como ya se ha visto en el propósito de la herramienta, Roodi se encarga de analizar código fuente Ruby. Una de las características más interesantes de Roodi es que depende de JRuby debido a que utiliza las bibliotecas JRuby AST para analizar el código fuente de Ruby, creando un árbol AST [70]. La herramienta no tiene ninguna opción para comparar diferentes versiones, únicamente analiza el código que se la introduce por línea de comandos y trata todo por igual. Es posible introducir varios archivos .rb a la vez mediante el uso de asteriscos como en Unix, como por ejemplo puede ser “ruta_del_archivo/*.rb” o “ruta_del_archivo/**/*.rb”.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Roodi	Independiente	Source Code (Ruby)	NO	AST

Tabla 52. Roodi. Tabla de la característica Target Artefact.

c.2. Design Smells:

Como se puede ver en el documento [71], creado por el propio desarrollador de la herramienta, Roodi realiza la búsqueda de *smells* mediante métodos como *Class Line Count*, *Method Line Count*, *Module Line Count*, *Block Cyclomatic Complexity* y *Method Cyclomatic Complexity*, los cuales se engloban como *Metrics Warnings* ya que Roodi advierte de que se sobrepasa lo establecido como mínimo para cada uno de ellos, y métodos como *Class Name*, *Method Name*, *Module Name*, *Assignment In Conditional*, *Case Missing Else*, *Empty Rescue Body*, *For Loop Check* y *Parameter Number* que se engloban dentro del *Bad Coding Style*. Un ejemplo de cómo actúa Roodi se puede encontrar en la figura incluida a continuación. En ella se ve, entre otros, como muestra los errores denominados *Metrics Warnings*.

```

from /usr/local/bin/roodi:19:in <main>
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/helper_testcase.rb

Found 0 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/inflections.rb

Found 0 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/10-patches.rb
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/config/initializers/10-patches.rb:17 - Method "full_messages" has 23 lines. It should have 20 or less.

Found 1 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/config/boot.rb
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/config/boot.rb:59 - Found = in conditional. It should probably be an ==

Found 1 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/unit/issue_test.rb
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/unit/issue_test.rb:20 - Class "IssueTest" has 597 lines. It should have 300 or less.
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/unit/issue_test.rb:43 - Method "test_create_with_required_custom_field" has 21 lines. It should have 20 or less.
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/unit/issue_test.rb:232 - Method "test_should_close_duplicates" has 24 lines. It should have 20 or less.

Found 3 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/unit/custom_value_test.rb

Found 0 errors.
mendo10@ubuntu:~$ roodi /home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/integration/account_test.rb
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/integration/account_test.rb:24 - Rescue block should not be empty.
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/integration/account_test.rb:40 - Method "test_autologin" has 29 lines. It should have 20 or less.
/home/mendo10/Escritorio/CodigoRuby/redmine-0.9.4/test/integration/account_test.rb:72 - Method "test_lost_password" has 24 lines. It should have 20 or less.

Found 3 errors.
mendo10@ubuntu:~$

```

Figura 48. Roodi. Ejecución de Roodi con varios códigos diferentes.

Al igual que sucedía con Reek, no se puede establecer la granularidad debido al desconocimiento del lenguaje Ruby pero por las búsquedas de posibles problemas en código que realiza lleva a pensar que existe comprobación de *smells* a nivel de clase y método.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Roodi	Metrics Warning	-	-	-	SI	SI	-	-	-
	Bad Coding Style	-	-	-	SI	SI	-	-	-

Tabla 53. Roodi. Tabla de la característica Design Smells.

c.3. Activity:

Una de las grandes diferencias que se encuentran con Reek en cuanto a la característica Actividad es que Roodi permite realizar una especificación relativa a las búsquedas. Ésta se realiza de manera totalmente manual y, como se indica en la web oficial de la herramienta, se realiza de la siguiente manera:

“Para cambiar el conjunto de comprobaciones incluido, o para cambiar los valores por defecto, puede proporcionarse su propio archivo de configuración. El archivo de configuración es un archivo YAML con las listas de los controles que deben incluirse. Cada comprobación puede incluir opcionalmente un hash de opciones.”

El aspecto que por defecto tiene el archivo de configuración se puede comprobar en la propia web, así como una descripción de cada una de las opciones de comprobación incluidos.

Otro tipo de actividad que Roodi incluye, como es habitual en las herramientas que tienen como representación interna un AST, es la de Análisis. Ésta fase se realiza de manera automática en cuanto se le facilita a la herramienta el código a analizar. Lo mismo sucede con la actividad de Detección, la cual se basa en métricas estructurales para mostrar los errores que se han incluido en *Metrics Warnings* y en un conjunto de reglas de diseño para los de tipo *Bad Coding Style*. El resultado se muestra de manera textual como se ha podido observar en la *figura 48*.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Roodi	Especificación	Process Description	Manual	Detection Rules	-
	Análisis	Model Description	Fully-automated	-	-
	Detección	Metrics-Based (Structural Metrics) Rule-Based	Fully-automated	Entity-metrics relationships	Textual

Tabla 54. Roodi. Tabla de la característica Activity.

STAN

3.2.16. STAN

a. Propósito

Stan [64] es una herramienta diseñada para realizar análisis de la estructura de código Java, basada en Eclipse. Stan facilita la visualización del diseño, la compresión del código, la medición de la calidad y la presentación de informes de defectos de diseño. La herramienta incluye un conjunto de indicadores, adecuados para cubrir la mayoría de los aspectos importantes de la calidad estructural. Se pone atención especial en el análisis de dependencias, que es clave para el análisis de la estructura.

b. Información para el análisis

Versiones disponibles:

- En el momento de realizar la memoria existen diferentes versiones. La versión plugin de Stan para integrarla en Eclipse y la versión independiente. También existe una versión híbrida que incluye las dos anteriores. Las versiones que se han encontrado y revisado son todas demostraciones que tienen una licencia de prueba de 30 días.

Versión analizada: Se analiza la versión independiente de Stan, más concretamente la 2.0.

Sistema Operativo requerido:

- La herramienta se encuentra disponible para ser instalada en cualquiera de estas tres plataformas: Windows (XP/Vista/7), Mac OSX (Tiger/Leopard) y Linux.

Instalación:

- Como plugin de Eclipse con la url <http://update.stan4j.com/ide>

La versión independiente se puede obtener de la web oficial de Stan en su sección de descargas. Una vez descargada basta con extraer la carpeta que se encuentra en el interior del archivo comprimido, donde se encuentra un archivo ejecutable para lanzar la aplicación.

Código utilizado para el análisis:

- jHotdraw 5.2.

c. Análisis

Se analiza la versión independiente de Stan debido a que todas las perspectivas de análisis se inician en el momento de su ejecución, mientras que en eclipse hay que cargarlas. No existen diferencias entre las dos versiones, por lo que la utilización de una u otra será decisión del usuario.

c.1. Target Artefact:

Como se ha visto en el apartado anterior, Stan está disponible como una aplicación independiente o como plugin para ser integrado en Eclipse. Stan proporciona un análisis para estructuras Java, utilizando para el mismo java *bytecode* en lugar del código fuente.

Stan no admite versiones, puesto que carece de una opción que permita comparar versiones distintas de un mismo proyecto para comprobar la evolución del código y principalmente de los *bad smells*.

Al analizar el java *bytecode*, Stan recoge toda la información necesaria para construir de forma detallada un modelo de la estructura de la aplicación que será su tipo de representación. A partir de dicho modelo la herramienta trabajará para la detección de los posibles *smells* que contenga el código introducido para su análisis [67]. No se conoce el tipo de representación usado.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
STAN	Independiente/Plugin Eclipse	Executable Code (Java <i>bytecode</i>)	NO	-

Tabla 55. STAN. Tabla de la característica Target Artefact.

c.2. Design Smells:

Stan presenta la detección de *metrics warnings* como uno de los principales tipos de error obtenidos en un análisis. Para ello, una vez cargados los archivos *.jar* o las carpetas que los contienen, Stan realiza un chequeo del código para hacer una evaluación de las métricas y comprobar si los valores establecidos son superados, y por tanto se marcan como *metrics warnings*. Las métricas analizadas por Stan se dividen en cuatro grupos: *count*, *complexity*, métricas de Robert C. Martin y métricas de Chidamber y Kemerer. Estas dos últimas son métricas incorporadas a Stan para dar una mayor información a nivel de paquete en el caso de C. Martin y en el de Chidamber y Kemerer a nivel de clase e inferiores, mostrando advertencias en valores correspondientes, por ejemplo, a la profundidad de la herencia. En la figura 49 se puede observar la interfaz de Stan mostrando en este caso un *metrics warning* correspondiente a las métricas de Chidamber y Kemerer. El gráfico superior de la derecha mostrará el punto donde supera el límite establecido para esa métrica. El gráfico de la izquierda muestra la clase donde se ha cometido la violación métrica, donde se muestran también las dependencias entre los métodos en el interior de la clase.

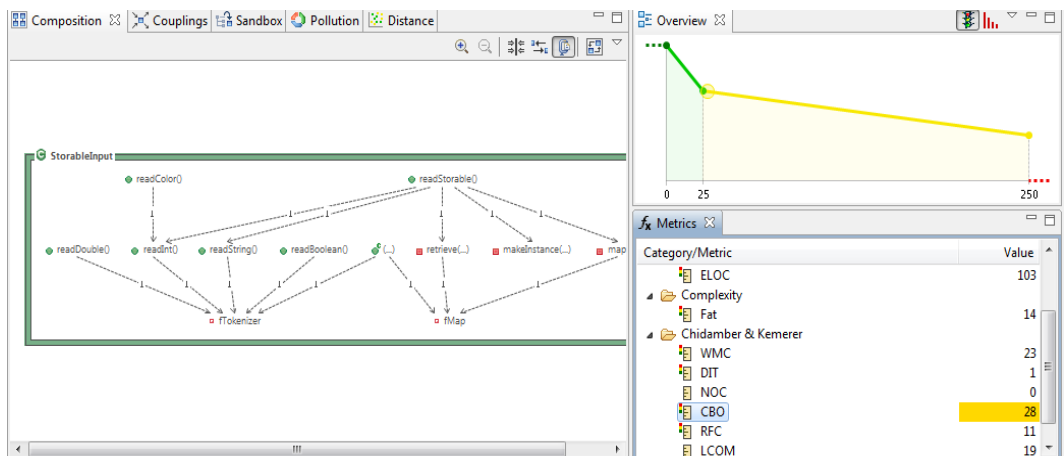


Figura 49. STAN. Metrics Warnings en Stan.

Stan proporciona, como se puede ver en la figura anterior, un gráfico de dependencias que muestra las relaciones existentes entre métodos, clases, etc. Una de las relaciones entre nodos que se marcan de forma distinta es el “*tangle*”, que viene a ser una dependencia que hace formar un ciclo entre dos nodos. Esta dependencia cíclica viene marcada por Stan en color rojo, de forma que sea fácilmente identificable por el usuario el lugar donde se encuentra este *smell*, que se denomina *dependency graph*. El *tangle* también se podrá identificar como *metrics warnings* si es que supera el valor establecido como límite dentro de los parámetros elegidos. En la siguiente figura se verá como Stan marca el ciclo en una zona de la estructura de la aplicación jHotdraw 5.2.

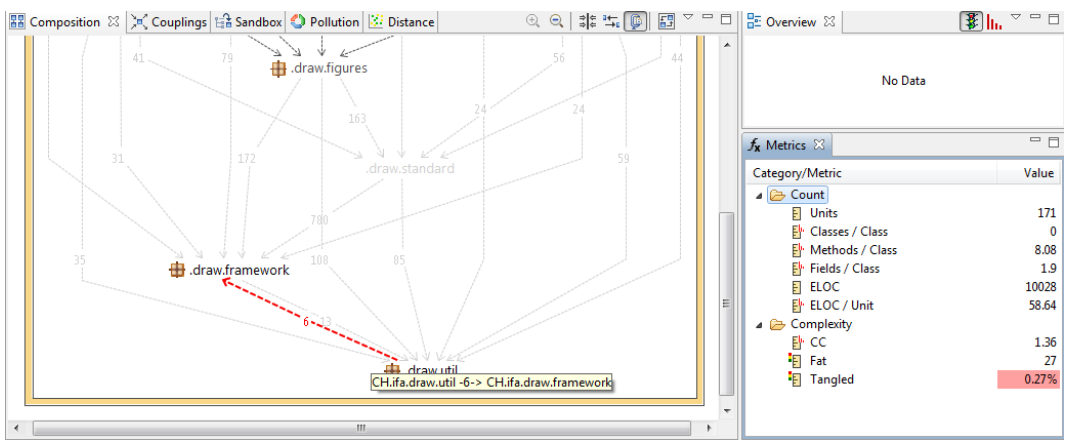


Figura 50. STAN. Visualización del *tangle*.

La granularidad de la herramienta viene marcada por el nivel de entidad que alcanzan *metrics warnings* y *dependency graph*, que en el caso de Stan llega hasta el nivel más alto que es el de sistema.

Existen relaciones inter e intra. Se hace tratamiento de la herencia ya que se muestran *metrics warnings* relacionados con su profundidad.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
STAN	Metrics Warnings	SI	SI	SI	SI	SI	SI	SI	SI
	Dependency Graph	SI	SI	SI	SI	SI	SI	SI	

Tabla 56. STAN. Tabla de la característica Design Smells.

c.3. Activity:

Stan presenta tres tipos de actividades incluidas dentro de la clasificación mostrada en la figura 8. La primera de ellas es la especificación. Como ya se ha explicado, esta actividad permite al usuario introducir o modificar nuevas reglas para la posterior detección de *smells*. En el caso de Stan, la especificación viene determinada por las diferentes opciones que se pueden dar a las métricas y que van desde la anulación del cálculo de una métrica determinada a la modificación de los parámetros de una de ellas. El usuario puede manipular dichos parámetros en función de cómo

se quieren considerar los resultados de las métricas. Por ejemplo, en la *figura 51* se muestra una modificación de la métrica ELOC, variando el intervalo de color amarillo, el de riesgo (el intervalo rojo es el de error). La modificación de una de las métricas conlleva en la mayoría de los casos la variación de los parámetros de otras métricas que dependan de la modificada.

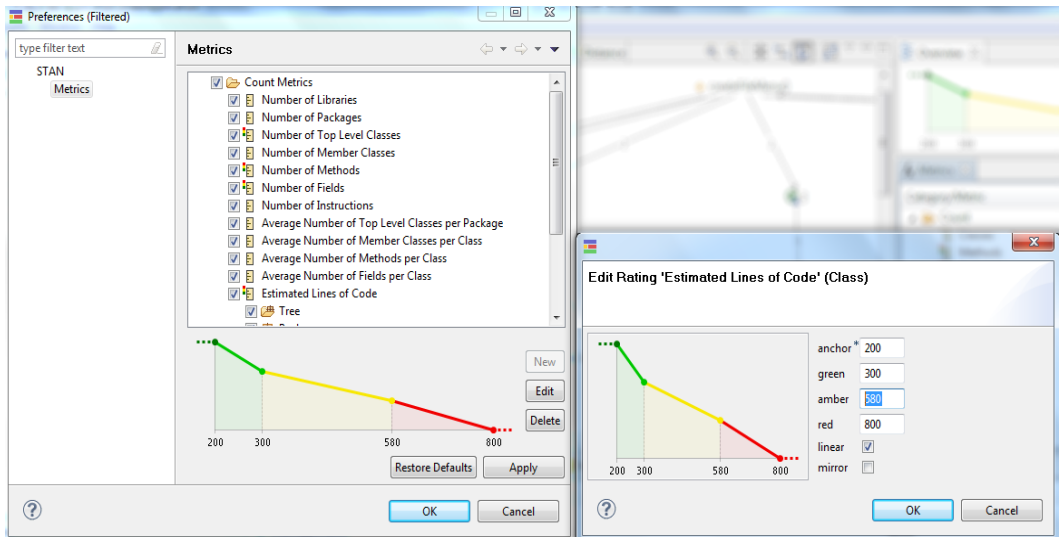


Figura 51. Stan. Proceso de especificación de las métricas.

La detección es la actividad más importante de la herramienta. Muestra los *smells* de forma textual y numérica basándose en la gran cantidad de métricas calculadas. En la figura siguiente se observa cómo se marcan los *metrics warnings*.

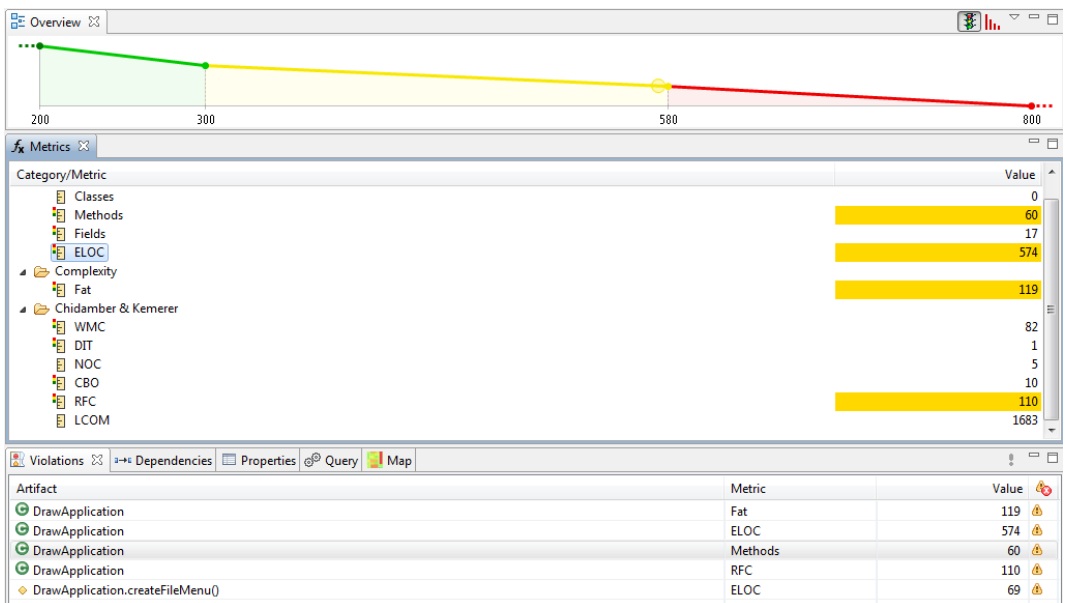


Figura 52. Stan. Muestra de *metrics warnings* detectados.

La última actividad destacada de la herramienta es la de visualización. Está basada en métricas y, como se puede comprobar en las imágenes anteriores, son tanto de tipo gráfico como de tipo diagrama.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
STAN	Especificación	Process description	Interactive / Manual	Detection Metrics	Numerical
	Detección	Metric-Based	Interactive	Entity-metrics relationships	Textual, Numerical
	Visualización	Metric-Based	Interactive	Graphs, Diagrams	Visual

Tabla 57. STAN. Tabla de la característica Activity.

Structural Analysis for Java

3.2.17. Structural Analysis for Java (SA4J)

a. Propósito

Structural Analysis for Java (SA4J) [63], ha sido desarrollado por IBM, y proporciona al usuario un amplio número de funcionalidades como vistas, métricas, análisis e informes. SA4J se basa en el concepto de dependencias y dependientes a cargo de una entidad determinada, ya sea un paquete, una clase o una interfaz. El número de dependencias de una entidad es el número de entidades sujetas a la entidad dependiente. El número de dependientes de una entidad es el número de entidades que dependen de las funcionalidades proporcionadas por la entidad sujeto.

Además de la explotación de puntos de vista que son muy similares a los diagramas de clases UML, y que permiten al usuario comprender la estructura física de los sistemas analizados, SA4J permite la detección de “antipatrones estructurales”, como *butterflies*, *breakable*, *hubs* y *tangles*, la evaluación y el análisis de un buen número de métricas sobre las clases y los paquetes y la evaluación y el análisis de la estabilidad del sistema mediante una vista del esqueleto para determinar qué componentes se basan en la parte superior de los demás y en consecuencia pueden afectar a la estabilidad del sistema.

b. Información para el análisis

Versiones disponibles:

- Independiente. Sólo hay una disponible en la web, la 1.0.

Versión analizada: Versión independiente para Windows, disponible en la web.

Sistema Operativo requerido:

- La herramienta funciona en Linux, Solaris o Windows. Bajo esta última se ha probado con éxito en Windows Vista y Windows 7 aunque es necesario el modo compatibilidad por lo que es una herramienta desarrollada para Windows anteriores (como puede ser XP).

Instalación:

La versión independiente de SA4J se puede obtener de la sección de descargas del sitio web de la herramienta: <http://www.alphaworks.ibm.com/tech/sa4j/download>. El archivo descargado es un ejecutable que se debe lanzar y posteriormente seguir sus instrucciones para completar la instalación.

Código utilizado para el análisis:

- lucene3.

c. Análisis

c.1. Target Artefact:

La única versión disponible de SA4j, es una versión independiente con fecha de publicación marzo de 2004. La herramienta admite como formatos de entrada tanto código fuente, como código ejecutable Java (Java *bytecode*). SA4j permite cargar directorios completos dónde esté ubicado el

proyecto java a analizar. La herramienta se encargará de buscar en dichos directorios los archivos *.class* o los java *bytecode*. No admite versiones y se desconoce si utiliza algún tipo de representación interna reconocido ya que no hay disponible información al respecto.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
SA4J	Independiente	Source Code (Java .class) Executable Code (Java <i>bytecode</i>)	NO	-

Tabla 58. SA4J. Tabla de la característica Target Artefact.

c.2. Design Smells:

SA4j destaca por ofrecer una amplia visión e información de todas las dependencias existentes entre los paquetes y clases del proyecto java que se está analizando. Se considera para ésta herramienta la aparición de *dependency warnings*, un tipo de error que señala de forma gráfica y a través de tablas el número de dependencias, tanto locales como globales, que tiene una determinada clase. Si excede el número de clases dependientes (dentro de unos valores establecidos por SA4j) se marcará la celda donde se especifica con tonos cada vez más oscuros a medida que las dependencias aumentan (ver figura 53).

Object	Include	Dependencies	Affected	Dependents	Affects
resourcemanager	✓	5	109	4	169
DefaultSimilarity	✓	3	190	4	169
SegmentTermDocs	✓	25	190	3	169
SegmentTermPositions	✓	9	190	2	169
CharStream	✓	0	1	14	19
FilteredTermEnum	✓	7	5	14	18
Spans	✓	0	1	31	17
CharReader	✓	3	2	3	16
NumericUtils	✓	0	1	4	16
ComplexExplanation	✓	1	2	4	15
SpanScorer	✓	13	92	4	15
TokenFilter	✓	4	5	10	14
Tokenizer	✓	5	7	11	14
SpanQuery	✓	6	195	43	14
SpanWeight	✓	23	195	2	14
FieldCache	✓	2	192	23	13
MultiTermQuery	✓	4	192	12	12
BitUtil	✓	0	1	3	12
DocValues	✓	2	2	16	12
ValueSource	✓	2	192	6	11
CharTokenizer	✓	7	10	3	10
OpenBitSet	✓	5	6	12	10
OpenBitSetIterator	✓	3	6	2	10
TypeAttribute	✓	1	2	13	9
LetterTokenizer	✓	2	11	2	7
Version	✓	0	1	14	7
TermQuery	✓	7	191	9	7
LowerCaseTokenizer	✓	2	12	3	6
FlagsAttribute	✓	1	2	7	6
PrefixTermEnum	✓	7	192	2	6
FieldCacheSource	✓	5	195	9	6

Figura 53. SA4j. Muestra de dependencias entre las clases.

El otro tipo de error que Stan detecta, es la aparición de mecanismos de detección de *antipatterns*. Debido a que los autores de la herramienta hacen referencia en el sitio web a la detección de “antipatrones estructurales”, y tras consultar en varios documentos como [72] y pese a que no aparecen en el listado inicial de antipatrones utilizado, se consideran *butterflies*, *breakable*, *hubs* y *tangles* como antipatrones. Dentro de la pestaña *summary* se podrá acceder a la información que la herramienta proporciona sobre los antipatrones mencionados anteriormente, dando el número de casos y el porcentaje de objetos afectados por cada patrón.

Explorer

Local Dependencies

Global Dependencies

Skeleton

What If

Summary

?

Structural Patterns

Pattern	Count	% of total objects
<u>Tangle</u> group of interdependent objects	8	n/a
<u>Global Hub</u> often affected when any object in the system changes and then affects a lot of other objects	81	20%
<u>Global Breakable</u> often affected when any object in the system changes	169	41%
<u>Global Butterfly</u> when changed may cause a lot of other objects to change	190	47%
<u>Local Hub</u> immediately depends on a lot of objects and a lot of objects immediately depend on it	26	6%
<u>Local Breakable</u> immediately depends on a lot of objects	87	21%
<u>Local Butterfly</u> a lot of objects immediately depend on it	85	21%

Figura 54. SA4j. Muestra de antipatrones.

Si se hace clic sobre cualquiera de los antipatrones que se muestran, se abrirá un listado de las clases que pueden verse afectadas por la aplicación del antipatrón en cuestión. Si sobre cada clase se hace clic con el botón derecho se desplegará un menú donde se puede explorar visualmente la situación de la clase seleccionada. A través de las pestañas superiores es posible comprobar su análisis de dependencias (ver *figura 55*).

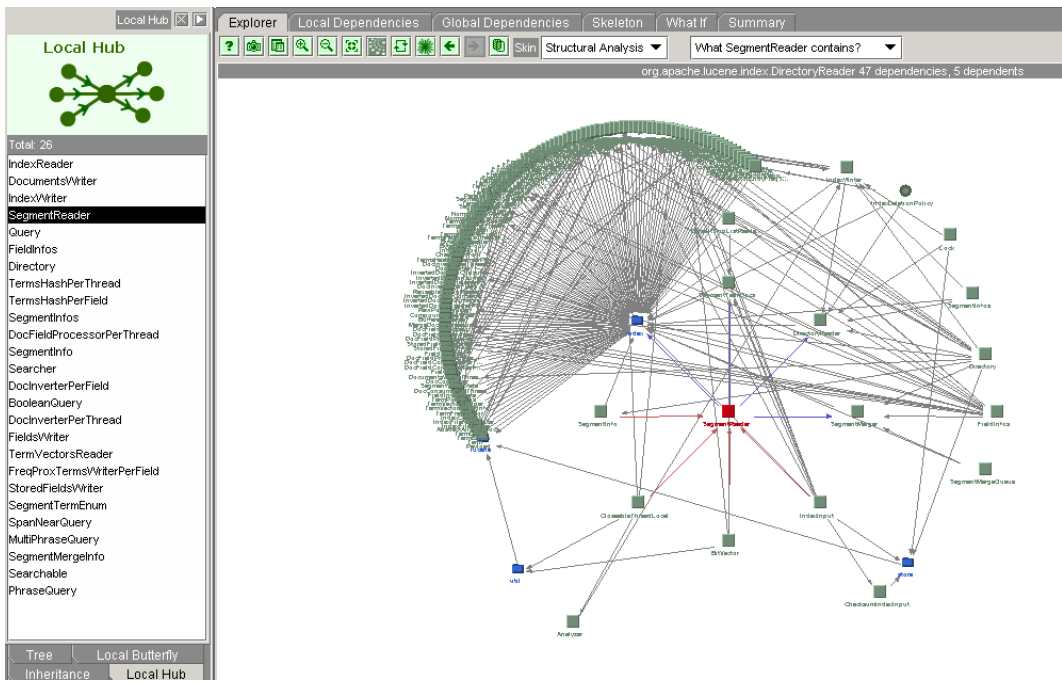


Figura 55. SA4j. Muestra del antipatrón *local hub*.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
SA4J	Dependency Warnings	NO	NO	SI	SI	NO	SI	SI	SI
	Antipatterns								

Tabla 59. SA4j. Tabla de la característica Design Smells.

c.3. Activity:

Se considera que SA4j realiza la actividad de análisis para poder transformar el modelo java introducido en un metamodelo. A pesar de no reconocer su representación interna dada la falta de información existente, esta se antoja clave para que la herramienta pueda llevar a cabo la detección de los antipatrones que SA4j puede encontrar. Dicha detección está basada en métricas y patrones que son calculadas por la herramienta de forma totalmente automática. Otra de las actividades que la herramienta soporta es la visualización, permitiendo observar la multitud de dependencias que pueden llegar a existir entre unas determinadas clases (ver *figura 55*). La herramienta también permite visualizar la estructura mediante una vista denominada *skeleton*, que proporciona otra manera de poder observar las relaciones entre las diferentes clases (ver *figura 56*).

Figura 56. SA4J. Visualización *skeleton*.

Por último, se puede hablar de la actividad análisis de impacto. En la parte superior se puede encontrar junto a las demás funcionalidades de las que ya se han hablado, una pestaña denominada *what if*. Si se accede a esta sección se puede encontrar una división en paquetes del proyecto que se tiene cargado con las clases que tienen contenidas en su interior. El uso que se ha determinado una vez puesto en práctica la aplicación *what if*, es comprobar cómo va trazando de forma gráfica las dependencias entre las clases y paquetes una vez iniciado el trazo de las dependencias desde cualquier punto de la estructura mostrada. Se pueden observar en tiempo de ejecución en una sección a la derecha de la visualización, denominada *potentially affected objects*, los objetos que se han visto afectados con el nuevo trazado propuesto por el usuario.

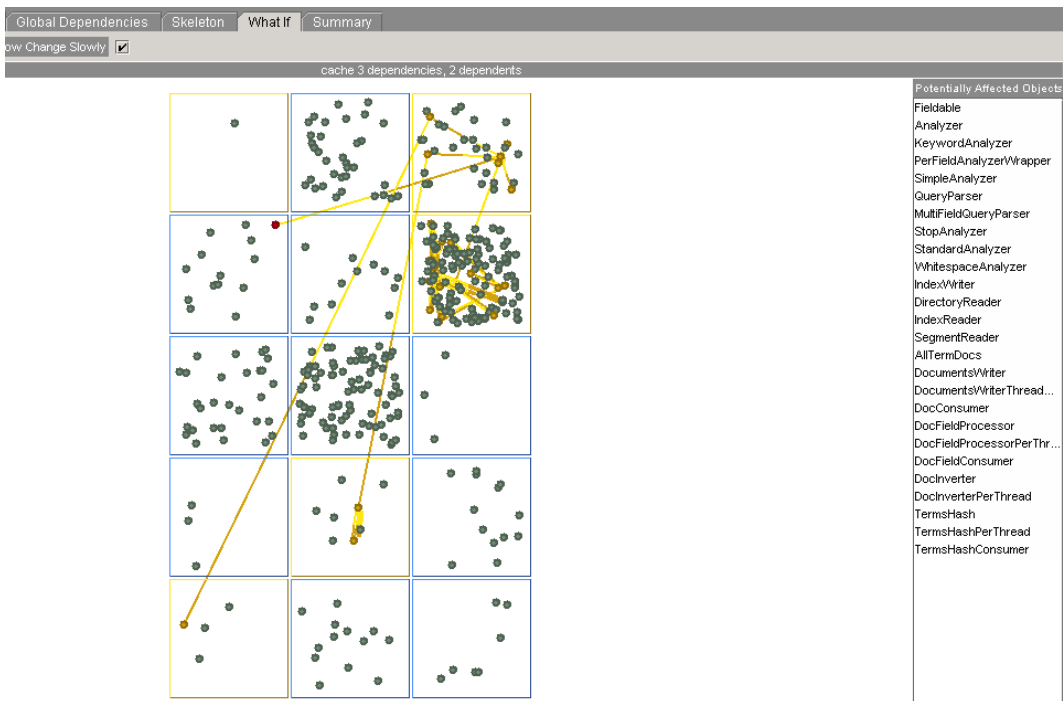


Figura 57. SA4J. Ejecución de *what if*.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
SA4J	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Metrics/Pattern-Based	Fully-automated	Entity smells relationships	Numerical, Visual
	Visualización	Visualisation	Fully-automated	Dependency Diagrams, Skeleton	Visual
	Análisis de Impacto	Visualisation-Based	Interactive	Dependency Diagrams	Visual

Tabla 60. SA4J. Tabla de la característica Activity.

Structure101

3.2.18. Structure101

a. Propósito

El objetivo principal de Structure101 [65] es ayudar al arquitecto de software a entender la estructura actual del sistema software, evaluar su complejidad estructural, planificar cómo debería ser la arquitectura de ese sistema, comunicar al equipo de desarrolladores una arquitectura pretendida y además monitorear la evolución de esa estructura a medida que el sistema se desarrolla.

b. Información para el análisis

Versiones disponibles:

- Independiente. Como herramienta independiente tiene diferentes versiones en función del lenguaje del código que se quiera analizar. Están disponibles versiones para Java, .NET, C/C++ y una versión genérica que no depende del lenguaje. La versión de C/C++ no está disponible directamente en la web oficial de la herramienta [65] sino en la de PRQA [66]. Todas estas versiones son de pago.

- Como plugin de Eclipse e IntelliJ IDEA para Java. Como plugin de Visual Studio para .NET. Son herramientas menos completas que las versiones independientes pero son gratuitas.

- Aplicación web. Al igual que las versiones que se encuentran como plugin, es una herramienta menos completa que las versiones independientes pero también es gratuita.

Versión analizada: Dado que la mayoría de código del que se dispone es Java, se va a analizar una de las versiones que soportan este lenguaje. De entre las disponibles, se analiza la versión independiente para código Java, ya que aunque es una demo (al ser esta versión de pago), es más completa que como plugin de Eclipse.

Sistema Operativo requerido:

- La versión analizada funciona en entornos Windows, Linux y Mac OS X (Esta última no ha sido probada).

Instalación:

- Para la instalación de la versión de prueba de la herramienta independiente para Java se debe descargar el archivo ejecutable que aparece en:

<http://www.headwaysoftware.com/downloads/structure101/java-win.php> y

A continuación se siguen los pasos que se van mostrando al ejecutarlo. Además se debe solicitar una licencia, la cual se envía a la dirección de correo electrónico que se haya indicado. Una vez recibida la licencia, se procederá a la descarga para almacenarla en el directorio de instalación de la herramienta. En caso de que se guarde en otro directorio tampoco hay mayor problema ya que si el programa no lo encuentra, este pide que se le indique donde se ha almacenado para poder realizar una copia en el lugar correcto.

Código utilizado para el análisis:

- jFreechart y CheckStyle5.0, incluidos en el repositorio de versiones de la herramienta.

c. Análisis

Aunque la herramienta principal que se va a analizar es la versión independiente para Java, es necesario manejar y controlar también la versión plugin para Eclipse y la versión web ya que son herramientas complementarias a la versión independiente.

La aplicación web permite efectuar el monitoreo de la arquitectura de uno o varios proyectos. Para cada proyecto, ésta aplicación puede informar al usuario acerca de los cambios arquitectónicos ocurridos en la última versión del sistema, y representar gráficamente el comportamiento de la complejidad estructural y el tamaño del mismo. Además, puede señalarle al usuario cualquier violación de las restricciones arquitectónicas previamente definidas o de los límites establecidos para la complejidad estructural.

El plugin permite que el equipo de desarrolladores vea los diagramas arquitectónicos definidos para la aplicación, como parte de su ambiente de desarrollo. Además les reporta, en la misma forma en la que anuncia los errores de compilación, cualquier violación a las restricciones arquitectónicas definidas. De esta manera controla que la estructura del sistema evolucione de acuerdo a lo planificado por el arquitecto de software. Los desarrolladores también pueden ubicar el código fuente responsable de cualquier dependencia no deseada entre las partes del sistema, o complejidad estructural excesiva.

c.1. Target Artefact:

Structure101 obtiene información del conjunto de archivos fuente y *bytecode* que conforman el proyecto a analizar. Los resultados del análisis estático son almacenados en un espacio específico independiente del área de trabajo del proyecto analizado. Ha sido imposible saber la representación exacta con la que almacena dichos resultados pero es de suponer que sea un *object model*. Esta característica no se ha podido verificar por lo que en la tabla aparecerá con un guión (desconocido).

Una característica destacable de Structure101 es el hecho de que la herramienta admita versiones. Concretamente la herramienta dispone de un repositorio de versiones en el que se almacenan versiones de diferentes proyectos y en el que además se pueden crear nuevos repositorios para nuevos proyectos. Dentro de la suite de productos Structure101, se incluyen dos componentes adicionales a la versión independiente que se está analizando y que aprovechan los datos almacenados en un repositorio como ya se ha dicho anteriormente. Son la aplicación web y el plugin de Eclipse. En la *figura 58* se puede observar la forma en que la aplicación web trabaja con el repositorio de versiones.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Structure101	Independiente/Plugin Eclipse/Plugin IntelliJ/Aplicación web	Source Code Executable Code (Java <i>bytecode</i>)	SI	-

Tabla 61. Structure101. Tabla de la característica Target Artefact.

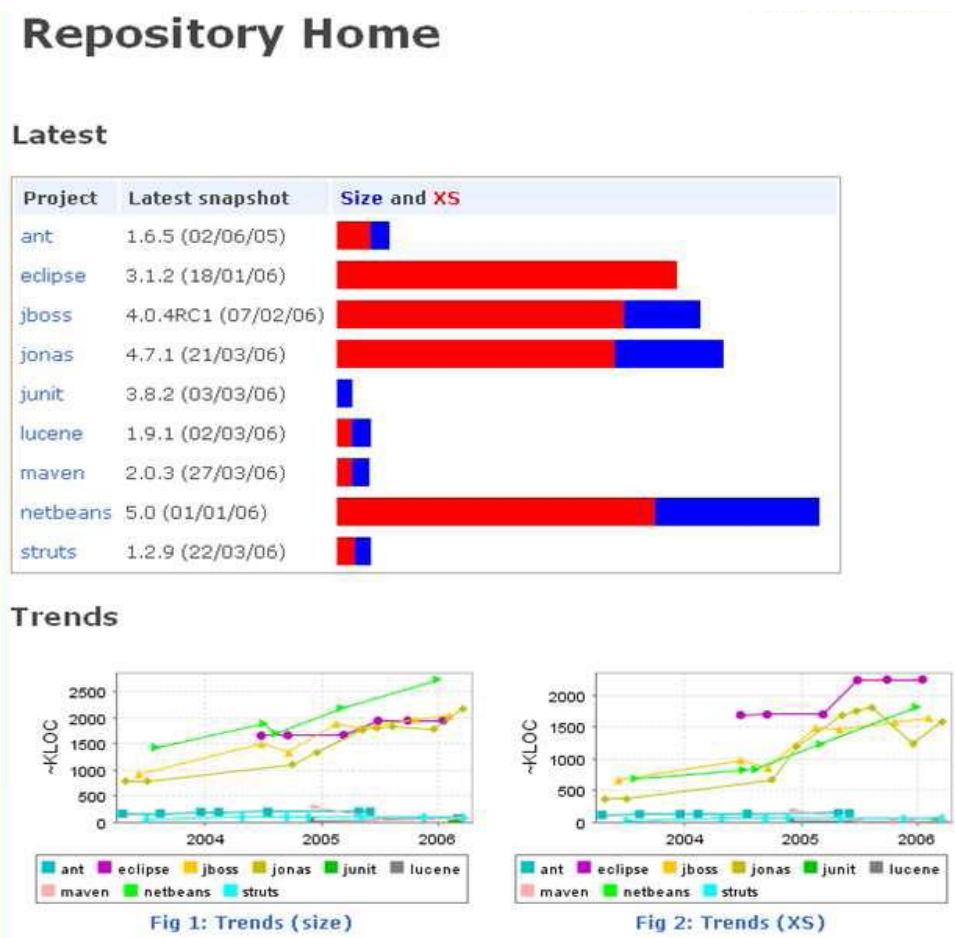


Figura 58. Structure101. Funcionamiento de la aplicación web con un repositorio de versiones.

c.2. Design Smells:

Structure101 es una herramienta que trabaja principalmente con gráficos, los cuales son muy útiles para usuarios expertos, pero en realidad la herramienta únicamente señala en ellos los ciclos que hay en el código (*Tangle*, se muestra en *figura 59*). Structure101 utiliza el número de dependencias en el gráfico de dependencias como una medida de la complejidad. A continuación se detalla lo que Structure101 denomina *tangle*:

“Es un tipo diferente de complejidad estructural causado por dependencias cíclicas en los niveles superiores de la descomposición (diseño, paquetes). En la herramienta se muestran de forma separada, ya que pueden pertenecer a código muy difícil de entender, modificar, ampliar, probar y desplegar” [65].

Structure101 analiza el gráfico de dependencias y destaca el *Minimum Feedback Set* (MFS). Este es el conjunto mínimo de dependencias que necesitan ser removidas con el fin de hacer el gráfico acíclico.

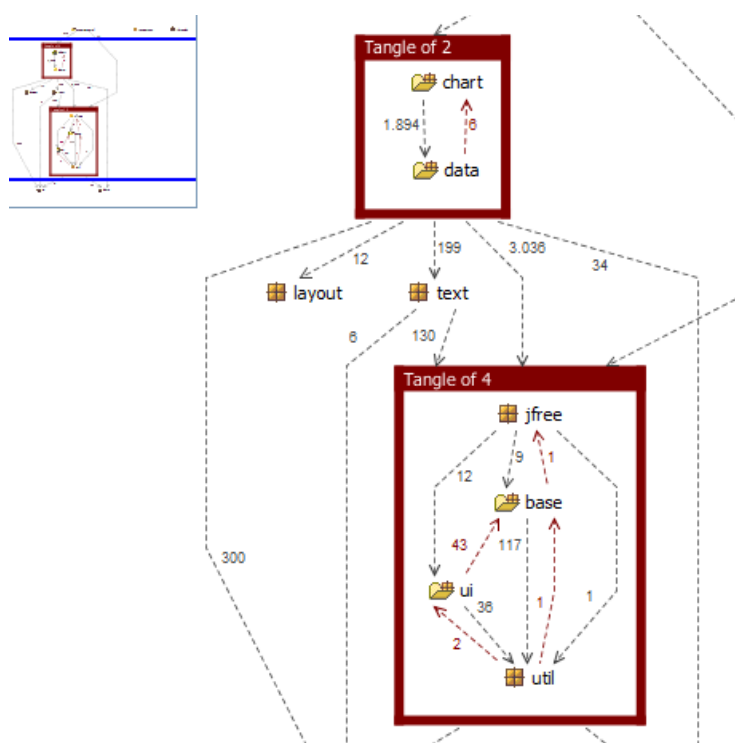


Figura 59. Structure101. Gráfico de dependencias en el que se señalan los ciclos.

Otro tipo de error que podría considerarse que identifica la herramienta es *Metrics Warnings*. Estas métricas se muestran en la vista Overview (ver figura 60) y en la vista XS Complexity pero dado que no se indica si hay un posible fallo, no se considerará *Metrics Warning* como un tipo de error detectado, aunque se hace mención a él porque un usuario experto podría detectar posibles fallos en el código gracias a estas métricas.

“Structure101 utiliza una sola métrica llamada XS (“exceso”) para medir el grado en que cualquier elemento de la composición (por ejemplo, un paquete, clase o método) supera el umbral de complejidad definido. XS refleja el tamaño de un elemento de código para el que, por ejemplo, un paquete de alto nivel excesivamente complejo tendrá mayor XS que un solo método. Esto refleja el posible impacto negativo en los procesos de desarrollo.

Una sola medida significa que es posible profundizar en un código base para descubrir las zonas de más complejidad. También ayuda a establecer prioridades para la simplificación” [65].

Se ha considerado que hay relación tanto de tipo inter como de tipo intra debido a que a la hora de mostrar los ciclos debe analizar tanto llamadas entre elementos del mismo tipo como de distintos como puede ser de clases a métodos. Ya que en la herramienta hay una vista dedicada exclusivamente a mostrar las jerarquías del código, se ha marcado la opción de herencia.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Structure101	Dependency Graph	SI	SI	SI	SI	SI	SI	SI	SI

Tabla 62. Structure101. Tabla de la característica Design Smells.

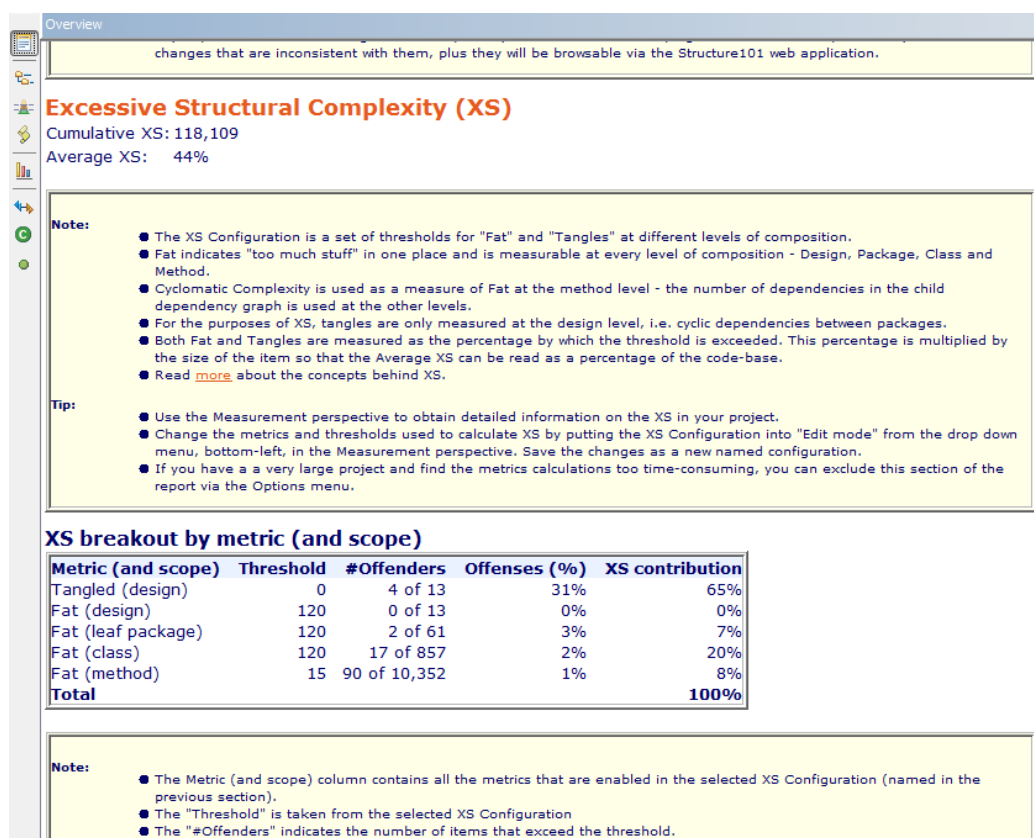


Figura 60. Structure101. Vista Overview en la que se calculan y muestran métricas.

c.3. Activity:

Lo primero que se debe mencionar en este apartado es que, pese a que en la web aparece que la herramienta incluye *Impact Analysis*¹, no se ha considerado como tal porque el programa en realidad no da la opción de conocer qué pasa si cambias algún elemento sino que, como puede verse en la figura 61, el programa muestra los elementos que dependen directamente de la clase, método o paquete que se seleccionen mediante el sombreado de los que no dependen y deja en las manos del usuario el comprobar que sucede si se modifica o elimina, por lo que no hace un análisis como tal.

¹ <http://www.headwaysoftware.com/products/structure101/client/impact-analysis.php>

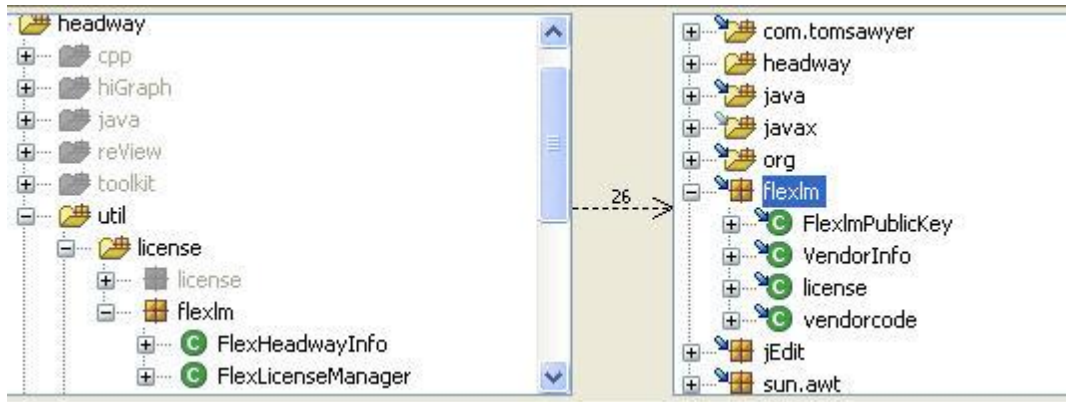


Figura 61. Structure101. Muestra del *Impact Analysis* que la herramienta incluye.

Dos tipos de actividades que si están claramente diferenciadas en la herramienta son las de Detección y Visualización.

En lo referente a la detección de los ciclos, la herramienta se sirve del cálculo de métricas, que se muestran en la vista Overview y que se calculan de manera automática nada más que se carga código. Los ciclos que detecta se muestran de manera visual, lo que lleva a la siguiente actividad, la de Visualización, la cual ofrece de forma gráfica y totalmente automática (Aunque sea el usuario el que tenga seleccionar la vista) dichos ciclos en un gráfico de dependencias.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Structure101	Detección	Metrics-Based	Fully-automated	Entity-Metrics relationships	Visual
	Visualización	Dependency Analysis	Fully-automated	Package dependencies and cycles	Visual

Tabla 63 Structure101. Tabla de la característica Activity.

StyleCop

3.2.19. StyleCop

a. Propósito

StyleCop [23] es una herramienta inicialmente creada por Microsoft que analiza el código fuente de C# para hacerlo cumplir un conjunto de reglas de estilo y consistencia. En Abril de 2010 pasó a ser una herramienta de código libre y desde entonces se ha integrado en muchas herramientas de desarrollo de terceros.

La última versión disponible de StyleCop a fecha de realización de este trabajo es totalmente compatible con todas las sintaxis de C# 4.0 y se integra en Visual Studio 2010. Los principios básicos de la herramienta son:

- *“StyleCop proporciona un valor mediante la aplicación de un conjunto común de reglas de estilo para el código C#. StyleCop continuará con el envío de un único y coherente conjunto de las normas, con la configuración de reglas mínimas permitidas. Los desarrolladores pueden aplicar sus propias normas si así lo desean”* [23].

- *“StyleCop cuenta con la capacidad para integrarse perfectamente con Visual Studio, MSBuild, Microsoft Team Foundation Server, etc. Los desarrolladores son libres de aplicar código personalizado para integrar StyleCop en el desarrollo de otras herramientas y entornos, como se describe en la documentación del SDK”* [23].

b. Información para el análisis

Versiones disponibles:

- Existen varias versiones disponibles que se integran tanto en el IDE Microsoft Visual Studio como en un MSBuild Project. Desde que Microsoft liberó el código, ésta se ha incluido también en herramientas de terceros pero se desconoce en cuales. Las versiones son la 4.4.0. que se encuentra en [23] y la 4.3.3.0. que se encuentra en el antiguo proyecto de Microsoft en la web <http://code.msdn.microsoft.com/sourceanalysis>.

Versión analizada:

- Se ha analizado la versión para Microsoft Visual Studio ya que se dispone de este entorno de desarrollo (Concretamente de la versión 2010) gracias al acuerdo existente entre la Universidad de Valladolid y Microsoft para que los alumnos de la primera puedan disponer de herramientas y programas de la segunda de manera gratuita.

Sistema Operativo requerido:

- La versión analizada funciona en entornos Windows, que es donde funciona Microsoft Visual Studio 2010.

Instalación:

- Se va a describir la instalación para Visual Studio 2010 que es la que se ha seguido para efectuar este análisis. Evidentemente, antes de instalar StyleCop será necesario tener instalada una versión compatible de Visual Studio.

El primer paso es descargar del sitio web el archivo con extensión .msi, y a continuación ejecutarlo. Aparecerá entonces una ventana de bienvenida, pulsamos siguiente y en la nueva ventana se debe seleccionar el entorno de trabajo en el que se desea instalar la herramienta (Aparecen dos opciones, Visual Studio y MSBuild), y se continúa la instalación normalmente. Tras esto, ya aparece StyleCop integrado en la pestaña herramientas de Visual Studio.

Código utilizado para el análisis:

- Es posible usar cualquiera de los códigos con extensión .csproj que se encuentran en la carpeta CSharpSamples, obtenida de <http://code.msdn.microsoft.com/cs2010samples>.

c. Análisis

c.1. Target Artefact:

Como ya se ha comentado en el propósito de la herramienta, StyleCop actúa como un plugin para Visual Studio o MSBuild que analiza código fuente escrito en C#. No se ha encontrado gran cantidad de información referente a la herramienta, lo que ha hecho imposible averiguar el tipo de representación interna que StyleCop realiza del código fuente; aparentemente no hay señales que indiquen una representación específica como podría ser AST o ASG y se marca como desconocido.

No se ha encontrado ninguna opción dentro de Visual Studio que esté relacionada con StyleCop que permita la comparación de dos versiones de un mismo proyecto ya sea cargando ambas simultáneamente o a través de un repositorio de versiones.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
StyleCop	Plugin Visual Studio / MSBuild	Source Code (C#)	NO	-

Tabla 64. StyleCop. Tabla de la característica Target Artefact.

c.2. Design Smells:

Para ver los *Design Smells* que la herramienta muestra, se ha procedido a realizar un análisis de código fuente. Para ello es necesario ejecutar StyleCop de manera manual haciendo clic con el botón derecho sobre el proyecto a analizar, que aparece cargado en el explorador de soluciones, y seleccionando Run StyleCop como puede verse en la figura 62.

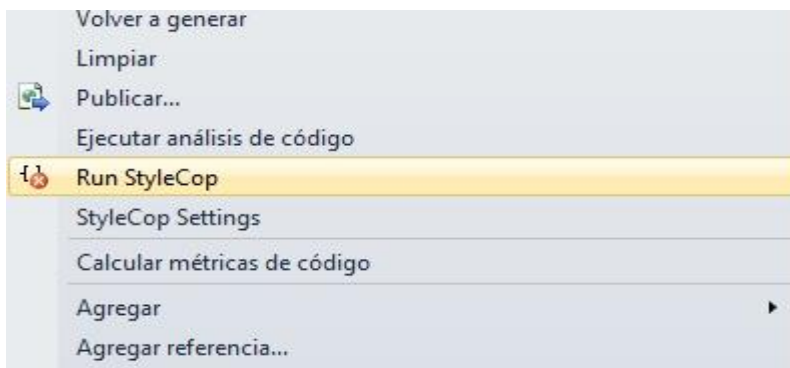


Figura 62. StyleCop. Ejecución de la herramienta sobre código C#.

Tras este proceso, StyleCop calcula automáticamente los problemas del código en función del conjunto de reglas establecidas y los muestra como se puede ver en la figura que se muestra a continuación:

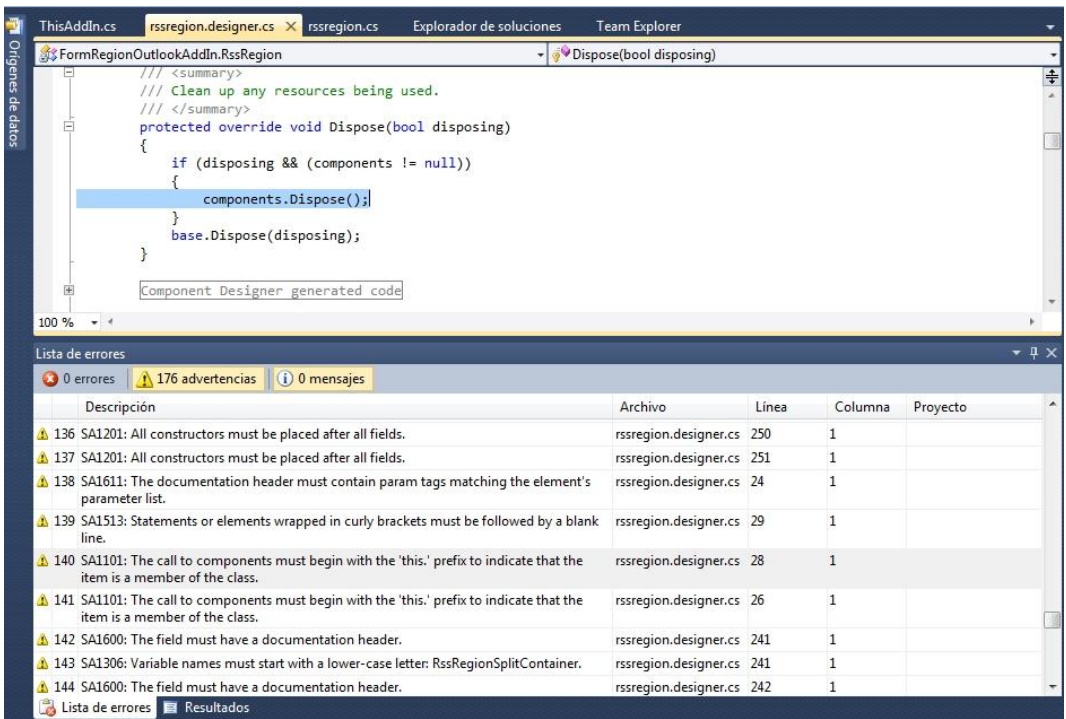


Figura 63. StyleCop. Resultados obtenidos tras efectuar un análisis.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
StyleCop	Bad Coding Style	-	-	-	-	-	-	-	-

Tabla 65. StyleCop. Tabla de la característica Design Smells.

c.3. Activity:

Una de las subcaracterísticas que incluye StyleCop de las que pertenecen a la característica Activity y que se observa ya en la figura 63, es la de detección. El proceso de detección debe iniciarlo el usuario y a continuación StyleCop, basándose en un conjunto de reglas, muestra el listado de problemas detectados. Otra de las características que incluye y que ya se ha nombrado en algún momento de este análisis es la de especificación, la cual se puede ver en la figura 64 y consiste en un proceso de selección de reglas incluidas ya en un listado que se puede encontrar haciendo clic con el botón derecho sobre el proyecto que se va a analizar y seleccionando la opción “StyleCop settings”. El grupo de reglas seleccionado forma un conjunto de reglas de detección que puede almacenarse para cargarlo en otro momento, incluso a la vez que otros conjuntos de reglas

guardados en otras ocasiones. Por defecto StyleCop tiene seleccionadas todas las reglas que incluye.

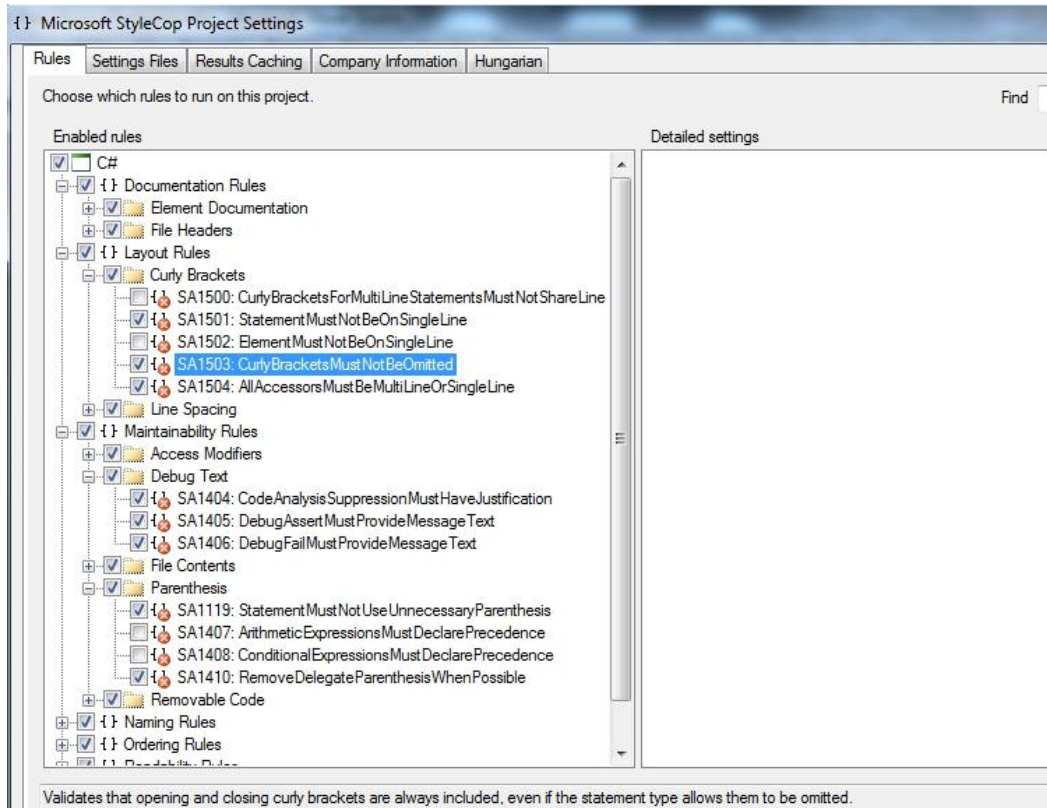


Figura 64. StyleCop. Ventana de selección de reglas para realizar un análisis.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
StyleCop	Especificación	Process Description	Interactive	Detection Rules	Textual
	Detección	Rules Based	Interactive	Entity-smells relationships	Textual

Tabla 66. StyleCop. Tabla de la característica Activity.

Together

3.2.20. Together

a. Propósito

Together [69] permite analizar, diseñar e implementar un software flexible y fácil capaz de mantener arquitecturas que deban ser modificadas cuando cambien algunos requerimientos. *“La integración de Together con las definiciones de requisitos principales y la gestión de soluciones permite el acceso directo, la reutilización y la trazabilidad por y para los requisitos, y garantiza que la expectativa de los clientes se cumpla cuando se entrega el software para ser utilizado en sus equipos”* [69].

b. Información para el análisis

Versiones disponibles:

Existe una única versión de la herramienta. Un plugin de Eclipse, que está contenido en el archivo ejecutable que se encuentra ubicado en la web oficial de Together. Este ejecutable, en un primer momento parece que instala dos versiones, una versión independiente de Together y un plugin contenido en Eclipse pero en realidad lo que instala es una versión de Eclipse con el plugin integrado y unos archivos ejecutables para lanzar Together que lo que realmente lanzan es, nuevamente, una versión de Eclipse que contiene Together. La versión gratuita de la herramienta es una versión de prueba que tiene una licencia que caduca a los 15 días de su instalación.

Versión analizada:

- Borland Together 2008 SP1 Worldwide en su versión para Windows. Se encuentra integrado en Eclipse 3.4.1

Sistema Operativo requerido:

- Windows, Linux, Mac OS X y Solaris (estos dos últimos no han sido probados).

Instalación:

- Una vez descargado el archivo ejecutable que contiene la instalación de Together, su instalación es la normal en una aplicación para Windows. Una vez instalada la carpeta contenedora de la herramienta en la ubicación deseada, al acceder al directorio de instalación se puede encontrar el ejecutable de Eclipse con el plugin ya integrado y otro archivo ejecutable, *Together.exe*, que es el otro lanzador de Eclipse con el plugin integrado al que se hacía referencia en el apartado de las versiones disponibles.

Para completar la instalación es necesario abrir Together y activar la licencia de 15 días de prueba, mediante un código de activación previamente descargado de su web.

Código utilizado para el análisis:

- Proyecto09 y jHotdraw 5.2

c. Análisis

Together es una herramienta asentada sobre los pilares de Eclipse, lo que puede considerarse un factor a favor de la herramienta ya que este entorno de desarrollo es muy popular. En cuanto a las características técnicas que el entorno aporta al usuario como diseñador, se pueden encontrar:

- *“Modelado visual de diagramas. Este es un punto básico y generalizado en este tipo de herramientas. Esta herramienta en concreto aporta un gran abanico de posibilidades”*. Aun así, la modificación de propiedades de los elementos gráficos resulta, en ocasiones, poco claro y consistente.
- *“Soporte para UML”*.
- *“Soporte del lenguaje OCL empleado actualmente, entre otras cosas, para los sistemas MDA”*.
- *“Generación de código fuente. El entorno permite la generación de código automáticamente a partir de los diseños de clases, y a la inversa, las modificaciones en los códigos fuente asociados, por ejemplo, a clases, se manifiestan y propagan en los diagramas”*.
- *“Soporte e integración de patrones de diseño. Existen de manera preestablecida una serie de patrones de diseño que pueden ser aplicados fácilmente al diseño. Además, se permite la creación de patrones propios y posteriormente poder exportarlos. Finalmente, también permite el reconocimiento de patrones en los diseños generados”*.
- *“Generación de documentación. El sistema permite la creación de plantillas de generación de documentación, y después ser exportado a varios formatos (.doc, .pdf ó .html). Estas plantillas de documentación se generan mediante un editor gráfico”*.

Entre todas estas características de Together se encuentra la detección de *design smells*, que es lo que realmente interesa en este trabajo.

c.1. Target Artefact:

Ya es sabido que Together trabaja como plugin integrado en Eclipse. Puede trabajar con varios formatos de entrada, desde modelados UML hasta el código fuente de lenguajes como Java, C, C++ y Corba.

Together permite a varios usuarios trabajar en un mismo modelo gracias al control de versiones, previa configuración para trabajar de este modo. Puede comparar los modelos, fusionar inconsistencias si es necesario y crear notificaciones en las partes del modelo donde se han realizado cambios.

La herramienta también propone una transformación del modelo basándose en el cumplimiento de la norma QVT. Este estándar define una forma de transformar el modelo de origen en un modelo de destino apropiado. Se utilizará este tipo de transformaciones (se considera *object model* por la falta de información específica) para que la herramienta trabaje de forma óptima en la búsqueda de problemas en el código.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
Together	Plugin Eclipse	UML Model Source Code	SI	Object Model

Tabla 67. Together. Tabla de la característica Target Artefact.

c.2. Design Smells:

Together es capaz de reconocer tres tipos *smells*. Dos de ellos son *bad coding style* y *bad smells/disharmonies*, los cuales detecta gracias a la herramienta *Code Audits* que se puede localizar una vez que está activada la licencia de Together y se carga un proyecto creado en uno de los lenguajes que la aplicación admite. Para realizar esta operación, dentro de *package explorer* se hace clic con el botón derecho encima del proyecto que se quiere estudiar, a continuación hay que desplazarse hasta *QA source* y seleccionar *Audits*. Entonces se despliega una ventana pidiendo que se marque la parte de código que se desea analizar. También se puede encontrar un botón denominado *preferences* que si se pulsa abre una ventana emergente dónde se pueden marcar los diferentes aspectos a analizar, como se muestra en la *figura 65*.

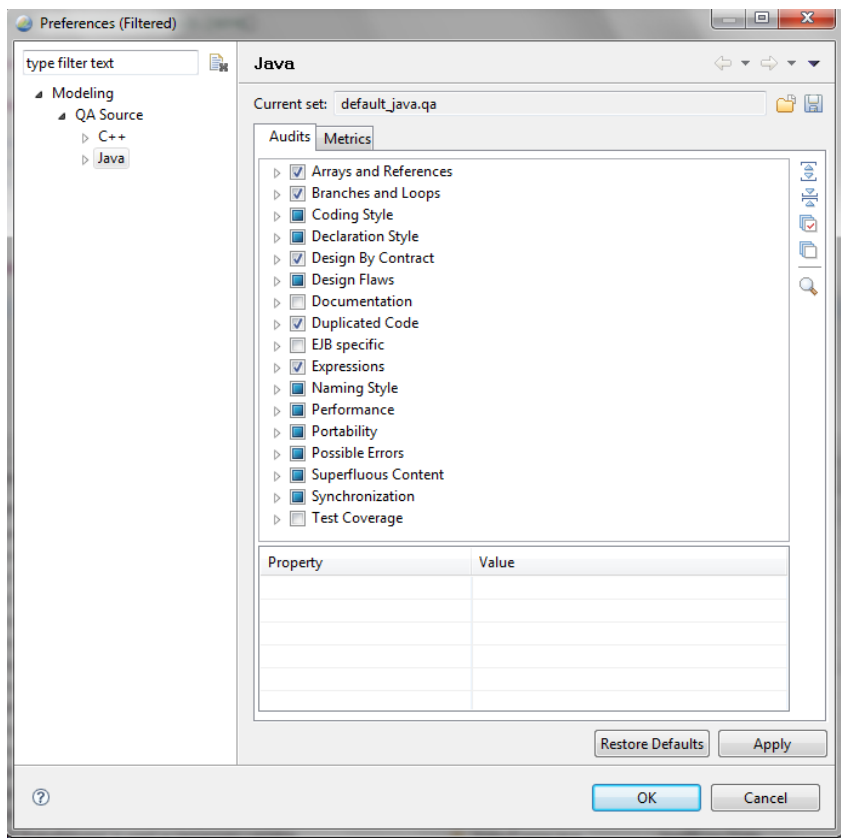


Figura 65. Together. Preferencias para Audits.

La gran mayoría de categorías que se muestran en la figura anterior se corresponden con posibles casos de *bad coding style* a excepción de *design flaw*, donde aparecen los errores relacionados con los *bad smells* y *disharmonies*. En el apartado **c.3. Activity** se verá cómo se puede cambiar el grado de tratamiento de los posibles *smells*, así como determinados parámetros que los delimitan. Una vez seleccionadas las preferencias, se pulsa OK y de nuevo aparece la ventana para la ejecución de *Audits*; pulsando nuevamente OK se realiza el análisis establecido, mostrando todos los errores considerados por la aplicación. Los *smells* aparecen listados en orden decreciente de riesgo atendiendo al baremo que Together utiliza.

Description	Resource	In Folder	Location
Overriding non-abstract method 'clone' with abstract method	Figure.java	/p1/sources/CH/ifa/draw/framework	line 124
Non-abstract method 'java.lang.Object.clone'	p1		
Integer division in floating-point context	Geom.java	/p1/sources/CH/ifa/draw/util	line 164
Integer division in floating-point context	Geom.java	/p1/sources/CH/ifa/draw/util	line 164
Integral argument 0 of method 'elementAt' may be out of domain	PolyLineFigure.java	/p1/sources/CH/ifa/draw/figures	line 242
Integral argument 0 of method 'elementAt' may be out of domain	PolyLineFigure.java	/p1/sources/CH/ifa/draw/figures	line 243
Integral argument 0 of method 'elementAt' may be out of domain	ReverseVectorEnumerato...	/p1/sources/CH/ifa/draw/util	line 30
Abstract class 'AbstractConnector' contains public constructors	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 17
Public constructor in abstract class	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 34
Public constructor in abstract class	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 41
Constant field 'abstractConnectorSerializedDataVersion' should be final	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 28
Field 'abstractConnectorSerializedDataVersion' is not used	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 28
Field 'abstractConnectorSerializedDataVersion' is used as temporary vari	AbstractConnector.java	/p1/sources/CH/ifa/draw/standard	line 28

Figura 66. Together. Muestra de *bad coding style*.

Otro tipo de error que Together es capaz de detectar es *metrics warnings*. La forma de proceder para llegar hasta estas métricas es igual que para llegar al análisis de *Audits*, variando que una vez que se este posicionado en *QA source*, se debe pulsar *Metrics* en lugar de *Audits*. Al igual que sucedía con *audits*, se despliega una ventana para seleccionar en qué parte del código se desea efectuar el cálculo de las métricas. Aquí también existe la opción de modificar las métricas a analizar mediante el botón preferencias. En la figura 67, se observa cómo se marcan los *metrics warnings* una vez ejecutado dicho análisis.

Resource	A	AC	AHF	A	CA	CBO	CC	CE	C.	CR	ChC	DAC	DD	DOIH	EC	FO	HDiff
p1		21	91	6		71	98		4	0		8		8	63	26	1709
CH.ifa.draw.applet	0	66			3	52	57	54		22				6	34	22	23
CH.ifa.draw.application	0	91			6	71	78	69		29		7		6	59	26	18
CH.ifa.draw.contrib	17	11			6	17	68	50		7		4		7	34	11	331
CH.ifa.draw.figures	9	11			16	19	59	60		6		3		4	41	13	446
CH.ifa.draw.framework	78	2			92	16	40	24		36		1		3	40	16	19
CH.ifa.draw.samples.javadraw	0	13			0	35	17	67		6		4		8	14	10	112
CH.ifa.draw.samples.net	0	5			0	14	19	25		9		1		7	13	8	20
CH.ifa.draw.samples.nothing	0	0			0	13	3	18		8		0		7	3	2	10
CH.ifa.draw.samples.pert	0	7			0	20	46	39		3		2		7	35	15	58
CH.ifa.draw.standard	18	10			58	27	98	66		0		8		7	63	21	456
CH.ifa.draw.util	25	11			66	10	32	57		12		3		7	16	8	216

Figura 67. Together. Muestra de métricas.

En la figura se puede observar cómo se marcan métricas con un punto rojo (cuando se supera un valor establecido) o azul (cuando está por debajo del valor establecido).

Together presenta relaciones tanto inter como intra, debido principalmente al gran número de comprobaciones realizadas tanto en elementos de la misma entidad como entre distintas entidades. También realiza la medición de diversas métricas relacionadas con la herencia, algunas de las cuales pueden aparecer indicadas como *metrics warnings*.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Together	Bad Smells / Disharmonies								
	Metrics Warnings	NO	NO	SI	SI	SI	SI	SI	SI
	Bad Coding Style								

Tabla 68. Together. Tabla de la característica Design Smells.

c.3. Activity:

La primera de las actividades que se aprecian al poner en marcha el análisis de un proyecto es la especificación. En el apartado anterior ya se hizo mención a ello cuando se habló de las reglas y normas que se podían activar o desactivar para la detección de *bad smells/disharmonies* y de *bad coding style*, y de como activar o desactivar el cálculo de las diferentes métricas que Together ofrece.

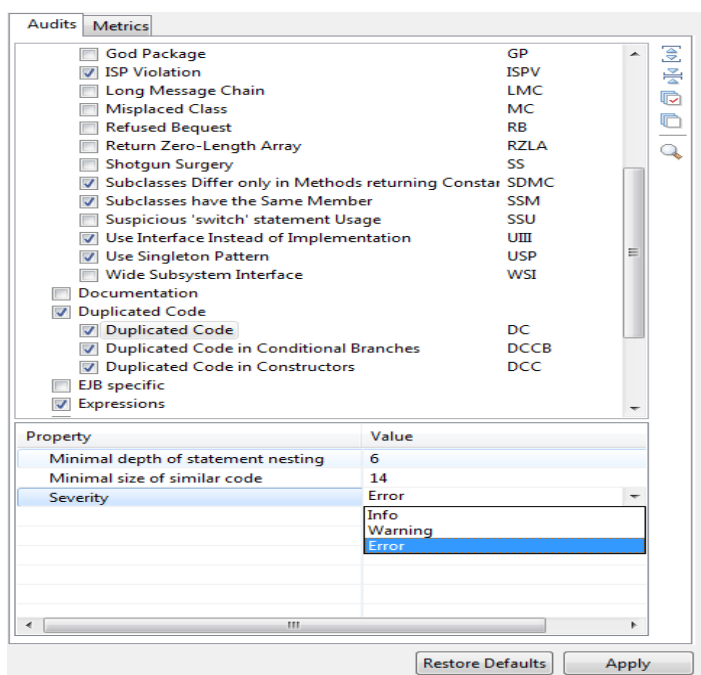


Figura 68. Together. Ventana con las opciones de especificación.

La actividad de detección de los *smells* se realiza de forma totalmente automática, una vez que el usuario ha definido en función de sus necesidades las preferencias tanto de *audits* como de *metrics*, mostrando los resultados de ambos análisis como aparecían en las figuras 66 y 67.

La corrección es la última actividad a destacar de la herramienta. Una vez que se ha realizado el análisis, existe la posibilidad de que la herramienta proporcione sugerencias de corrección para determinados errores identificados mediante la opción *audits* y relacionados con el *bad coding*

style, pero no se da para la totalidad de los *smells* detectados. Para saber si un error tiene posibilidad de corrección basta con observar si se encuentra activado el icono *quick fix*, situado en la zona superior derecha encima de la lista de errores detectados. Como se muestra en la *figura 69*, al pulsar sobre *quick fix* se despliega una ventana de confirmación para aceptar la corrección. Si se desea observar las modificaciones que se aplicarán y en que zona del código se harán efectivas, basta con pulsar en *preview*, dónde se mostrará el código antes y después de la corrección.

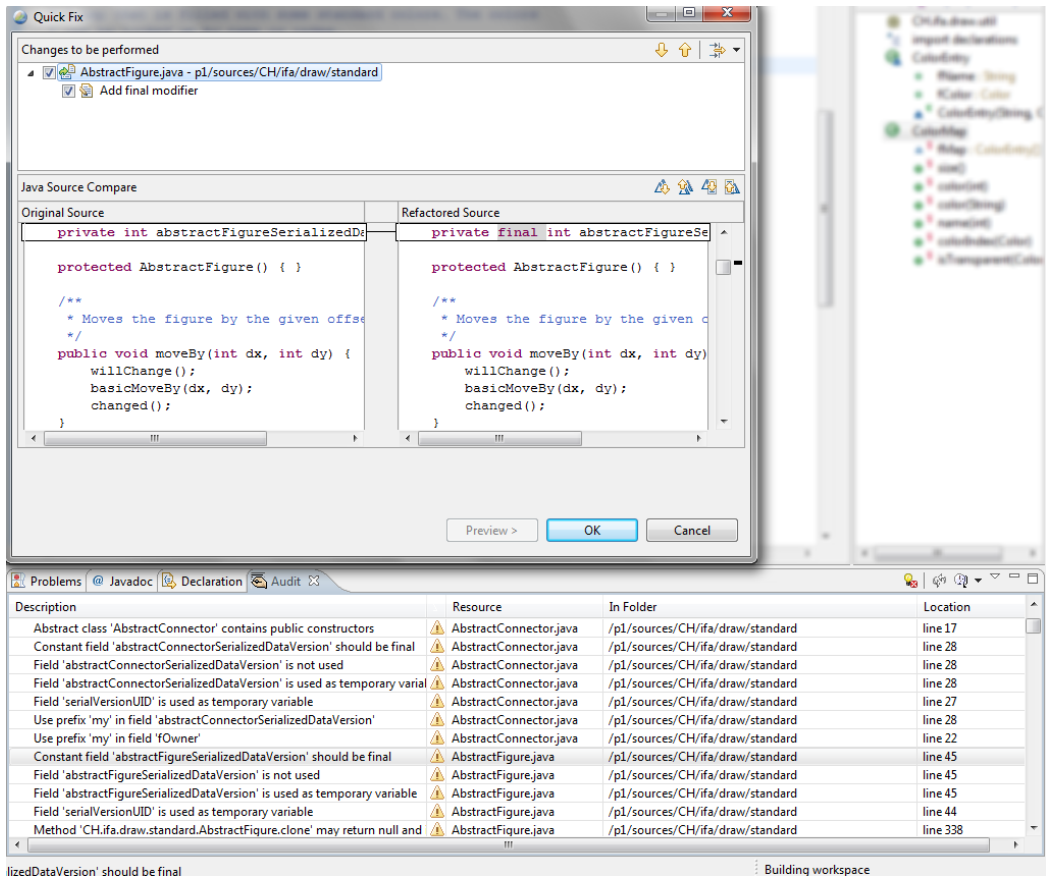


Figura 69. Together. Actividad de Corrección.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Together	Especificación	Process description	Interactive/ Manual	Detection Metrics / Design Smell Specification	Numerical
	Detección	Metrics Based/ Rules Based	Fully-automated	Entity-smells/metrics relationships	Textual
	Corrección	Rules-Based	Interactive	Correction suggestions	Textual

Tabla 69. Together. Tabla de la característica Activity.

TRex

3.2.21. TRex

a. Propósito

El “Testing and Test Control Notation” (Notación de escritura y control de pruebas, TTCN-3) realiza tests de especificación y de implementación de lenguaje estandarizados por el Instituto Europeo de Normas de Telecomunicación (ETSI) y la Unión Internacional de Telecomunicaciones (UIT). Aunque TTCN-3 tiene sus raíces en las pruebas funcionales de caja negra de los sistemas de telecomunicaciones, hoy en día también se utiliza en otros ámbitos, tales como protocolos de Internet, arquitectura aeroespacial o arquitecturas orientadas a servicios. *“TTCN-3 puede ser utilizado no sólo para la especificación y ejecución de pruebas funcionales, sino también para pruebas de escalabilidad, robustez o esfuerzo. TTCN-3 se basa en una notación de base textual y varios formatos de presentación”* [73, 74]. Las herramientas comerciales TTCN-3 y las soluciones internas dan apoyo a conjuntos de pruebas de edición y las compilan en código ejecutable.

“La experiencia de Motorola ha demostrado que el mantenimiento no sólo de edición y ejecución de bancos de pruebas TTCN-3, sino también de suites de prueba TTCN-3 es una cuestión importante que requiere el apoyo de herramienta. En la actualidad, no hay herramientas existentes para evaluar y mejorar la calidad de las suites de prueba TTCN-3” [73, 74]. Para ello, Motorola ha colaborado con la Universidad de Göttingen para desarrollar una herramienta de refactorización y de medición del código TTCN-3, llamada TRex [43].

Los objetivos de TRex son los siguientes: (1) permitir la evaluación de una serie de pruebas TTCN-3 con respecto a las lecciones aprendidas con la experiencia, (2) proporcionar un medio para detectar las oportunidades para evitar cualquier problema, y (3) proporcionar un medio para la reestructuración de bancos de pruebas TTCN-3 para mejorar el código con respecto a cualquier problema existente.

b. Información para el análisis

Versiones disponibles:

- Versión independiente v.4.1.1, tanto en un archivo ejecutable como en código fuente.
- Plugin de Eclipse

Versión analizada: Versión independiente para Windows.

Sistema Operativo requerido:

- La versión independiente tiene descargas para sistemas Windows y Linux. El plugin de Eclipse funcionará en los Sistemas Operativos en los que Eclipse funcione.

Instalación:

- La instalación de la versión independiente se realiza mediante la descarga de un archivo ejecutable. Lanzando dicho archivo y siguiendo las instrucciones que aparecen por pantalla se completa su instalación.

Código utilizado para el análisis:

- Se ha utilizado el código que se encuentra dentro de la carpeta ETSI WiMax Network Abstract Test Suite, a su vez incluida en la carpeta Source Code TTCN-3 contenida en el cd. El código se ha obtenido de la web <http://www.ttcn-3.org/PublicTTCN3TestSuites.htm>.

c. Análisis**c.1. Target Artefact:**

Uno de los aspectos más destacables de esta herramienta, como ya se ha comentado en el propósito, es que está destinada al análisis de código TTCN-3. La herramienta en sí es un plugin de Eclipse incluso en su versión independiente, ya que como sucedía con Together, esta versión no es más que una versión de Eclipse optimizada para su uso con TRex, en la que se ocultan algunas opciones del IDE que no tienen relación con la herramienta (Es posible acceder a ellas, por lo que no se eliminan) y se añaden las que incluye el plugin para Eclipse.

Para transformar el código fuente TTCN-3 y posteriormente realizar el análisis, la herramienta utiliza un árbol AST (ver *figura 70*) [73].

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
TRex	Independiente Plugin Eclipse	Source Code (Test TTCN-3)	NO	AST

Tabla 70. TRex. Tabla de la característica Target Artefact.

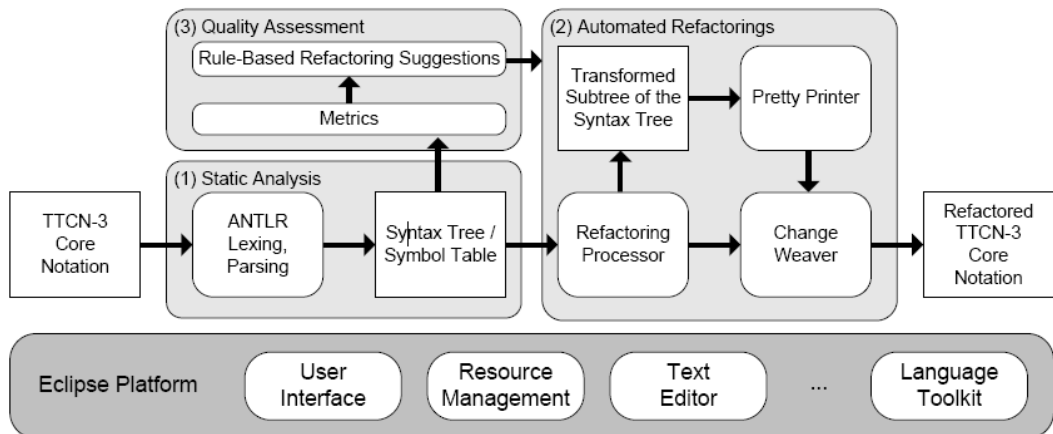


Figura 70. TRex. Cadena de acciones de la herramienta. Imagen extraída de [73].

c.2. Design Smells:

Analizando la documentación disponible [73, 74, 75, 76 y 77], se observa que la herramienta realiza una búsqueda de código duplicado, lo cual se ha catalogado en esta memoria como *Bad Smell* y también como *Disharmony*. Se procede a realizar un análisis de código fuente para comprobar que esto no sólo aparece en las especificaciones sino que también está incluido en la

herramienta. Se puede observar en la *figura 71* que efectivamente, la herramienta realiza dicha búsqueda ya que aparecen resultados para *Duplicated Code*.

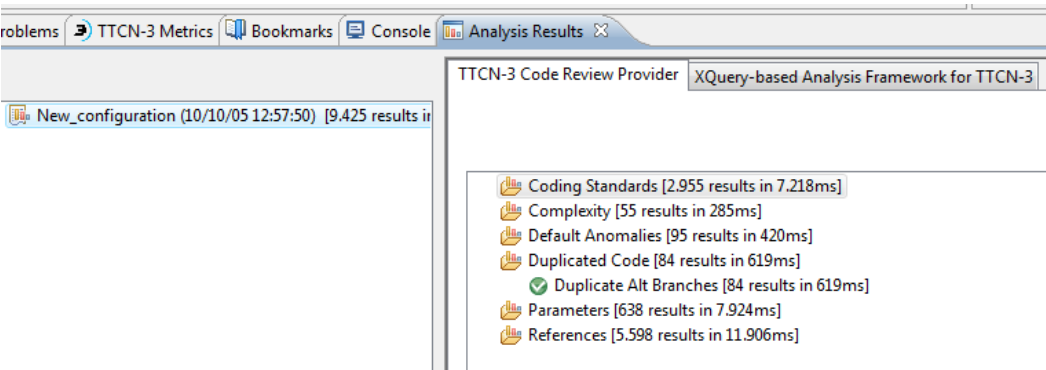


Figura 71. TRex. Resultado del análisis realizado a código TTCN-3.

Otra de los defectos que la herramienta detecta y que se ve también en la *figura 71* es lo que se viene denominando *Bad Coding Style* ya que la herramienta se encarga de detectar defectos de diseño en función de los estándares de código TTCN-3 (Carpeta *Coding Standards* de la figura).

Debido a la singularidad de esta herramienta en cuanto al código analizado (Test de Scripts), no es posible especificar granularidad, relaciones y herencia ya que se desconoce la estructura que sigue el código TTCN-3 y por tanto las posibles jerarquías existentes en el código.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
TRex	Bad Smells / Disharmonies	-	-	-	-	-	-	-	-
	Bad Coding Style	-	-	-	-	-	-	-	-

Tabla 71. TRex. Tabla de la característica Design Smells.

c.3. Activity:

Como ya se ha visto en la *figura 70*, TRex utiliza un AST para realizar la extracción del modelo en la fase de análisis. En la misma figura se observa también que la herramienta realiza la actividad de detección mediante el uso de métricas y, como se indica en el documento de Martin Bisanz [76], también se basa en patrones para realizar dicha tarea. Para ejecutar esta detección, basta con hacer clic con el botón derecho del ratón sobre el código de la pestaña Navigator que se desea analizar y seleccionar Analysis → Analysis configurations e inmediatamente se abrirá una nueva ventana con el mismo nombre, Analysis Configurations. Aparecen las principales opciones ya preseleccionadas por lo que para realizar un análisis basta con hacer clic en el botón Analyze.

En la misma ventana “Analysis Configurations”, se puede apreciar otra de las actividades de la herramienta, la especificación. Si se selecciona la pestaña Rules, aparecen varias opciones preseleccionadas, las cuales se pueden combinar como se desee para así crear diferentes análisis.

También permite exportar otras reglas o importar las nuevas combinaciones que se realicen para poder cargarlas en futuros análisis.

La última de las actividades que realiza TRex es la de corrección. Dicha actividad aparece mostrada nuevamente en la *figura 70*, donde se ve un apartado de sugerencias de corrección, las cuales de basan en reglas (*Rule-Based Refactoring Suggestions*). Se comprueba que estas sugerencias están implementadas en la herramienta y así es, en la herramienta independiente las opciones de refactorización aparecen en la pestaña Refactor de la ventana principal.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
TRex	Análisis	Model Extraction	Fully-automated	-	-
	Especificación	Process Description	Interactive	Detection Rules	Textual
	Detección	Pattern-Based / Metrics-Based	Fully-automated / Interactive	Entity-metrics relationships	Textual
	Corrección	Rules-Based	Interactive	Refactoring Suggestions	Textual

Tabla 72. TRex. Tabla de la característica Activity.

XRefactory

3.2.22. XRefactory

a. Propósito

XRefactory [38], también conocido como Xref, X-ref y Xref-Speller, es una herramienta de desarrollo profesional para C/C++ y Java que proporciona técnicas de detección/corrección para completar el código analizado y posibles refactorizaciones.

b. Información para el análisis

Versiones disponibles:

- Existen diferentes versiones disponibles tanto para la versión que analiza código C/Java, como para la que analiza C/C++. Esta última no es gratuita, necesita del pago de una licencia de activación.

Versión analizada: XRefactory 1.6.10 para C/Java en Linux.

Sistema Operativo requerido:

- La herramienta puede ser integrada en jEdit, Emacs [81] y XEmacs [82] para plataformas Windows, Linux, Solaris y Mac OS X.

Instalación:

- Como ya se ha mencionado en el apartado de versión analizada, se ha elegido una de las versiones para Linux para llevar a cabo su estudio. El porqué de esta decisión se retomará en el punto **c.1. Target artefact**. Para el correcto funcionamiento de la herramienta, es necesario tener instalado en el equipo uno de los editores dónde XRefactory podrá ser integrado. En este análisis se ha probado la herramienta con Emacs y XEmacs. Tras esta instalación (basta con descargar del sitio web del editor deseado el ejecutable y lanzarlo) se descarga la herramienta de la página oficial, para posteriormente proceder a su instalación siguiendo los pasos siguientes (Pasos para la instalación a partir del código fuente):

- Descomprimir el archivo descargado.
- Ir al directorio *xref-any* y ejecutar *make*.
- Para finalizar la instalación, ir al directorio creado "xref" y lanzar la orden
sh ./xrefsetup

Código utilizado para el análisis:

- jHotdraw 5.2.

c. Análisis

c.1. Target Artefact:

XRefactory funciona como herramienta integrada en los editores jEdit, Emacs y Xemacs. Para el análisis se ha utilizado su versión para Emacs y Xemacs, ya que son las únicas que tienen una

interfaz relativamente sencilla para el uso de la herramienta y que proporcionan algún tipo de resultado relacionado con la aparición de defectos de diseño en el código.

El código fuente que la herramienta es capaz de interpretar es el escrito en Java y C. En la versión de pago también es capaz de analizar C++. Se desconocen las características diferentes que dicha versión de pago incorpora respecto de la versión analizada. En la versión de XRefactory analizada no existe la posibilidad de comparar diferentes versiones de código para efectuar el análisis.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERSIONES	TIPO DE REPRESENTACION
XRefactory	Plugin jEdit / Emacs / Xemacs	Source Code (JAVA/C/C++)	NO	-

Tabla 73. XRefactory. Tabla de la característica Target Artefact.

Es importante destacar que para su correcta utilización se debe de tener algún conocimiento previo de las herramientas Emacs y Xemacs, ya que de lo contrario se complica bastante el manejo de XRefactory.

c.2. Design Smells:

Como ya se ha comentado en el apartado anterior, la integración tanto en Emacs como Xemacs es la única que a la hora de efectuar el análisis ha ofrecido de forma evidente la detección de algún tipo de error de los buscados en este trabajo (Es posible que en el resto de editores también aparezcan e incluso aparezcan más características de las que aquí se detallan tanto en los otros editores como en Emacs y Xemacs ya que la necesidad de conocer el funcionamiento y manejo de dichos editores en los que XRefactory se puede integrar ha limitado bastante el análisis realizado). En la *figura 72* se puede observar que al desplegar las posibles tareas que es capaz de realizar XRefactory, la única con una posible relación con la detección de *smells* es la que se denomina *dead code detection*, que se agrupa dentro del *bad coding style* [1].

Al analizar solamente la posible aparición de código muerto, se considera que la actividad de xrefs sólo llega hasta nivel de método. Se considera también que carece de cualquier posible relación tanto entre entidades del mismo nivel como entre otras de nivel jerárquico diferente. La herramienta carece de mecanismos que analicen alguna característica relacionada con la herencia. Todos estos puntos podrían variar si apareciese algún nuevo tipo de error que no se ha localizado ya que como se ha comentado, el análisis está limitado por el conocimiento de los procesadores Emacs y XEmacs y por la nula información disponible acerca de XRefactory.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
XRefactory	Bad Coding Style	NO	NO	NO	NO	SI	NO	NO	NO

Tabla 74. XRefactory. Tabla de la característica Design Smells.

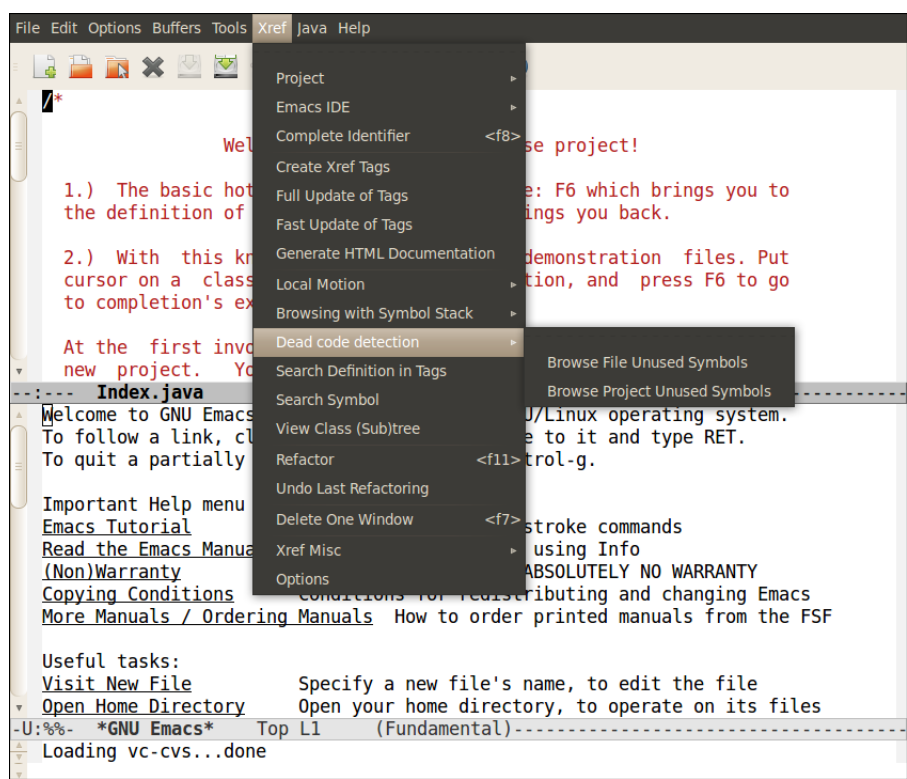


Figura 72. XRefactory. Dead Code en XRefactory integrado en Emacs.

c.3. Activity:

De esta herramienta sólo se ha localizado un tipo de actividad existente, la de detección, aunque es posible que exista la actividad de corrección ya que, como puede verse en la figura anterior, existe la opción Refactor pero en las pruebas efectuadas no se ha observado ningún cambio (Probablemente debido, de nuevo, al desconocimiento del funcionamiento de los editores utilizados). Como se muestra en la figura 72, al hacer clic sobre *dead code detection*, se despliega una nueva ventana donde se muestra el lugar dónde la herramienta ha encontrado los posibles errores.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
XRefactory	Detección	Rules-based	Interactive	Entity-smells relationships	Textual

Tabla 75. XRefactory. Tabla de la característica Activity.

*Como ya se ha venido comentando, es posible que este análisis tenga muchas deficiencias ya que el desconocimiento de los editores en los que XRefactory se puede integrar hace que la utilización de dichos editores sea bastante compleja. Además XRefactory no incluye una interfaz propia, es la misma que la de los editores y lo único que incluye son las opciones en el submenú de la barra de herramientas del editor, por lo que, junto a la escasa información disponible acerca de la herramienta, hacen muy difícil la tarea de efectuar un análisis a fondo ya que estudiar el manejo de uno de los editores es una tarea que llevaría mucho tiempo y que los tutores desaconsejan ya que no se considera una tarea incluida en este trabajo.

Otras herramientas

3.3. Otras herramientas

Aquí se encuentra información acerca de las herramientas que aparecen en la lista inicial de herramientas descargadas y que han dado diversos problemas bien en su instalación o bien a la hora de efectuar el análisis se descartan por no estar relacionadas con la búsqueda de *Design Smells* tal y como aparecen descritos en este proyecto.

3.3.1. CodeClinic

CodeClinic [10] no es una herramienta de uso comercial o software libre como casi todas las analizadas en este trabajo sino que fue desarrollada en una tesis por Adrian Trifu [85], a quién hay que agradecer el disponer de ella ya que la facilitó vía correo electrónico.

Como el propio Adrian Trifu ha indicado en el email en el que adjuntaba la herramienta, CodeClinic es un plugin de Eclipse. La herramienta ha sido recibida para este estudio dentro de un Workspace (UniWorkspace2, incluido en el CD que acompaña a esta memoria) por lo que los pasos a seguir para ejecutar la herramienta son los siguientes:

Abrir Eclipse y seleccionar File → Import → General → Existing Project into Workspace → Siguiente; en la ventana que aparece, buscar UniWorkspace2 y finalizar. Aparece entonces el código cargado de CodeClinic en la pestaña Package Explorer de Eclipse. A continuación hacer clic con el botón derecho sobre la carpeta cargada y seleccionar Run As → Eclipse Application. Se abrirá automáticamente un nuevo Eclipse en el que se puede encontrar un nuevo ítem del menú superior bajo el nombre CodeClinic.

Las características principales de la herramienta vienen muy bien detalladas en este fragmento extraído de la tesis de Adrian Trifu [85]:

“CodeClinic está totalmente integrado con el IDE Eclipse. La herramienta se integra perfectamente con las herramientas de desarrollo Java, añadiendo varios elementos para el entorno gráfico, y permitiendo a los desarrolladores tener un escáner de proyectos para encontrar posibles defectos de diseño en el código. Además, es posible restringir el análisis a paquetes individuales y las unidades de compilación.

Los candidatos a design smell que se encuentran se presentan en dos puntos de vista específicos. La vista design flaw list, que se muestra en la parte inferior de la pantalla, muestra una lista por categorías de los casos de problemas de diseño que se encuentran en el código, y permite al ingeniero saltar a los fragmentos de código afectado que se muestran y se destacan en la ventana principal de edición del entorno. El segundo punto de vista, que se muestra a la derecha de la ventana principal, muestra información detallada (que es específica para cada tipo de defecto de diseño) sobre la instancia defecto seleccionada. Además, este punto de vista presenta al ingeniero una secuencia de preguntas que debe responder, con el fin de confirmar o rechazar a un candidato determinado.

La herramienta en sí está diseñada como un marco que puede ser extendido mediante el mecanismo estándar de Eclipse de puntos de extensión. La figura 73 representa una vista lógica muy abstracta de la arquitectura del sistema. El plugin de Eclipse que contiene el marco de la aplicación real se muestra a la izquierda, y la extensión plugin que define a un defecto de diseño único a la derecha. El componente sombreado que se describe a continuación de los dos plugins

pertenece a las herramientas para desarrolladores de Java (JDT) de la plataforma Eclipse, y representa el modelo estructural del proyecto Java de código fuente que se analizó.”

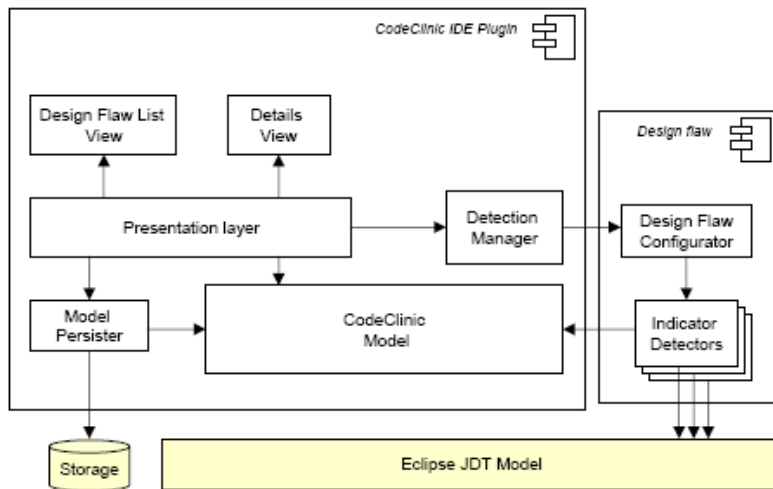


Figura 73. CodeClinic. Vista lógica de la arquitectura del sistema [85].

Pese a que, como se ha podido ver en la figura anterior y se ha podido leer en el fragmento extraído de [85], CodeClinic tiene unas características que parecen adaptarse a la perfección a las herramientas que se están analizando en este trabajo, no se ha podido efectuar el análisis de la herramienta debido a un error en la ejecución que no se ha logrado solucionar y que se muestra en la figura 75. A la vez que se muestra este error, en el primer Eclipse se van indicando los fallos que va generando CodeClinic y que se muestran en la figura 74.

```

Eclipse Application [Eclipse Application] C:\Program Files\Java\jre6\bin\javaw.exe (30/11/2010 10:11:43)
java.lang.NullPointerException
    at de.fzi.codeclinic4eclipse.model.framework.ProblemAlgorithm.checkSearchSpace(ProblemAlgorithm.java:45)
    at de.fzi.codeclinic4eclipse.model.framework.ProblemAlgorithm.run(ProblemAlgorithm.java:37)
    at de.fzi.codeclinic4eclipse.model.framework.ProblemDetector.detect(ProblemDetector.java:59)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectWithoutErrorCheck(DetectionManager.java:133)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:170)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:184)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:194)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.startRecursiveDetection(DetectionManager.java:75)
    at de.fzi.codeclinic4eclipse.presenter.DetectionJobFactory$1.run(DetectionJobFactory.java:54)
    at org.eclipse.core.internal.jobs.Worker.run(Worker.java:54)

!ENTRY de.fzi.codeclinic4eclipse 4 0 2010-11-30 10:21:12.622
!MESSAGE java.lang.NullPointerException
!STACK 0
java.lang.NullPointerException
    at de.fzi.codeclinic4eclipse.model.framework.ProblemAlgorithm.checkSearchSpace(ProblemAlgorithm.java:45)
    at de.fzi.codeclinic4eclipse.model.framework.ProblemAlgorithm.run(ProblemAlgorithm.java:37)
    at de.fzi.codeclinic4eclipse.model.framework.ProblemDetector.detect(ProblemDetector.java:59)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectWithoutErrorCheck(DetectionManager.java:133)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:170)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:184)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.detectRecursive(DetectionManager.java:194)
    at de.fzi.codeclinic4eclipse.model.framework.DetectionManager.startRecursiveDetection(DetectionManager.java:75)
    at de.fzi.codeclinic4eclipse.presenter.DetectionJobFactory$1.run(DetectionJobFactory.java:54)
    at org.eclipse.core.internal.jobs.Worker.run(Worker.java:54)
  
```

Figura 74. CodeClinic. Error generado por el Eclipse que actúa como lanzador de la aplicación.

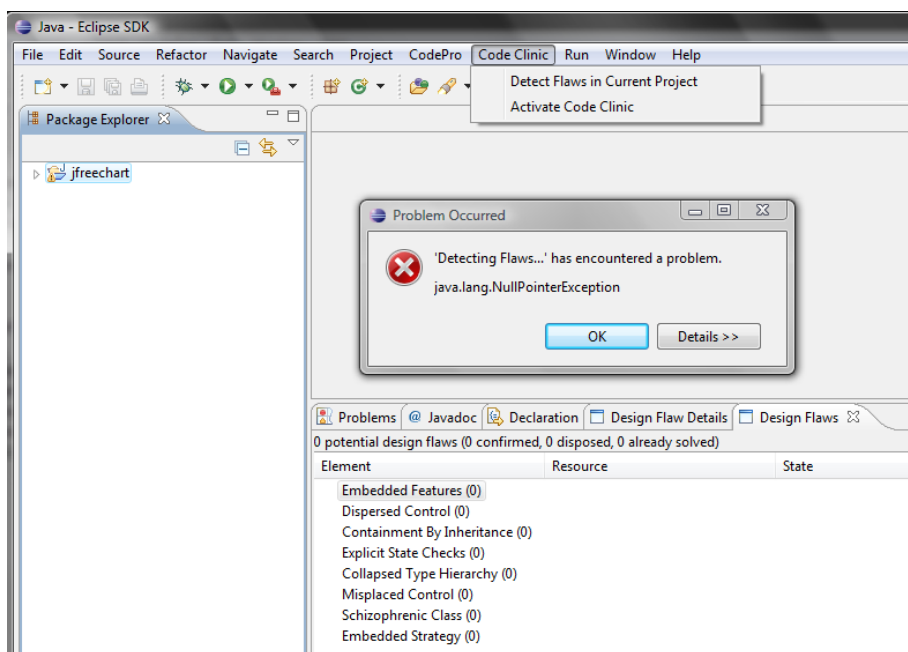


Figura 75. CodeClinic. Error obtenido en la ejecución de la herramienta.

3.3.2. Hammurapi

Hammurapi [86] “es una plataforma que mide la calidad del código para mitigar los riesgos de la externalización del desarrollo del software. Es otras palabras, Hammurapi es una solución integral de análisis de grandes sistemas software desarrollados con el uso extensivo de mano de obra por contrata. Hammurapi produce un informe consolidado de toda la aplicación. El informe identifica los problemas más importantes en la aplicación y permite navegar hasta la fuente en unos pocos clics” [86].

En documentos como [87], creado por el principal responsable de la herramienta, y también en la propia web, se puede observar cómo aparece bastante información que de haber sido posible realizar el análisis hubiese sido de gran utilidad.

El motivo de incluir esta herramienta en este apartado se debe a que no se puede realizar ningún tipo de análisis con ella una vez integrada en Eclipse, IDE para el que incluye un plugin en los ficheros de instalación. Aunque, como se acaba de decir, Hammurapi contiene un plugin de Eclipse, la instalación es algo más complicada que la que es habitual en estos plugin:

(1) El primer paso es la descarga del archivo que contiene Hammurapi y que se encuentra disponible en el sitio web [86]. Con dicha descarga se obtiene un fichero ejecutable que hay que lanzar e instalar siguiendo los pasos que se indican por pantalla. (2) Tras la instalación del ejecutable, se debe abrir Eclipse e ir a Help → Software Updates. En la pestaña de Available Software, seleccionar “Add site...” e indicar en el apartado Location la ruta a la carpeta Eclipse Plugin contenida en la anterior instalación. (3) Tras esto se reinicia Eclipse y en las propiedades del proyecto que se desea analizar se activa la pestaña Checks. En el apartado Configuration URL se

pone la ruta al archivo java-plugin.xml que se encuentra en la carpeta del plugin de Eclipse que contiene la instalación del ejecutable de Hammurapi. (4) Tras esto, deberían aparecer en la pestaña problems los errores encontrados por Hammurapi

Tras seguir los pasos de instalación indicados anteriormente en las dos versiones disponibles de Hammurapi (5.7.0 y 6.3.0) en el momento de realizar esta memoria y haber probado ambas en diferentes versiones de Eclipse, en unas ocasiones se ha obtenido como resultado una instalación correcta del plugin (o al menos no se han mostrado errores tanto por pantalla como en el .log) pero sin conseguir obtener resultados en el apartado Problems y en otras ocasiones ni tan siquiera a aparecido el apartado Hammurapi en las propiedades del proyecto a analizar para poder efectuar el paso (3) del proceso de instalación.

3.3.3. jCosmo

jCosmo [80] detecta *Code Smells* en código fuente Java que pueden utilizarse para revisar la calidad del código analizado e indicar las regiones que podrían beneficiarse de una refactorización. La herramienta funciona en sistemas Linux.

“La extracción del modelo de código, implementado mediante ASF+SDF Meta-Environment, analiza el código Java y muestra las salidas en formato estándar Rigi (RSF). Los modelos extraídos contienen información sobre varias señales que sugieren que el código es problemático o que viola las directrices de programación (Code Smells). Estos smells se pueden utilizar para evaluar la calidad del código analizado. Además de los Code Smells, los modelos también contienen datos sobre la estructura del programa, tales como paquetes, clases y métodos, y las relaciones entre ellos”.

“Los modelos extraídos se visualizan mediante la herramienta de visualización Rigi. La estructura del programa se utiliza para crear un gráfico que puede ser manipulada para proporcionar los diferentes puntos de vista necesarios para diversas tareas de ingeniería inversa, tales como la comprensión del programa, la extracción de la arquitectura y la re-documentación”. Se puede encontrar más información de la herramienta en el artículo de Eva van Emden y Leon Moonen [88], de donde se han extraído estos fragmentos de código.

Vista esta descripción de la herramienta, jCosmo encaja perfectamente en el modelo de herramientas que se están analizando en esta memoria pero se incluye en este apartado porque aparece el problema de que ha sido imposible completar su instalación. Aunque el proceso de instalación aparece perfectamente detallado en el sitio web de jCosmo, ya en el segundo paso tras la descarga y descompresión del archivo surgen los primeros problemas y es que la herramienta intenta acceder a un sitio web que ya no se encuentra disponible. Se ha probado a hacer diferentes instalaciones por separado del elemento que la herramienta busca y luego continuar con las instrucciones o incluso a saltarse este paso pero, con esta opción, a la hora de ejecutar el comando **make** vuelven a surgir problemas, los cuales no se logran resolver. Estos errores se pueden observar en la figura que se adjunta a continuación. Primero se muestra el relacionado con el comando **make** y a continuación el intento que hace de acceso a una dirección web.

```
checking for working autoheader... found
checking for working makeinfo... missing
checking for wget... (cached) /usr/bin/wget
creating ./config.status
creating Makefile
creating pre-checks/Makefile
mendo10@ubuntu:~/Escritorio/jcosmo-bundle-0.2$ make
Making all in pre-checks
make[1]: se ingresa al directorio `/home/mendo10/Escritorio/jcosmo-bundle-0.2/pr
make[1]: *** No hay ninguna regla para construir el objetivo `configure.in', nec
make[1]: se sale del directorio `/home/mendo10/Escritorio/jcosmo-bundle-0.2/pre-
make: *** [all-recursive] Error 1
mendo10@ubuntu:~/Escritorio/jcosmo-bundle-0.2$ ./collect.sh
Obtaining a distribution of "aterm" via HTTP.
http://projects.cwi.nl/MetaEnv/aterm/aterm-1.6.6.tar.gz:
2010-10-22 11:20:20 ERROR 404: Not Found.

gzip: stdin: unexpected end of file
tar: Esto no parece un archivo tar
tar: Saliendo con fallos debido a errores anteriores
Obtaining and/or unpacking of "aterm-1.6.6" failed.
mendo10@ubuntu:~/Escritorio/jcosmo-bundle-0.2$
```

Intento de
acceso a
dirección web.

Figura 76. jCosmo. Muestra de los problemas encontrados en la instalación.

3.3.4. Sissy

Sissy [16], “Structural Investigation of Software Systems”, es una plataforma que permite ejecutar un análisis automatizado con el fin de evaluar la viabilidad de mantener un sistema orientado a objetos usando técnicas de análisis estático.

Aunque la herramienta parece de gran utilidad, no se ha podido probar que efectivamente tiene implementado todo lo aparece acerca de ella en [78] y [79] y que se detalla a continuación. El motivo de esta falta de pruebas es el complejo funcionamiento de la herramienta (Sissy trabaja con bases de datos), el cuál apenas aparece explicado en el archivo readme.txt que se incluye en el directorio de la herramienta que aparece tras su instalación.

Cada vez que la herramienta se ejecuta, hay que indicar primeramente el lenguaje con el que se va a trabajar, bien sea Java, C o Delphi y a continuación una serie de opciones. Para Java, hay que indicar el archivo que contiene los datos de la base de datos mediante la opción `-cfg <cfg-file>`. Si se omite esta opción la herramienta carga la base de datos configurada en el fichero `jdbc.cfg` que se incluye por defecto. La base de datos que se le indique es en la que se almacenarán los resultados salvo que se exprese lo contrario con el uso de la opción `-o <filename>`.

En la figura que se muestra a continuación se pueden observar todos los pasos que la herramienta sigue cuando efectúa un análisis. También se puede comprobar como su representación interna viene marcada por la utilización de un *object model*, haciendo uso de un ASG (abstract semantic graph) según se puede leer en el artículo *SiSSy, Recoder y Java2PCM* [94].

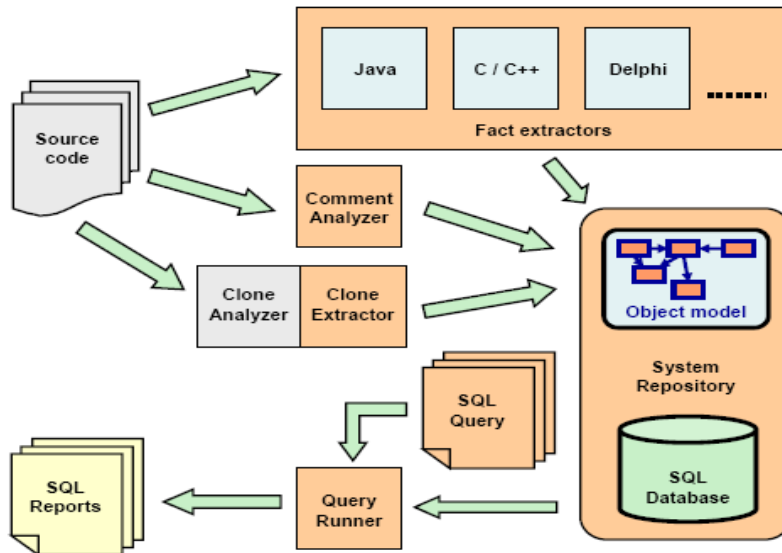


Figura 77. Sissy. Pasos que sigue para analizar código.

Sissy, como se muestra en la figura 78, puede analizar el código en busca de una gran cantidad de *Design Smells* como *god classes*, *data classes*, *cyclical dependencies between classes or packages...* y también puede realizar el cálculo de algunas métricas como *depth of inheritance*, *cyclomatic complexity*, *coupling*, *cohesion...* que pueden ser útiles a la hora de detectar problemas de diseño. Además permite la detección de *bad coding style* como se puede ver en [94].

Architectural level	Design level	Implementation level
Cyclical dependency between packages	Attribute overlap	Complex method
Dead imports	Constant redefinition	Cyclical dependency between source files
God package	Cyclical dependency between classes	Cyclical method calls
Import chaos	Dead attribute	Dead code
Mini-packages	Dead method	Duplicated code
Reversed package and inheritance hierarchies	General parameter	God file
Type name duplication	Generation conflict	God method
Unfinished code	God class, attribute form	Improper name length
	God class, method form	Inappropriate commenting
	Ignored abstraction	Informal documentation
	Inconsistent operations	Long parameter list
	Interface bypass	Misleading naming of source files
	Knows of derived	Object placebo, attribute form
	Late abstraction	Object placebo, method form
	Mini-classes	Overloaded file
	Orphan sibling attributes	Risky code
	Orphan sibling methods	Variables having constant value
	Permissive visibility, attribute form	Violation of naming convention
	Permissive visibility, method form	
	Polymorphic calls in constructor	
	Polymorphism placebo	
	Refused bequest, implementation form	
	Refused bequest, interface form	
	Similar unrelated abstractions	
	Simulated polymorphism	
	Violation of data encapsulation	

Figura 78. Sissy. Listado de problemas de código que busca la herramienta.

3.3.5. CrocoPat

CrocoPat [102] es una herramienta que manipula relaciones de cualquier aridad, incluyendo gráficos. La aplicación se basa en el diagrama de estructura de datos de decisión binaria (BDD), que es conocido por ser una representación compacta de las relaciones grandes en la verificación asistida por ordenador. CrocoPat es fácil de integrar con otras herramientas, ya que lee y escribe las relaciones desde varios archivos en el formato RSF de un determinado proyecto, utilizando la combinación de Doxygen y CCVisu.

La herramienta queda descartada de las herramientas a analizar debido a que su análisis relacional sobre lógica de predicados no tiene cabida en el estudio sobre detección de errores de diseño que se está llevando a cabo.

3.3.6. XRay

XRay [13] es un plugin *open source* de visualización de software; su objetivo es proveer a los administradores, desarrolladores y diseñadores de software de tres vistas polimétricas (vista de complejidad del sistema, de dependencia de clases y de paquetes) basadas en el código fuente. La herramienta permite al usuario identificar defectos de diseño, acoplamiento y cohesión a través de la navegación, el entendimiento, y el análisis de proyectos de pequeño y mediano tamaño en términos de clases, métodos y líneas de código fuente, además de la visualización de relaciones de herencia.

XRay obtiene los datos para las vistas a partir del código fuente del proyecto analizado, específicamente de las relaciones de dependencia y composición de clases, así como del número de líneas que conforman las entidades del proyecto.

Una vez que se procede a su análisis en búsqueda de elementos que detecten la presencia de alguno de los posibles defectos de diseño que se manejan, se observa que XRay es una herramienta que realiza los análisis de dependencias mencionados en el párrafo anterior, pero al contrario que otras herramientas similares analizadas anteriormente no resalta de ninguna forma específica los posibles *smells* que pueda incluir el código, por lo que queda descartada del catálogo de herramientas analizadas ya que deja todo en manos del usuario. Esta herramienta muestra información sobre las conexiones entre clases y paquetes, el número de métodos y las líneas de código que existen en una clase mediante la representación de rectángulos y la visualización del grosor y la altura de las clases mostradas, las relaciones de herencia existentes, o la visualización de los diferentes tipos de clases. Con este conjunto de visualizaciones el usuario puede guiarse para modificar si él cree necesario la estructura del proyecto de forma independiente a XRay.

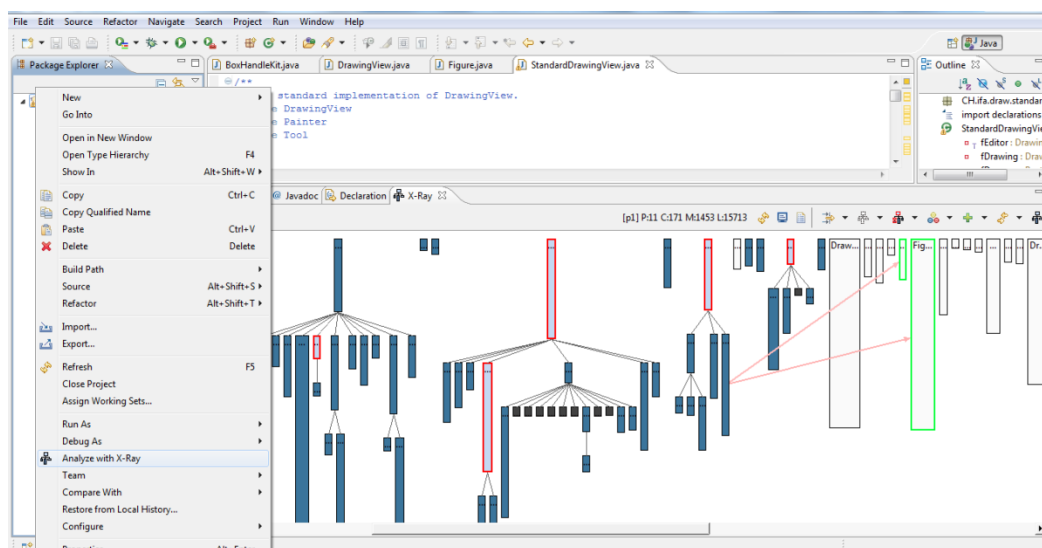


Figura 79. XRay. Visualización de XRay en Eclipse.

Capítulo 4

Tablas Completas

CONCLUSIONES

4.1. Target Artefact

A continuación se presenta la tabla resumen de la característica Target Artefact. Esta tabla engloba todas las tablas que se pueden encontrar en el apartado Target Artefact de cada una de las herramientas analizadas anteriormente.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERS.	TIPO DE REPRESENTACION
Analyst4j	Plugin Eclipse/Independiente	Source Code (Java)	NO	-
ArgoUML	Independiente	Source Code (JAVA, C, C++) Model (UML, Zargo, XMI, XML)	NO	-
Argus CodeWatch	Plugin Eclipse	Source Code (Java)	NO	-
CodePro Analytix	Plugin Eclipse	Source Code (Java)	NO	-
Cultivate	Plugin Eclipse	Source Code (Java)	NO	AST
Eclipse	Independiente	Source Code (Java, C, C++,Python)	SI	AST
FxCop	Independiente / Plugin Visual Studio	Microsoft .NET Assemblies	NO	Object Model, Graphs
inCode	Plugin Eclipse	Source Code (Java)	NO	-
jDeodorant	Plugin Eclipse	Source Code (Java)	NO	AST
JRefractory	Independiente / Plugin jEdit / Plugin NetBeans 3.6. / Plugin jBuilderX	Source Code (Java) Executable Code	NO	AST
M2 Resource Standard Metrics	Independiente Plugin de: Eclipse/ Visual Studio 6/ Visual Studio .NET/ jBuilder/ UltraEdit-Studio/ Return	Source Code (C,C++,C#, Java)	NO	Object Model
PMD	Independiente / Plugin Eclipse / Plugin jEdit / Plugin NetBeans / Otros IDE	Source Code (Java) (CPD admite Java, JSP, C, C++, Fortran, Ruby y PHP code)	NO	AST
Reek	Independiente (Linux)	Source Code (Ruby)	NO	AST
RevJava	Independiente	Executable Code	NO	AST
Roodi	Independiente	Source Code (Ruby)	NO	AST
STAN	Independiente / Plugin Eclipse	Executable Code	NO	-
SA4J	Independiente	Source Code (Java (.class)) Executable Code	NO	-

Tabla 76. Tabla que recoge la característica Target Artefact de Analyst4j a StyleCop.

HERRAMIENTA	ENTORNO DE TRABAJO	FORMATO DE ENTRADA	VERS.	TIPO DE REPRESENTACION
Structure101	Independiente / Plugin Eclipse / Plugin IntelliJ / Aplicación web	Source Code Executable Code	SI	-
StyleCop	Plugin Visual Studio / MSBuild	Source Code (C#)	NO	-
Together	Plugin Eclipse	UML Model Source Code (Java/C/C++/CORBA)	SI	Object Model
TRex	Independiente/Plugin Eclipse	Source Code (TTCN-3)	NO	AST
XRefactory	Plugin jEdit / Emacs / Xemacs	Source Code (JAVA/C/C++)	NO	-

Tabla 77 Tabla que recoge la característica Target Artefact de Structure101 a XRefactory.

4.2. Design Smells

A continuación se presenta la tabla resumen de la característica Design Smells. Esta tabla engloba todas las tablas que se pueden encontrar en el apartado Design Smells de cada una de las herramientas analizadas anteriormente.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
Analyst4j	Antipatterns Metrics Warnings	NO	NO	SI	SI	SI	SI	SI	SI
ArgoUML	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO
Argus CodeWatch	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO
CodePro Analytix	Metrics Warnings Bad Coding Style	NO	NO	SI	SI	SI	SI	SI	SI
Cultivate	Bad Smells Disharmonies Bad Coding Style Dependency Warnings	NO	NO	SI	SI	SI	SI	SI	SI
Eclipse	Bad Coding Style	NO	NO	NO	SI	SI	NO	NO	NO
FxCop	Bad Coding Style	-	-	-	-	-	-	-	-
inCode	Bad Smells / Disharmonies Metrics Warnings	SI	SI	SI	SI	SI	SI	SI	SI
jDeodorant	Bad Smells	NO	NO	NO	SI	SI	SI	SI	NO
JRefactory	Bad Coding Style Antipatterns	NO	NO	NO	SI	SI	SI	SI	NO
M2 Resource Standard Metrics	Metrics Warnings Bad Coding Style	NO	NO	SI	SI	SI	SI	SI	SI
PMD	Bad Coding Style Antipatterns	NO	NO	NO	SI	SI	SI	SI	NO
Reek	Bad Smells/Disharmonies Bad Coding Style	-	-	-	SI	SI	-	-	NO
RevJava	Bad Coding Style Bad Pattern Usage Bad smells / Disharmonies	SI	NO	SI	SI	SI	SI	SI	SI
Roodi	Metrics Warnings Bad Coding Style	-	-	-	SI	SI	-	-	-
STAN	Metrics Warnings Dependency Graph	SI	SI	SI	SI	SI	SI	SI	SI
SA4J	Dependency Warnings Antipatterns	NO	NO	SI	SI	NO	SI	SI	SI
Structure101	Dependency Graph	SI	SI	SI	SI	SI	SI	SI	SI

Tabla 78. Tabla que recoge la característica Design Smells de Analyst4j a Structure101.

HERRAMIENTA	TIPO DE ERROR	GRANULARIDAD					RELACIÓN		HERENCIA
		Sistema	Subsistema	Paquetes	Clases	Métodos	Inter	Intra	
StyleCop	Bad Coding Style	-	-	-	-	-	-	-	-
Together	Bad Smells / Disharmonies Metrics Warning Bad Coding Style	NO	NO	SI	SI	SI	SI	SI	SI
TRex	Bad Smells / Disharmonies Bad Coding Style	-	-	-	-	-	-	-	-
XRefactory	Bad Coding Style	NO	NO	NO	NO	SI	NO	NO	NO

Tabla 79. Tabla que recoge la característica Design Smells de StyleCop a XRefactory.

4.3. Activity

A continuación se presenta la tabla resumen de la característica Activity. Esta tabla engloba todas las tablas que se pueden encontrar en el apartado Activity de cada una de las herramientas analizadas anteriormente.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
Analyst4j	Especificación	Process description	Manual	Design Smell Specification	Textual
	Detección	Metrics-Based (Structural Metrics)	Interactive	Entity-metrics relationships	Numerical, textual
	Visualización	Diagrams	Fully-automated	Design Smell visualisation	Visual
ArgoUML	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Interactive	Correction Suggestions	Textual
Argus CodeWatch	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Manual	Correction Suggestions	Textual
CodePro Analytix	Especificación	Process Description	Interactive	Design Smell Specification	Textual
	Detección	Dependency analysis, metric-based	Interactive	entity-metrics y entity-smells relationships	Numerical, textual
	Corrección	Rules-based	Interactive	Correction suggestions	Textual
	Visualización	Dependency analysis	Interactive	Package dependencies and cycles	Visual
Cultivate	Análisis (jTransformer)	Model Extraction	Fully-Automated	-	-
	Detección	Metric-Based, Dependency Analysis	Interactive	Entity-smells relationships	Textual, Numerical
	Visualización	Dependency Analysis	Interactive	Package dependency graph	Visual
Eclipse	Detección	Rules-Based	Fully-automated	Entity-smells relationships	Textual
	Corrección	Rules-Based	Interactive	Correction Suggestions	Textual
FxCop	Especificación	Process Description	Interactive	Design Smell Specification	Textual
	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Rules-based	Interactive	Entity-smells relationships	Textual
	Corrección	Rules-based	Manual	Correction suggestions	Textual

Tabla 80. Tabla que recoge la característica Activity de Analyst4j a FxCop.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
inCode	Detección	Metric-Based	Interactive / Fully-automated	Entity-smells relationships	Textual
	Visualización	Visualisation-based	Interactive	Polymetrics views	Visual
	Corrección	Refactoring Suggestions	Interactive	Refactoring / Actions	Models
jDeodorant	Detección	Metrics-Based	Interactive	Entity-smells relationships	Textual, numerical
	Corrección	Refactoring Suggestions	Interactive	Refactorings / actions	Textual, Model
	Análisis de impacto	Metrics-Based	Interactive	Metric-Impact	Numerical
JRefractory	Análisis	Model Extraction	Fully-automated	-	Visual
	Detección	Rules-Based	Interactive	Entity-smells relationships	Textual
M2 Resource Standard Metrics	Detección	Metrics-Based	Interactive	Entity-metrics relationships	Textual, numerical
PMD	Especificación	Process description	Interactive	Detection Rules	Textual
			Manual	Detection Programs	Textual
	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Rules-Based	Interactive	Entity-smells relationships	Textual / Visual
Reek	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Metrics/Rules-Based	Fully-automated	Entity-smells relationships	Textual
RevJava	Análisis	Model Extraction	Fully-automated		
	Detección	Metrics / Rules / Pattern-Based	Interactive	Entity-smells relationships	Textual
Roodi	Especificación	Process Description	Manual	Detection Rules	-
	Análisis	Model Description	Fully-automated	-	-
	Detección	Metrics-Based (Structural Metrics) Rule-Based	Fully-automated	Entity-metrics relationships	Textual

Tabla 81 Tabla que recoge la característica Activity de inCode a Roodi.

HERRAMIENTA	Tipo de Actividad	Técnica	Automatización	Tipo de Resultado	Formato de salida
SA4J	Análisis	Model Extraction	Fully-automated	-	-
	Detección	Metrics/Pattern-Based	Fully-automated	Entity smells relationships	Numerical, Visual
	Visualización	Visualisation	Fully-automated	Dependency Diagrams, Skeleton	Visual
	Análisis de Impacto	Visualisation-Based	Interactive	Dependency Diagrams	Visual
STAN	Especificación	Process description	Interactive / Manual	Detection Metrics	Numerical
	Detección	Metric-Based	Interactive	Entity-metrics relationships	Textual, Numerical
	Visualización	Metric-Based	Interactive	Graphs, Diagram	Visual
Structure101	Detección	Metrics-Based	Fully-automated	Entity-Metrics relationships	Visual
	Visualización	Dependency Analysis	Fully-automated	Package dependencies and cycles	Visual
StyleCop	Especificación	Process Description	Interactive	Detection Rules	Textual
	Detección	Rules Based	Interactive	Entity-smells relationships	Textual
Together	Especificación	Process description	Interactive / Manual	Detection Metrics / Design Smell Specification	Numerical
	Detección	Metrics Based/ Rules Based	Fully-automated	Entity-smells / metrics relationships	Textual
	Corrección	Rules-Based	Interactive	Correction suggestions	Textual
TRex	Análisis	Model Extraction	Fully-automated	-	-
	Especificación	Process Description	Interactive	Detection Rules	Textual
	Detección	Pattern-Based / Metrics-Based	Fully-automated / Interactive	Entity-metrics relationships	Textual
	Corrección	Rules-Based	Interactive	Refactoring Suggestions	Textual
XRefactory	Detección	Rules-Based	Interactive	Entity-metrics relationships	Textual

Tabla 82. Tabla que recoge la característica Activity de SA4J a XRefactory.

4.4. Conclusiones

Tras el análisis exhaustivo de cada una de las herramientas y con la ayuda de las tablas globales que se han mostrado anteriormente, se pueden sacar algunas conclusiones acerca de aspectos comunes entre ellas, así como de rasgos que hacen diferenciables a determinadas herramientas.

Dentro de la característica *target artefact*, destaca el gran número de herramientas que han sido desarrolladas para poder ser integradas como *plugin* del IDE Eclipse, haciendo más fácil su instalación y uso que el ofrecido por otras herramientas que o bien son independientes o bien se integran en otros entornos de desarrollo (como por ejemplo jEdit, Emacs, Xemacs, Visual Studio, etc.). Cabe destacar que la mayoría de las herramientas admiten como formato de entrada código fuente Java o código ejecutable escrito en Java, aunque otras muchas analizan también C y C++. Existen también algunas herramientas como Reek o Roodi que analizan código fuente escrito en Ruby o como TRex que analiza código TTCN-3. El análisis de versiones es poco usual en las herramientas analizadas ya que principalmente se ocupan de trabajar con el proyecto que se tenga cargado en el momento de solicitar el análisis y no con repositorios de versiones o cargando diferentes versiones de un mismo proyecto. Por este aspecto son destacables las herramientas Eclipse, Structure101 y Together ya que estas sí permiten realizar dicho análisis de versiones. Respecto al tipo de representación se puede decir que predomina la transformación interna hacia un AST en aquellas herramientas que facilitan información respecto de este punto. Existe otra gran mayoría de ellas de las que se desconoce la transformación interna que efectúan y que se catalogan como desconocidas (en las tablas se identifica este aspecto mediante un guión), aunque la mayoría deberán tener su propio tipo representación interna del artefacto de entrada para trabajar con él de manera más rápida.

En la información reflejada en la tabla de la característica *design smells* se puede ver como las diferentes herramientas son capaces de detectar uno o varios tipos de error de los que se buscan en este trabajo pero predominan sobre todo *Metrics Warnings* y *Bad Coding Style*. Se pueden resaltar varias herramientas por ser de especial utilidad ya que son muy completas en el sentido de que son capaces de detectar una gran variedad de *smells*. Este es el caso, por ejemplo, de Together o Cultivate. Por el contrario existen otras que se limitan a facilitarnos información acerca de un solo tipo de *smell* como pueden ser Argus CodeWatch y jDeodorant. Tal vez sea el número de características que agrupa cada herramienta uno de los aspectos más importantes por los cuales un usuario se pueda decantar por la elección de una u otra para intentar mantener y mejorar su código, ya que de esta forma se evita tener una herramienta para tratar de mantener detectado cada tipo de error. Es interesante comentar que es posible que existan diferencias entre los resultados ofrecidos para un mismo código por distintas herramientas que identifican un mismo tipo de error debido a que sigan reglas, mediciones métricas u otro tipo de parámetros diferentes.

Con respecto a la granularidad propia de cada herramienta es necesario comentar que ha sido elaborada con base a la jerarquía de código Java que es la utilizada en el artículo que se usa como base, *J. Pérez et al.* [1]. Debido a esto, existen herramientas, como ya se ha comentado durante su análisis, que han sido diseñadas para analizar otro tipo de código (como TRex con TTCN-3, Reek o Roodi con Ruby, FxCop con .NET Assemblies, etc.) y que carecen de dicha jerarquía y por tanto no han sido evaluadas en este punto (Aparecen marcadas en su tabla correspondiente con un guión).

Del último punto importante de la taxonomía, la característica *activity*, se puede concluir que la gran mayoría de las herramientas no se centran únicamente en un tipo de actividad, a excepción

de M2 Resource Standars Metrics y XRefactory que sólo incorporan la actividad de detección, la cual, como es lógico, aparecerá en todas las herramientas analizadas. Hay que comentar en este punto que en el caso de M2 Resource Standars Metrics, la herramienta tiene muchas más opciones que las que se buscan en este trabajo, entre las que destaca el cálculo de gran cantidad de métricas que permitan a usuarios expertos el hacer uso de éstas para modificar el diseño de sus trabajos.

Un gran número de las herramientas incorporan la actividad de especificación, la cuál resulta de gran utilidad para aquellos usuarios que deseen ajustar la herramienta usada a las necesidades específicas de cada análisis, añadiendo, eliminado o variando los valores de una serie de métricas o reglas. Las herramientas más completas, por norma general, incorporarán además de las características ya comentadas, la actividad de visualización que las dotará de diagramas y gráficos útiles para la mejor comprensión de la estructura del código analizado. Otras herramientas que destacan por tener actividades diferentes a las del resto son jDeodorant y SA4j, que incorporan la actividad Análisis de impacto.

Capítulo 5

SITIO WEB

5.1. Introducción

5.1.1. Objetivos

El objetivo en este apartado es el de crear una página web en la que se muestra la información recopilada acerca de las herramientas de análisis de código que han sido analizadas con anterioridad. Además, se permite que los autores (u otros usuarios con conocimientos acerca de la herramienta) puedan actualizar los datos de la herramienta que deseen, para lo cuál se establece un control de usuarios.

5.1.2. Lista de tareas

1. Reuniones de proyecto. Al menos, el grupo se reúne diariamente por las mañanas de 10 a 13 y los martes y jueves por las tardes de 16 a 19.
2. Captura y especificación de requisitos.
3. Elección de tecnologías de desarrollo y herramientas.
4. Configuración del entorno de desarrollo y del sitio web de alojamiento del proyecto.
5. Realización del apartado de la memoria que corresponde a la creación del sitio web.

5.2. Especificación de requisitos

En la fase de especificación de requisitos se han seguido algunos de los modelos que aparecen en el artículo de M.J. Escalona y N. Koch [83]. Según esto, el proceso de especificación de requisitos se puede dividir en tres grandes actividades [23]:

- Captura de requisitos.
- Definición de requisitos.
- Validación de requisitos.

En la figura se presenta el proceso de ingeniería de requisitos que incluye estas tres actividades. Para la representación se ha usado la notación de diagrama de actividades propuesta en UML

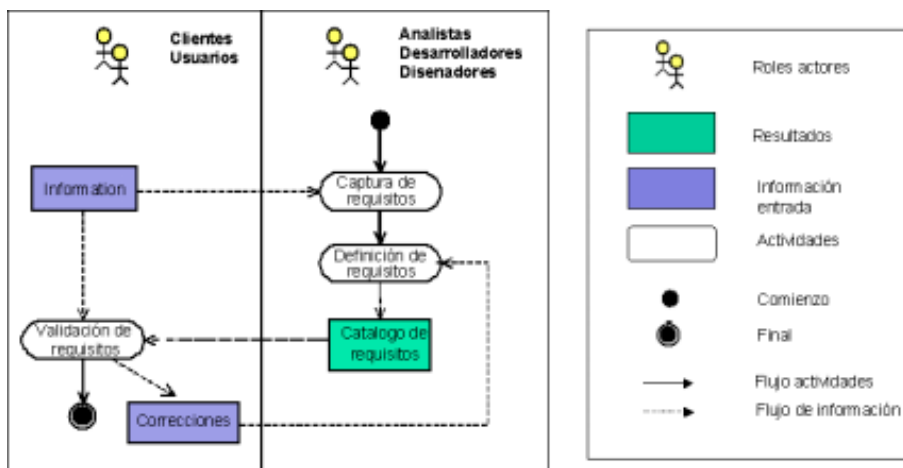


Figura 80. Proceso de ingeniería de requisitos.

5.2.1. Captura de requisitos

La captura de requisitos es la actividad mediante la que el equipo de desarrollo de un sistema extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema. Dado que los requisitos del sitio web que se va a desarrollar han sido dados por los tutores de este proyecto a través de las diferentes reuniones mantenidas con ellos y en estas no se ha seguido ningún patrón específico, se ha decidido indicar los requisitos finales en forma de entrevista.

- **Entrevista**

Las entrevistas permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural. Básicamente, la estructura de la entrevista abarca tres pasos: identificación de los entrevistados, preparación de la entrevista y realización de la entrevista y documentación de los resultados. Como ya se ha comentado, los requisitos no han sido facilitados de ninguna forma específica y la entrevista no se realizó como tal, debido a esto, se ha decidido simularla. Por ello, no se seguirán exactamente estos tres pasos básicos.

En esta simulación los entrevistados son Yania Crespo y Javier Pérez (Tutores de este proyecto) y los entrevistadores son los autores del proyecto, Javier Sobrino y Alberto Mendo.

¿Cuál es el propósito?

Poder mostrar el análisis de las herramientas realizado, como se ha comentado en el apartado 5.1.1.

¿Qué es lo que se debe mostrar?

Es necesario mostrar el propósito de cada herramienta y sus respectivas tablas, así como una descripción correspondiente a cada apartado en los que se divide la tabla.

¿A qué tipo de usuarios va dirigida la web?

Principalmente a usuarios que necesiten información sobre herramientas de análisis de código, estos generalmente serán programadores profesionales. Debido a esto, se considera avisar a los autores de las herramientas acerca de la existencia de la web.

¿Cómo puede un usuario interactuar con la web?

Todo el que lo desee podrá acceder a la web y navegar en ella, pero no todos los usuarios tendrán los mismos derechos. El autor de cada herramienta tiene que poder modificar los aspectos que desee relacionados con ella, aunque también podrán modificar dichos aspectos los usuarios registrados que posean conocimientos acerca de la herramienta. Se tiene que permitir realizar comentarios para cada una de ellas.

¿Se desea algún tipo de diseño específico?

Debido al tipo de usuarios que se espera que tenga la web, el diseño de la misma deberá ser de carácter formal y de uso simple, permitiendo un acceso rápido a cada uno de los

apartados para así ahorrar el mayor tiempo posible. Como a determinados usuarios se les va a dar permiso para la modificación de tablas, lo más recomendable es hacer una web dinámica.

5.2.2. Definición de requisitos

5.2.2.1. Introducción

También para la actividad definición de requisitos, en el proceso de ingeniería de requisitos hay un gran número de técnicas propuestas. Algunas se pueden ver en [83]. De todas las propuestas, aquí se han tratado de seguir principalmente dos, la de Plantillas o Patrones y la de Casos de Uso.

Plantillas o patrones: Esta técnica, recomendada por varios autores, tiene por objetivo el describir los requisitos mediante el lenguaje natural pero de una forma estructurada. Una plantilla es una tabla con una serie de campos y una estructura predefinida que el equipo de desarrollo va cumplimentando, usando para ello el lenguaje del usuario. Para la realización de la definición de requisitos se ha utilizado el programa REM [84], desarrollado por la Universidad de Sevilla específicamente con el fin de dar soporte a la fase de Ingeniería de Requisitos.

Una clasificación de requisitos relevante en sistemas web puede ser la siguiente:

- **Requisitos de datos**, también denominados requisitos de contenido, requisitos conceptuales o requisitos de almacenamiento de información. Éstos requisitos responden a la pregunta de qué información debe almacenar y administrar el sistema.
- **Requisitos de interfaz** (*al usuario*), también llamados en algunas propuestas requisitos de interacción. Responden a la pregunta de cómo va a interactuar el usuario con el sistema.
- **Requisitos navegacionales**, recogen las necesidades de navegación del usuario.
- **Requisitos de personalización**, describen cómo debe adaptarse el sistema en función de qué usuario interactúe con él y de la descripción actual de dicho usuario.
- **Requisitos no funcionales**, son por ejemplo los requisitos de portabilidad, de reutilización, de entorno de desarrollo, de usabilidad, de disponibilidad, etc.

Antes de entrar a definir los requisitos y presentar los diferentes casos de uso, es necesario establecer los distintos grupos de usuario que van a interactuar con el sistema. Estos rangos de usuario se pueden extraer de las respuestas dadas en la entrevista mantenida.

- **Administrador:** encargado de supervisar la web y conceder permisos de usuario experto.
- **Experto:** usuario con privilegios de poder publicar un determinado análisis de alguna herramienta en concreto y poder comentar los ya existentes.
- **Registrado:** rango de usuario al que le esta solamente permitido comentar mediante un campo de texto los análisis realizados por los administradores o por los autores (Usuarios expertos).
- **Invitado:** usuarios que entran en la web sin ningún tipo de privilegio.

A continuación se pueden observar las actividades que los diferentes usuarios descritos anteriormente pueden realizar con el sitio web.

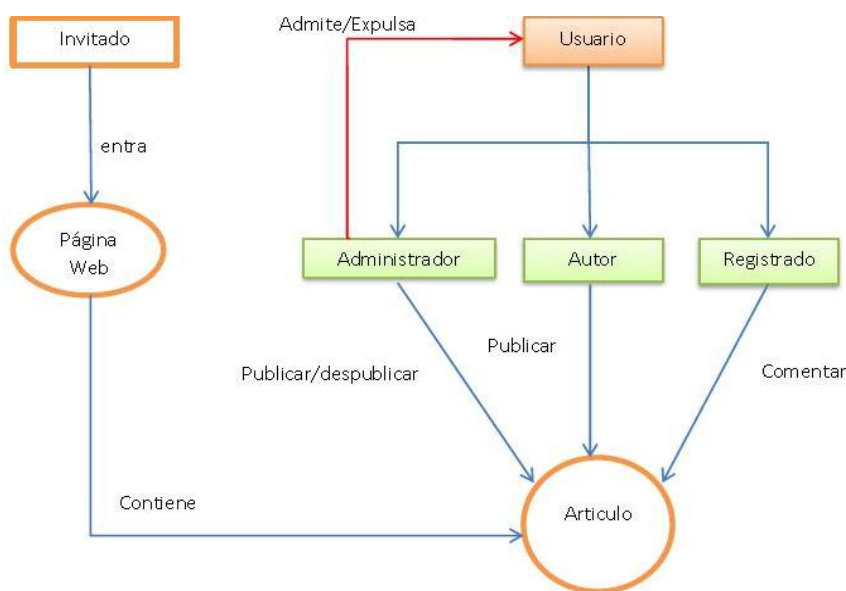


Figura 81. Interacción Web y usuarios.

5.2.2.2. Requisitos de datos

RQDatos-1	Registro
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	<p>El sistema deberá permitir el registro de un usuario almacenando los siguientes datos:</p> <p>Nombre y Apellidos: Nombre y apellidos del usuario.</p> <p>Nombre de Usuario: Identificador único de cada usuario.</p> <p>Contraseña: Clave elegida por el usuario que será requerida para autenticar a un usuario.</p> <p>E-mail: Dirección del correo electrónico del usuario registrado.</p>
Comentarios	Requisitos de información de usuario.

RQDatos-2	Almacenar datos
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá ser capaz de almacenar los datos de las tablas, de los comentarios y de los usuarios.
Comentarios	Ninguno

5.2.2.3. Requisitos de interfaz

RQInterfaz-1	Requisitos usuario invitado
Versión	1.0 (28/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá controlar que un usuario invitado sólo pueda visualizar los elementos de la web hasta que decida registrarse. El sistema deberá permitir loguearse o en su defecto acceder al formulario de registro.
Comentarios	Ninguno

RQInterfaz-2	Requisitos usuario registrado
Versión	1.0 (28/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá habilitar la opción de que el usuario pueda realizar comentarios. El sistema deberá permitir poder solicitar la opción de ser usuario Experto.
Comentarios	Ninguno

RQInterfaz-3	Requisitos usuario experto
Versión	1.0 (28/10/2010)
Dependencias	Previamente se tiene que ser usuario registrado.
Descripción	El sistema deberá permitir publicar y despublicar nuevos análisis de herramientas diferentes a las ya existentes. El sistema deberá permitir modificar los análisis publicados por el usuario o a los que se le haya dado permiso. El sistema deberá habilitar la opción de que el usuario pueda realizar comentarios.
Comentarios	Ninguno

RQInterfaz-4	Requisitos administrador
Versión	1.0 (28/10/2010)
Dependencias	Ninguno
Descripción	<p>El sistema deberá permitir publicar y despublicar nuevos análisis de herramientas diferentes a las ya existentes.</p> <p>El sistema deberá permitir modificar cualquier análisis.</p> <p>El sistema deberá habilitar la opción de que el usuario pueda realizar comentarios.</p> <p>El sistema deberá permitir gestionar a todos los usuarios, permitiendo la creación y eliminación. Deberá permitir también la modificación de datos y permisos.</p>
Comentarios	Ninguno

5.2.2.4. Requisitos Navegacionales

RQNav-1	Visualizar análisis
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá permitir el acceso a la visualización de los análisis realizados de las herramientas.
Comentarios	Ninguno

RQNav-2	Solicitar ser Usuario Experto
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá dar a un usuario registrado la opción de solicitar ser un usuario experto para la modificación de alguna de las tablas. El botón estará habilitado cuando corresponda.
Comentarios	Ninguno

RQNav-3	Comentar los análisis
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá mostrar un apartado que permita realizar comentarios en las herramientas analizadas. El apartado se habilitará cuando corresponda.
Comentarios	Ninguno

5.2.2.5. Requisitos de personalización

RQPers-1	Adaptación al usuario
Versión	1.0 (28/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá habilitar y deshabilitar botones y herramientas en función del rango del usuario.
Importancia	Importante
Comentarios	Requisitos de personalización.

5.2.2.6. Requisitos no funcionales

NFR-0001	Facilidad de navegación
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá ofrecer al usuario una navegación a través de los diferentes contenidos de forma rápida e intuitiva
Importancia	Importante
Comentarios	Ninguno

NFR-0002	Accesibilidad (navegadores)
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá permitir que la web pueda ser visualizada con cualquier navegador de forma correcta (Explorer, Firefox, Chrome, Opera, etc.).
Importancia	Importante
Comentarios	Ninguno

NFR-0003	Disponibilidad
Versión	1.0 (26/10/2010)
Dependencias	Ninguno
Descripción	El sistema deberá estar disponible para todo el visitante que desee poder observar los análisis realizados de las herramientas.
Importancia	Importante
Comentarios	Ninguno

5.2.2.7. Casos de uso

- Actores participantes en los casos de uso:

ACT-0001	Autor
Versión	1.0 (26/10/2010)
Fuentes	Usuario Experto
Descripción	Este actor representa un cliente con rango de usuario Experto como autor de una de las herramientas.
Comentarios	El autor será avisado de la realización de la web y se le asignará automáticamente el rango de Usuario Experto.

ACT-0002	Experto
Versión	1.0 (26/10/2010)
Fuentes	Usuario Experto
Descripción	Este actor representa un cliente con rango de usuario Experto.
Comentarios	El usuario puede modificar una herramienta sin ser el autor de la misma.

ACT-0003	Registrado
Versión	1.0 (26/10/2010)
Fuentes	Usuario Registrado
Descripción	Este actor representa un usuario Registrado.
Comentarios	Puede realizar únicamente comentarios en la página web.

ACT-0004	Invitado
Versión	1.0 (26/10/2010)
Fuentes	Usuario Invitado
Descripción	Este actor representa <i>un Usuario Invitado</i> .
Comentarios	Este actor únicamente puede navegar por la web.

- Casos de Uso considerados:

UC-0001	Registro de Usuario	
Versión	1.0 (26/10/2010)	
Autores	Alberto Mendo Javier Sobrino	
Fuentes	Javier Pérez Yania Crespo	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando inicio de registro	
Secuencia normal	Paso	Acción
	1	El actor Desconocido (ACT-0004) inicia el registro pulsando, Nuevo usuario.
	2	El sistema muestra al usuario los campos necesarios para llevar a cabo el registro: Nombre y apellidos, username, contraseña, compañía y email.
	3	El actor Desconocido (ACT-0004) rellena los campos marcados requeridos por el sistema para el registro. Cuando llegue al final del mismo, pulsará ENVIAR para efectuar la operación.
	4	El sistema conectará con la base de datos de los usuarios para registrar al nuevo usuario.
	5	Si los datos introducidos han sido validados como correctos en la base de datos, el sistema mostrará el mensaje de registro con éxito.
Excepciones	Paso	Acción
	4	Si el nombre y apellidos coincide con alguno de los ya almacenados en la base de datos, el sistema mostrará de nuevo los campos de registro para que se modifique el campo de nombre y apellidos , a continuación este caso de uso continúa
	4	Si el username introducido coincide con alguno de los ya almacenados en la base de datos, el sistema mostrará de nuevo los campos de registro para que se modifique el campo username., a continuación este caso de uso continúa
	4	Si el email introducido coincide con alguno de los ya almacenados en la base de datos, el sistema el sistema mostrará de nuevo los campos de registro para que se modifique el campo email, a continuación este caso de uso continúa

UC-0002	Login de usuario	
Autores	Alberto Mendo Javier Sobrino	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario desee loguearse en el sistema.	
Secuencia normal	Paso	Acción
	1	El actor Desconocido (ACT-0004) deberá rellenar los dos campos solicitados en la página principal, username y contraseña, para proceder a su autenticación una vez pulsado ENVIAR.
	2	El sistema comprobará que los dos campos introducidos por el usuario son correctos y coinciden con los almacenados en la base de datos.
	3	El sistema eliminara de la página principal los campos para loguearse y aparecerán en su lugar los datos del usuario identificado.
Excepciones	Paso	Acción
	2	Si el username no está registrado en la base de datos, el sistema enviará un mensaje de error. Username no existe!, a continuación este caso de uso continúa
	2	Si la contraseña introducida es errónea, el sistema enviará un mensaje de error. Contraseña Incorrecta!, a continuación este caso de uso continúa

UC-0003	Solicitud Usuario Experto	
Autores	Alberto Mendo Javier Sobrino	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se pulse, Expert User?	
Precondición	Ser un usuario registrado.	
Secuencia normal	Paso	Acción
	1	El actor Pepe (registrado) (ACT-0003) solicita ser usuario experto al pulsar: Expert User?
	2	El sistema abre una ventana donde el usuario registrado deberá rellenar un cuestionario dónde explique brevemente porque quiere modificar algún análisis existente.
	3	El actor Pepe (registrado) (ACT-0003) rellenará el cuestionario y pulsará al botón enviar.
	4	El sistema reenviará la solicitud a los administradores mediante correo interno para que estos decidan.
Excepciones	Paso	Acción
	1	Si el usuario no está registrado o logueado, el sistema mandará el mensaje de error, Usuario no registrado., a continuación este caso de uso queda sin efecto

UC-0004	Agregar comentario	
Autores	Alberto Mendo Javier Sobrino	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando el usuario registrado o experto desee realizar un comentario en alguna de las herramientas analizadas.	
Precondición	El usuario debe estar registrado o ser Experto.	
Secuencia normal	Paso	Acción
	1	El actor Pepe (registrado) (ACT-0003) Desea añadir un comentario en el apartado correspondiente a una herramienta analizada. Para ello seleccionará del menú de herramientas disponible aquella que vaya a comentar.
	2	El sistema permitirá que el usuario realice el comentario mediante un editor de texto.
	3	El actor Pepe (registrado) (ACT-0003) redactará su comentario en el editor habilitado y una vez finalizado pulsará ENVIAR.
	4	El sistema actualizará la tabla de comentarios realizados, colocando el último comentario debajo del último existente.

UC-0005	Modificar tabla de una herramienta	
Autores	Alberto Mendo Javier Sobrino	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un usuario experto quiere modificar una de las tablas existentes para aportar un conocimiento específico.	
Precondición	El usuario debe tener rango Experto.	
Secuencia normal	Paso	Acción
	1	El actor Jesús (Experto) (ACT-0002) desea modificar una tabla de una herramienta en concreto. Para ello seleccionará del menú de herramientas disponible aquella que vaya a modificar.
	2	El sistema habilitará un botón, EDITAR encima del análisis (tabla) para que el usuario Experto pueda entrar en el editor.
	3	Si el usuario pulsa EDITAR, el actor Jesús (Experto) (ACT-0002) podrá manipular los datos existentes en la tabla para poder modificarlos. Una vez modificados el usuario podrá pulsar GUARDAR si desea que se hagan efectivas las modificaciones. O pulsar CANCELAR si no quiere que se guarden
	4	Si la tabla ha sufrido alguna modificación, el sistema actualizará la información de la tabla.

5.2.3. Validación de requisitos

Se define la etapa de validación de requisitos como el proceso que tiene como misión demostrar que la definición de los requisitos realizada anteriormente define realmente el sistema que el usuario necesita. En este trabajo, son los tutores los que mediante una serie de reuniones o auditorías dan validez a los requisitos.

Los requisitos que se han presentado anteriormente son los ya validados. Las reuniones y auditorías de validación se han llevado a cabo a la vez que la etapa de captura de requisitos donde se iban listando y se aceptaban o descartaban en el momento.

5.3. Introducción a los CMS

Un sistema de gestión de contenidos (en inglés “*Content Management System*”, **CMS**) es un programa que permite crear una estructura de soporte framework para la creación y administración de contenidos, principalmente en páginas web, por parte de los participantes.

Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que dar formato de nuevo al contenido, además de permitir la publicación fácil y controlada en el sitio a varios usuarios que actúan como editores.

El gestor de contenidos genera páginas dinámicas interactuando con el servidor para generar la página web bajo petición del usuario, con el formato predefinido y el contenido extraído de la base de datos del servidor. Esto permite gestionar, bajo un formato estandarizado, la información del servidor, reduciendo el tamaño de las páginas para descarga y reduciendo el coste de gestión del portal con respecto a una página estática, en la que cada cambio de diseño debe ser realizado en todas las páginas, de la misma forma que cada vez que se agrega contenido tiene que maquetarse una nueva página HTML y subirla al servidor.

El gestor de contenidos se aplica generalmente para referirse a sistemas de publicación, pudiendo subestimarse las funcionalidades de soporte, en detrimento de las funcionalidades relacionadas con la optimización de los tiempos de publicación.

El posicionamiento en buscadores está altamente relacionado con el volumen de contenidos de un portal y con la forma en la que éste se presenta. Es importante tener eso en cuenta para la estructura del portal para garantizar un correcto posicionamiento orgánico.

Dependiendo de la plataforma escogida se podrá elegir entre diferentes niveles de acceso para los usuarios; yendo desde el administrador del portal hasta el usuario sin permiso de edición, o creador de contenido y, por supuesto, el visitante. Dependiendo de la aplicación podrá haber varios permisos intermedios que permitan la edición del contenido, la supervisión y reedición del contenido de otros usuarios, etc.

El sistema de gestión de contenidos controla y ayuda a manejar cada paso de este proceso, incluyendo las labores técnicas de publicar los documentos en uno o más sitios [95].

¿Por qué elegir un CMS?

La elección de un CMS para la exposición del trabajo realizado viene provocada principalmente por la búsqueda de un sistema sencillo, que se amolde a la especificación de requisitos definidos con anterioridad, contruidos a partir de la entrevista mantenida con los tutores. Esta entrevista marca tres aspectos que se consideran importantes para la elección de un sistema de gestión de contenidos: la posibilidad de mostrar los resultados obtenidos de una forma sencilla, la gestión de usuarios pudiendo otorgar diferentes rangos con el fin de delimitar determinadas funciones en la web y la condición de crear un portal dinámico. Estas tres características u objetivos descritos cumplen a la perfección con la definición de CMS dada anteriormente, además de incluir otras muchas posibilidades para la creación de sitios web.

5.4. Joomla!

Dentro del amplio abanico de aplicaciones CMS existentes y que se adapten a las características que el sitio web requiere, se ha elegido Joomla! por estar considerado como una aplicación moderna y sencilla, que se encuentra en continuo desarrollo y tiene una buena crítica y aceptación entre miles de usuarios. Además ofrece un gran número de módulos y plugins que se pueden instalar para el propio beneficio del usuario.

“Joomla! es una aplicación de código abierto programada mayoritariamente en PHP bajo una licencia GPL (Licencia Pública General de GNU, orientada principalmente a proteger la libre distribución, modificación y uso de software). Este administrador de contenidos puede trabajar en Internet o intranets y requiere de una base de datos MySQL, así como, preferiblemente, de un servidor HTTP Apache” [96].

Instalación y configuración de Joomla!

Para llevar a cabo la instalación de Joomla! es necesario conocer unos requisitos previos que harán posible la operación. El usuario debe asegurarse antes de acometer dicha instalación de que su servidor local o su servidor web (servicio de hosting) debe ofrecer las siguientes características:

- PHP 4.2.x o superior.
- MySQL 3.23.x o superior.
- Apache 1.13.19 superior.

Además, debe asegurarse de que el módulo PHP tenga instalado el soporte para MySQL, XML y Zlib para que Joomla! funcione de manera exitosa. Se recomienda el uso de un servidor Apache para instalar Joomla! en Windows.

Joomla! puede utilizarse con los principales navegadores web, incluyendo: Internet Explorer, Mozilla Firefox, Google Chrome y Opera. Estos navegadores se aprovechan de la interfaz Administrativa de Joomla! [96].

Instalación.

En este apartado se detallan los pasos que hay que seguir para realizar una correcta instalación de Joomla!.

- (1) Se deberá disponer de un servidor web o un servidor independiente local que contenga los servicios descritos en el apartado anterior (PHP, MySQL y Apache). Para la demostración

siguiente se ha simulado la instalación en local, que es muy similar a la de un servidor web. Para ello se descarga e instala la herramienta Wamp [92] que ofrece las características requeridas. A modo de información, decir que existe una gran variedad de herramientas de este tipo para llevar a cabo la simulación de servidores y que la mayoría de ellas ofrecen garantías de un buen funcionamiento con Joomla!.

- (2) Una vez que se haya instalado correctamente habrá que dirigirse a phpMyAdmin a través de Wamp. En el caso de estar en un servidor web, se deberá entrar en el panel proporcionado por la empresa del hosting contratado y entrar igualmente en phpMyAdmin o el gestor de bases de datos equivalente.

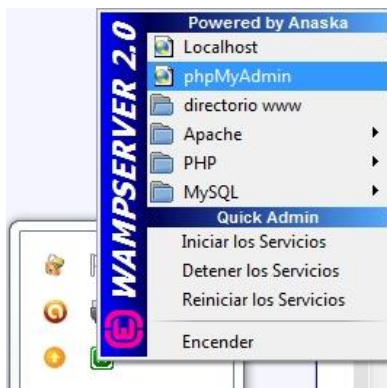


Figura 82. Instalación de Joomla!. Lanzar phpMyAdmin en Wampserver.

(3) Una vez dentro se creará una base de datos (en este caso con el nombre “pfc”) que será la que contendrá las tablas del sitio web y que se deberá referenciar posteriormente durante la instalación como tal de Joomla!.

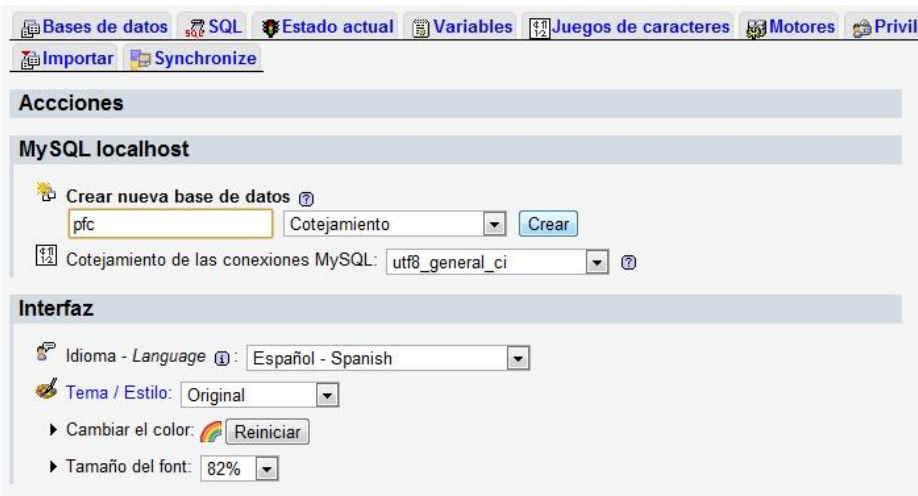


Figura 83. Instalación de Joomla!. Vista general de phpMyAdmin.

- (4) El siguiente paso a dar es ubicar los archivos de Joomla! en el lugar correcto para iniciar su instalación. Para ello se deberá tener descargado el paquete que contiene la herramienta. Si no se dispone de él, se podrá encontrar en <http://www.joomlaspanish.org>, en su sección de descargas.

Una vez se disponga de Joomla!, (5) se irá al directorio donde se instaló Wamp y de ahí a la carpeta `www`, dónde se creará un directorio (denominado “PFC” en este ejemplo de instalación). En el caso de instalarlo en un servidor web remoto habrá que dirigirse al directorio raíz y buscar la carpeta equivalente a `www`. El directorio raíz puede estar ubicado en distintos lugares o rutas dependiendo del sistema operativo o del paquete del servidor que se utilice. Las denominaciones más comunes son `httpdocs`, `httpd`, `www-data`, `www` o `public_html`, pero es algo que puede variar y de lo que seguramente quien proporcione el hospedaje podrá facilitar información. (6) Tras crear el directorio, se deberá descomprimir el archivo que contiene Joomla! dentro de la carpeta especificada anteriormente (`./wamp/www/PFC`). Si se trata de una instalación remota, este proceso requerirá normalmente del uso de un programa cliente de FTP (File Transfer Protocol) o del panel de control proporcionado por quien sirva el hospedaje si está disponible (normalmente será Cpanel aunque existen muchos diferentes). Lo destacable es que usando Cpanel (o similares), hay que subir los archivos/carpetas de Joomla! a un lugar que esté dentro del espacio web que se esté usando. En ocasiones, normalmente con servidores basados en Linux que permitan el acceso a la línea de comandos, también será posible subir el paquete comprimido al espacio web y descomprimirlo directamente desde ahí mismo (Éste es un proceso mucho más rápido).

- (7) Una vez que se haya realizado la descompresión de todas las carpetas y se hayan ubicado en el lugar correcto, se procederá a la instalación de Joomla! en el servidor. Para ello se introducirá en la barra de direcciones del navegador la ruta `http://localhost/PFC/` en el caso de instalación local, y la url del dominio para el caso de la instalación remota. Una vez que el navegador haya abierto la dirección indicada, aparecerá la ventana inicial de instalación, (8) dónde se deberá seleccionar el idioma que se desea para la instalación y pulsar en siguiente.



Figura 84. Instalación de Joomla!. Ventana de selección de idioma para la instalación.

- (9) Tras seleccionar el idioma, aparecerá una ventana de comprobación, en la que se advierte que si alguno de los parámetros del recuadro superior se encuentra en rojo, el funcionamiento de Joomla! puede que no sea correcto. Si se obtienen los parámetros deseados (lo ideal es obtener todos en verde), se continúa con la instalación haciendo clic en siguiente.



Figura 85. Instalación de Joomla!. Ventana de comprobación previa.

- Tras la comprobación, (10) se mostrará una ventana con la licencia y tras pulsar en siguiente (11) aparecerá una nueva ventana para la configuración de la base de datos y el hosting, dónde se deberán especificar los campos que se pueden observar en la figura 86. Los datos introducidos en la figura se corresponden con la instalación en local. Si la instalación es remota se deberá introducir el nombre del servidor y el nombre de usuario de acuerdo con las instrucciones o datos proporcionados por la empresa del hospedaje. En el nombre de la base de datos se tecleará el nombre de la creada en el paso (3). Si alguno de los datos es erróneo, al pulsar en siguiente aparecerá una ventana advirtiéndose de que se ha producido un error y se deberá volver al paso anterior.

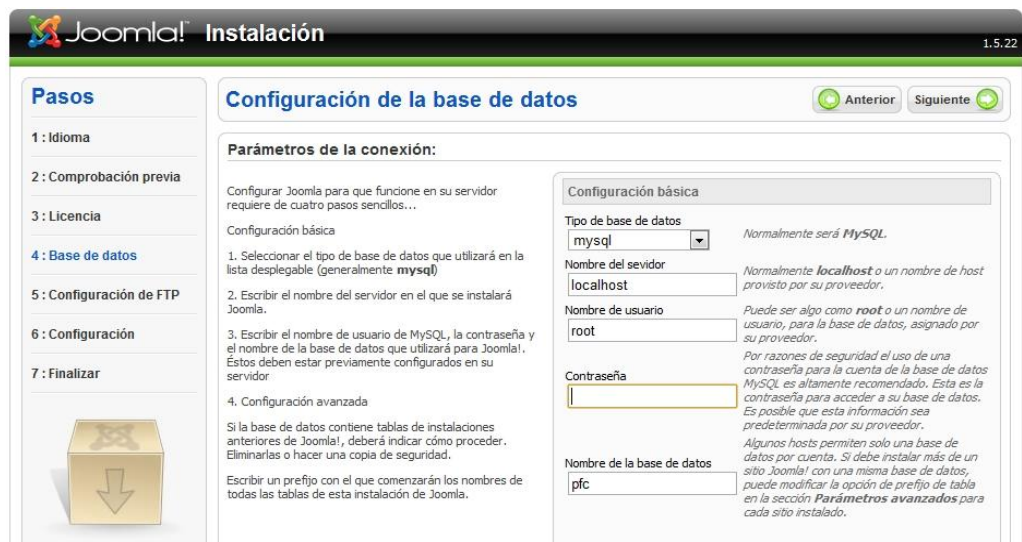


Figura 86. Instalación de Joomla!. Ventana de configuración de la base de datos.

- (12) En la siguiente ventana aparecerá la opción de habilitar una capa FTP para la gestión de archivos dentro del propio Joomla!. Para la instalación local no es necesaria y para la instalación en un servidor, los datos tienen que haber sido facilitados por la empresa de hospedaje.

Figura 87. Instalación de Joomla!. Ventana de configuración principal del sitio web.

- Para ir finalizando con la instalación de Joomla!, (13) aparecerá una ventana dónde se deberá introducir el nombre del sitio web, un correo electrónico, una contraseña y la confirmación de la misma. Por último existe la opción de instalar los datos de ejemplo predeterminados, que ayudarán a los usuarios inexpertos a familiarizarse con Joomla!. Todo esto se muestra en la figura 87.

- (14) El último paso para completar la instalación será borrar del directorio raíz, dónde antes se había descomprimido el contenido de Joomla en los pasos 5 y 6, la carpeta *installation* ya que por motivos de seguridad, Joomla! no permite avanzar en la instalación hasta eliminarlo.

Figura 88. Instalación de Joomla!. Último paso en el que se pide borrar el directorio de instalación.

Una vez borrada la carpeta del directorio se podrá acceder tanto a la portada del sitio, que simplemente contendrá una de las plantillas predeterminadas que trae consigo Joomla!, como a la administración del portal (para ello se indica la ruta o dirección web seguida de /administrator).

5.5. Diseño

Todo buen diseño debe integrarse con el contenido. Se ha buscado una organización de los contenidos que sea coherente; con unos menús de navegación claros que faciliten una rápida navegación por el sitio web. A esto se une que el uso del portal web a tratado de simplificarse al máximo, intentando resultar sencillo y cómodo para todos sus usuarios. Se ha tratado de garantizar, en la medida de lo posible, el correcto funcionamiento y visionado de la web en los diferentes navegadores existentes (Firefox, Internet Explorer, Google Chrome, Opera, etc.).



Figura 89. Design Smell Management Tools. Página de inicio del sitio web.

El diseño final del sitio web que se ha desarrollado puede verse en la *figura 89*, donde también se puede observar que se han incluido dos módulos de menús que aunque puedan resultar repetitivos, facilitan la navegación al usuario; un menú horizontal situado en la parte superior, bajo el logotipo del portal web, y otro vertical situado en el lateral izquierdo. El menú horizontal incluye cinco apartados:

- **Inicio:** Un botón de inicio que permite volver a la página de bienvenida de la web siempre que se desee.

- **Estudio Previo:** Menú desplegable que incluye una descripción de los tres principales apartados descritos en el capítulo dedicado a la **Taxonomía** en esta memoria, *Target Artefact*, *Design Smells* y *Activity*.
- **Design Smells:** Apartado que incluye una introducción a los *Design Smells* a partir de un menú desplegable que contiene enlaces a las definiciones y listados de *Antipatterns*, *Bad Smells* y *Disharmonies* utilizados para realizar los análisis y que se encuentran en esta memoria.
- **Herramientas Analizadas:** Menú desplegable en el que se encuentran enlaces a todas las herramientas analizadas en este trabajo. La distribución que sigue el análisis de una herramienta se describe a continuación y puede verse en la *figura 90*:
 - (1) Una introducción a la herramienta, (2) las tablas completas para cada uno de los apartados de la Taxonomía descrita en esta memoria, junto a las que se incluye (3) un botón de información que permite ir a la descripción de lo que cada tabla contiene (es decir, a la descripción de las características *Target Artefact*, *Design Smells* o *Activity* incluida en el apartado Estudio Previo). (4) Además de esto, los nombres de la tabla enlazan con (5) la parte de la página que describe el contenido de la tabla y (6) la imagen de presentación de cada herramienta incluye un enlace hacia el sitio web oficial de cada una. También se incluyen (7) dos flechas al final del análisis que permiten navegar entre herramientas, moviéndose a la anterior o a la siguiente en caso de que existan. Debajo de estas flechas se localiza la sección de comentarios (8).
- **Otras herramientas:** Apartado que incluye información acerca de las herramientas que no han podido analizarse en este proyecto.

El menú vertical incluye, además de los cinco apartados que contiene el menú horizontal, un apartado para el **login de usuario** a través del cuál cualquier usuario registrado puede identificarse para poder realizar comentarios en la web y otras operaciones que variarán en función de su rango de usuario; dichas operaciones están descritas en el apartado. El apartado para el login de usuario incluye además las opciones de recordar contraseña y recordar usuario en caso de que un usuario registrado haya olvidado alguno de los dos datos y una tercera opción para permitir el registro de nuevos usuarios. Una vez que un usuario registrado está identificado en la web, en el mismo menú vertical aparecerá un **submenú de Usuario** en el que se podrán modificar los datos de registro así como contactar con los administradores del sitio para mandar sugerencias, dar o pedir información acerca de una herramienta o solicitar tener acceso a la modificación o creación de un artículo. Otra opción de login se puede encontrar en la esquina superior derecha de la vista general del portal, donde aparecen además dos botones que permiten aumentar o disminuir el tamaño del texto. Bajo estos dos componentes de la página web se puede encontrar un buscador que permite realizar búsquedas dentro de todos los apartados del sitio.

Otro apartado que incluye el portal web y que se aprecia tanto en la *figura 89* como en la *figura 90* es el pie de página (9) (*footer*) en el que se incluyen:

- Dos banderas, la española y la británica, que permiten la traducción del sitio a los lenguajes nativos de España y Reino Unido respectivamente.
- Un botón que permite ir a la parte superior de la web siempre que se desee y que resulta especialmente útil cuando se está navegando por el análisis de una herramienta.
- Una serie de enlaces a las entidades relacionadas con el desarrollo, como son la Universidad de Valladolid y el grupo GIRO.
- Varios enlaces de contacto con los principales responsables del mantenimiento del portal.
- Los derechos de autor, así como los permisos que se otorgan acerca de la copia y distribución de los contenidos.

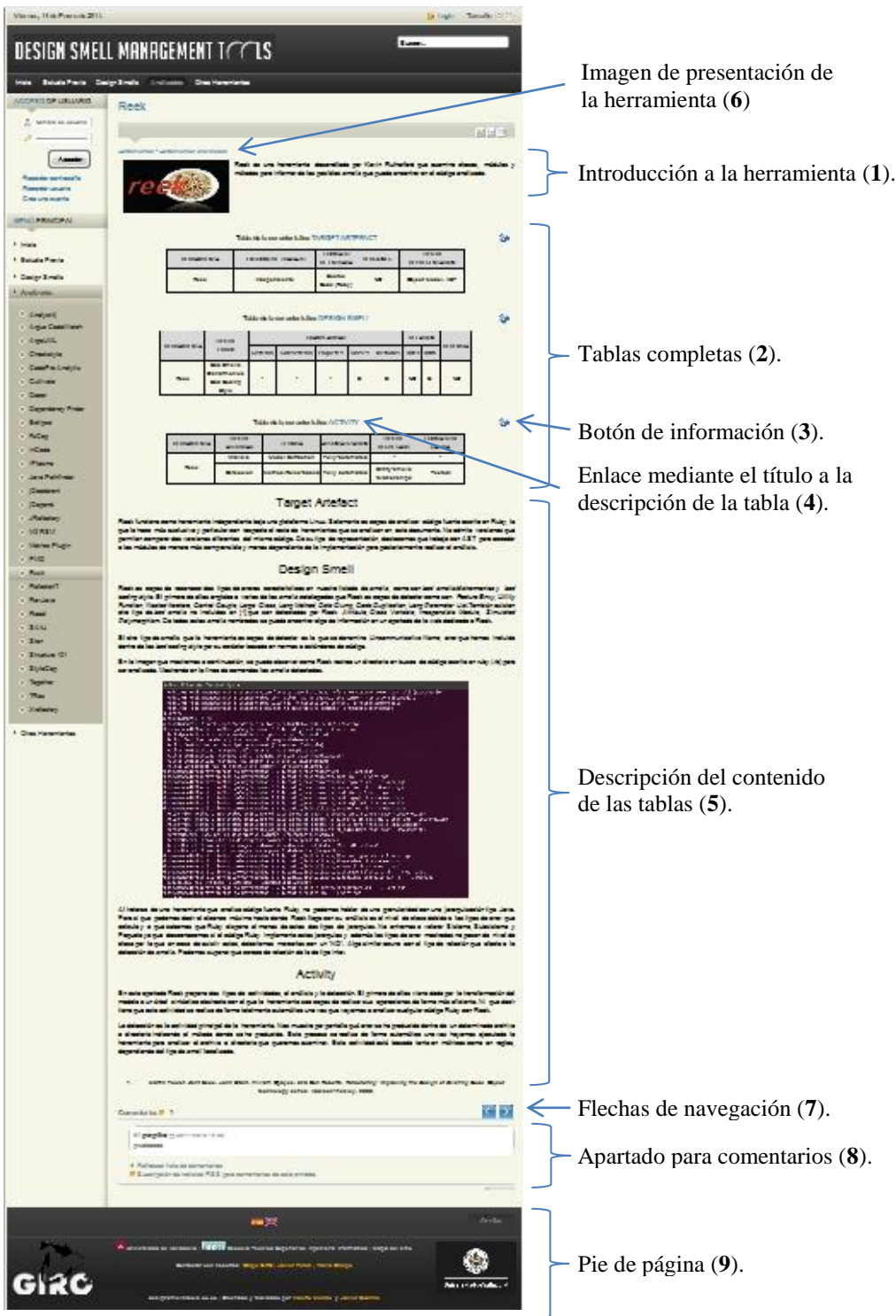


Figura 90. Design Smell Management Tools. Vista del análisis de una herramienta.

5.6. Mantenimiento Web

En esta sección se expondrá un breve tutorial sobre las partes que componen la administración de Joomla! del sitio web diseñado y sus posibles utilidades. Para ello se irán desglosando los apartados que se consideran más importantes dentro de ella y se darán instrucciones concisas sobre su utilización.

El primer aspecto fundamental es conocer como se accede a la administración del sitio, para ello se introducirá en el navegador la url *http://www.infor.uva.es/DesignSmells/administrator*. Una vez hecho esto aparecerá la ventana de login pidiéndose el nombre de usuario y contraseña.

Figura 91. Design Smell Management Tools. Login administración.

Para futuros administradores que realicen el mantenimiento del portal se ha creado un usuario con rango de Super Administrador (máximo rango que se puede poseer) con el que puedan acceder a dicha administración. El nombre de usuario y contraseña es *admin* para ambos campos.

Una vez que se hayan introducido los datos correctamente aparecerá la venta principal de la administración que como se podrá observar en la siguiente figura es una interfaz clara, sencilla e intuitiva.

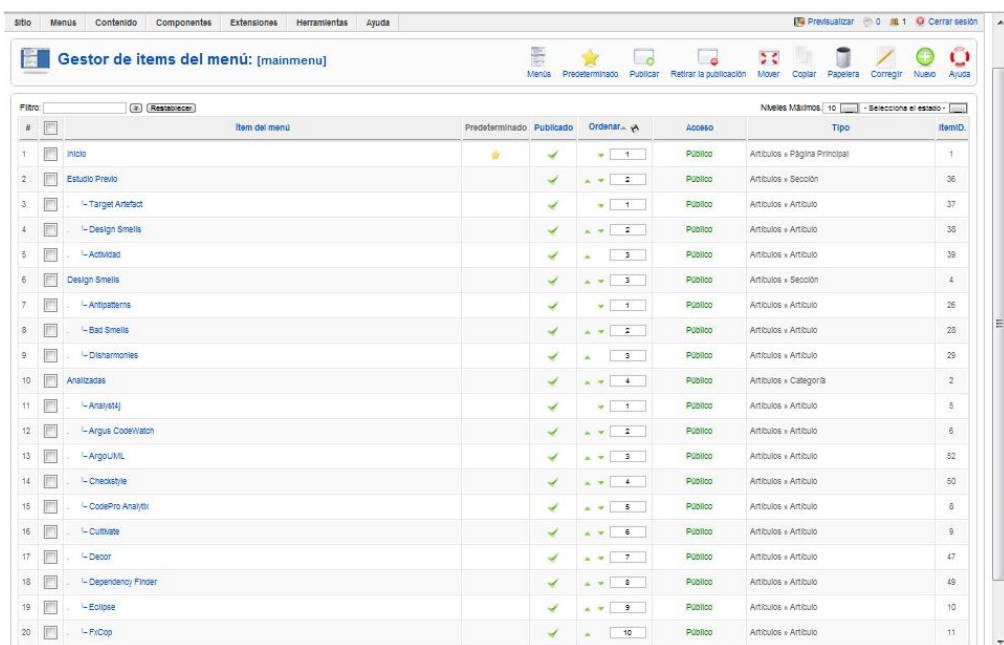
#	Nombre	Grupo	Ciente	Última vez Activo	Cerrar Sesión
1	mendo10	Super Administrador	administrador	0.0 horas	
2	xavidax	Super Administrador	administrador	0.0 horas	

Figura 92. Design Smell Management Tools. Vista principal administración.

En la parte superior de la ventana aparece un menú horizontal, dónde se encuentran todas las funcionalidades que Joomla! ofrece para su administración.

Si se pasa el ratón por **Sitio** se desplegará un submenú vertical con diferentes secciones de las que se destacan el gestor de usuarios y la configuración global. El gestor de usuarios tendrá la funcionalidad de poder otorgar un determinado rango (si es que un usuario registrado lo ha solicitado) así como poder modificar alguno de sus datos y poder eliminar un usuario o crearlo. En la configuración global se deberán incluir algunos parámetros necesarios para el correcto funcionamiento de la web tales como la configuración del servidor, de la base de datos, de correo, etc., así como determinadas funciones que dependerá del diseñador del sitio incluirlas o no, como son la posibilidad de que se registren usuarios, configuración de la caché, configuración de la sesión, etc.

La siguiente pestaña, **Menús**, desplegará tres opciones diferenciadas. La primera de ellas, gestor de menús, se utilizará básicamente para crear los diferentes menús que posteriormente estarán ocupados con las referencias al contenido. En el caso de la web desarrollada, si se accede a él, se podrá comprobar la existencia de dos únicos menús, el menú principal y el de usuario. También se encontrará en esta sección una papelerera de menús, dónde se ubicaran aquellos menús que no se vayan a utilizar pero que se quieran mantener por si en el algún momento se quieren recuperar. La última sección del submenú mostrará los menús que se encuentran activos en la web. En este caso serán el menú principal y el menú de usuario. Si se hace clic en el menú principal por ejemplo, se abrirá una ventana dónde se podrá comprobar la disposición de los ítems del menú según la división establecida por el usuario (figura 93).



#	Item del menú	Predeterminado	Publicado	Ordenar	Acceso	Tipo	ItemID
1	Inicio		✓	1	Público	Artículos » Página Principal	1
2	Estudio Previo		✓	2	Público	Artículos » Sección	36
3	↳ Target Antefact		✓	1	Público	Artículos » Artículo	37
4	↳ Design Smells		✓	2	Público	Artículos » Artículo	38
5	↳ Actividad		✓	3	Público	Artículos » Artículo	39
6	Design Smells		✓	3	Público	Artículos » Sección	4
7	↳ Antipatterns		✓	1	Público	Artículos » Artículo	26
8	↳ Bad Smells		✓	2	Público	Artículos » Artículo	28
9	↳ Disharmonies		✓	3	Público	Artículos » Artículo	29
10	Análizasis		✓	4	Público	Artículos » Categoría	2
11	↳ Analistid		✓	1	Público	Artículos » Artículo	5
12	↳ Argus CodeWatch		✓	2	Público	Artículos » Artículo	6
13	↳ ArguUML		✓	3	Público	Artículos » Artículo	52
14	↳ Checkstyle		✓	4	Público	Artículos » Artículo	50
15	↳ CodePro Analytix		✓	5	Público	Artículos » Artículo	8
16	↳ Cultivate		✓	6	Público	Artículos » Artículo	9
17	↳ Decor		✓	7	Público	Artículos » Artículo	47
18	↳ Dependency Finder		✓	8	Público	Artículos » Artículo	49
19	↳ Eclipse		✓	9	Público	Artículos » Artículo	10
20	↳ FixCop		✓	10	Público	Artículos » Artículo	11

Figura 93. Design Smell Management Tools. Menu Principal.

Si se desea añadir un nuevo ítem a este menú se deberá pulsar en la esquina superior derecha sobre el icono “Nuevo” (circulo verde). También existen otros iconos en la zona superior como: publicar, retirar publicación, mover, copiar y papeleria, entre otros, que ayudarán al desarrollador a

operar con facilidad sobre las acciones en el menú. Volviendo al tema de añadir un nuevo ítem, una vez pulsado en nuevo aparecerá una pantalla en la que se pedirá seleccionar el tipo de ítem que se desea incluir en el menú. Como se puede observar en la figura siguiente las opciones son variadas. Si se desea añadir un nuevo artículo se pulsaría sobre él desplegándose una nueva lista sobre las diferentes opciones de presentación, seleccionando después Presentación del artículo.

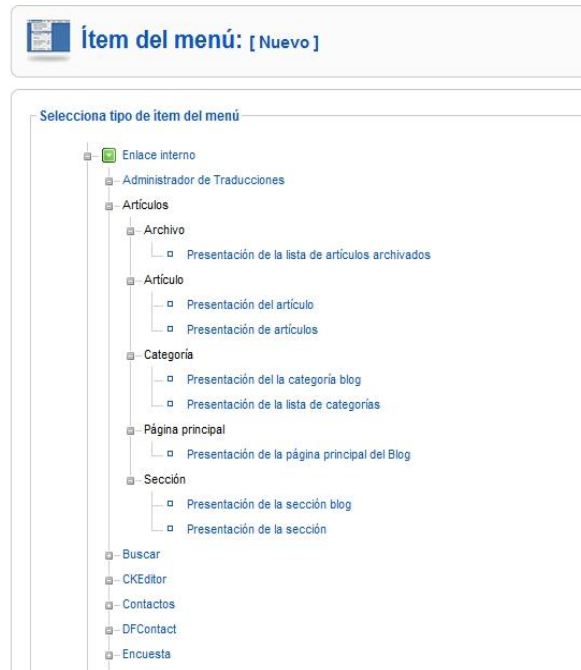


Figura 94. Design Smell Management Tools. Items del Menú.

Antes de crear el ítem de menú para un artículo concreto es recomendable haber creado dicho artículo antes, ya que en la ventana de presentación de artículo se pedirá adjuntar el artículo al que se haga referencia para ese ítem. En la pestaña Contenido de la administración se verá como se crea un artículo.

Contenido posee un submenú que consta en primer lugar de un gestor de artículos. Si se hace clic sobre el gestor, el usuario se encontrará con una especie de “almacen” dónde se encontrarán todos los artículos que se hayan creado hasta ese momento, pudiendo entrar en ellos sin más que hacer clic sobre cada uno para poder editarlos. De vuelta a la ventana del gestor de artículos, en la parte superior se encontraran diferentes iconos para operar con los artículos que se encuentren en el gestor, de tal modo que se puedan archivar, desarchivar, mover, borrar, etc. Entre esos iconos estará ubicado el de crear un artículo nuevo (Nuevo). Si se pulsa en este icono se desplegará en la ventana un interfaz con un editor de texto integrado para la creación del artículo (existen una gran variedad de editores de textos que pueden ser instalados).

The screenshot shows the 'Artículo: [Nuevo]' (Article: [New]) form. At the top, there's a title bar with the text 'Artículo: [Nuevo]' and a toolbar with icons for 'Previsualizar' (Preview), 'Guardar' (Save), 'Aplicar' (Apply), 'Cancelar' (Cancel), and 'Ayuda' (Help). The form is divided into several sections:

- Form Fields:**
 - Título:** A text input field containing 'Together'.
 - Alias:** A text input field.
 - Sección:** A dropdown menu with 'Herramientas' selected. A tooltip is visible showing options: 'Herramientas', 'Selecciona sección - Sin clasificar', 'Herramientas', 'Design Smells', and 'Estudio Previo'.
 - Publicado:** Radio buttons for 'No' (selected) and 'Sí'.
 - Página principal:** Radio buttons for 'No' (selected) and 'Sí'.
 - Categoría:** A dropdown menu with 'Herramientas analizadas' selected.
- Rich Text Editor:** A large text area with a toolbar for formatting (bold, italic, underline, link, unlink, list, etc.). The toolbar also includes 'Fuente' (Font) and 'Tamaño' (Size) options.
- Metadata and Parameters (Right Sidebar):**
 - Estado:** 'Publicado'.
 - Impresiones:** '0 Vezes'.
 - Revisado:** 'Miércoles, 19 Enero 2011 13:36'.
 - Creado:** 'No modificado'.
 - Modificado:** 'No modificado'.
 - Parámetros - Artículo:**
 - Autor:** 'Javier Sobrino'.
 - Alias del autor:** A text input field.
 - Nivel de acceso:** 'Público'.
 - Fecha de creación:** '2011-01-19 13:36:44'.
 - Iniciar publicación:** '2011-01-19 13:36:44'.
 - Publicación finalizada:** 'Nunca'.
 - Parámetros - Avanzados:** A section for advanced parameters.
 - Información de metadatos:** A section for meta-information.

Figura 95. Design Smell Management Tools. Artículo nuevo.

Es fundamental que se indique la sección y la categoría que se quiera dar al artículo que va a crearse para mantener una correcta organización (es recomendable que se creen las diferentes secciones y categorías antes de crear los artículos). Una vez finalizado el artículo y haber marcado las opciones que el usuario desee, se pulsará el icono de guardar y automáticamente pasará a formar parte de uno de los artículos contenidos en el gestor y podrá usarse para ser adjuntado como ítem de un menú como se comentó anteriormente.

La pestaña contenido también posee una papelerera de artículos dónde se encontrarán todos aquellos que hayan sido enviados allí por el desarrollador. Los artículos que se encuentren en ella podrán ser restaurados o borrados de forma definitiva.

El gestor de secciones permitirá crear las secciones que el desarrollador crea oportunas. Cada sección contendrá diferentes categorías que se generarán en el gestor de categorías.

La opción del gestor de la página de inicio se utilizará por si se quiere incluir algún contenido en el index del portal web.

La siguiente pestaña, **Componentes**, incluirá todos aquellos módulos en formato componente, que vienen instalados junto con Joomla! o que han sido instalados por el usuario. Estos componentes permiten al desarrollador variar la configuración de los diferentes módulos instalados. En el caso de la web realizada cabe destacar la utilización del administrador de traducciones por si existe algún elemento en la web que deba traducirse manualmente; el componente *DFcontact* para transcribir y modificar los datos que se quiere que aparezcan en el apartado de contacto de la web para usuarios registrados; el componente *jcomments* para manipular las opciones de los comentarios que pueden hacer los usuarios registrados en determinados artículos de la web; o el componente *joomlawatch* para poder visualizar estadísticas en tiempo real del volumen de visitas.

Extensiones será la pestaña que contenga una de las funcionalidades más utilizadas en Joomla! que no es otra que la de instalar (o desinstalar) módulos, componentes o plugins. La forma más fácil de instalar es haber descargado previamente el módulo o plugin y haberlo guardado en el disco duro, después hacer clic sobre "Instalar/Desinstalar" donde se podrá observar en la sección "subir paquete" un botón con la función de seleccionar un archivo desde el equipo. Una vez seleccionado, se pulsará sobre "Subir archivo e instalar" y el componente será instalado. Cabe la posibilidad de hacer la instalación desde la url donde esté ubicado el paquete a instalar.



Figura 96. Design Smell Management Tools. Instalar.

Si se desea desinstalar algún componente, módulo, plugin, idioma o plantilla; simplemente habrá que hacer clic dentro de la ventana mostrada anteriormente en la *figura 96* en alguno de los identificadores nombrados anteriormente. Una vez dentro de alguno de ellos se seleccionará el elemento deseado y se pulsará a desinstalar.

De los otros apartados que contiene el submenú de extensiones se destaca el Gestor de Módulos, mediante el cual se podrá crear nuevos módulos para el sitio web o administración a partir de los componentes que se hayan instalados o de los propios que trae consigo Joomla!. Estos nuevos módulos podrán ubicarse en la web a partir de las posiciones que la plantilla con la que se este trabajando lo permita. El resto de gestores (plugins, plantilla e idioma) únicamente servirán para publicar o despublicar aquellos elementos que el usuario decida, como ocurre en el gestor de plugins, o para marcar la plantilla o el idioma predeterminados como ocurre en el caso de los dos gestores restantes (Gestor de plantillas y Gestor de idiomas).

De la pestaña **herramientas** destaca la función de “desbloqueo global” que se utilizará, por ejemplo, en el caso de haber salido sin guardar o cerrar un artículo que se haya estado editando, lo que provoca que éste quede bloqueado. Ésta función desbloqueará todos los posibles casos existentes. Otra opción existente en la pestaña herramientas es la de limpiar caché, que permitirá liberar espacio en caché cuando la web tenga mucha afluencia de usuarios.

REFERENCIAS

1. Javier Pérez, Nahuel Moha, Tom Mens, and Carlos López. *A classification framework and survey for design smell management*. Enviado a Journal of software maintenance and evolution: Research and practice. Pendiente de revisión en la fecha de creación de esta memoria.
2. Brown, William J., Raphael C. Malveau, Hays W. "Skip" McCormick, Scott W. Thomas, Theresa Hudson. *Anti-Patterns in Project Management*. John Wiley & Sons, Ltd. (2000).
3. Anti patterns Catalog. <http://c2.com/cgi/wiki?AntiPatternsCatalog>. Último acceso: 01/08/2010.
4. Anti-Pattern from Wikipedia. <http://en.wikipedia.org/wiki/Anti-pattern>. Ultimo acceso: 01/08/2010.
5. Raúl Marticorena, Carlos López, and Yania Crespo. *Extending a Taxonomy of Bad Code Smells with metrics*.
6. Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Object technology series. Addison-Wesley, 2000.
7. 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.
8. Gustavo Damián Campo. *Patrones de Diseño, refactorización y Antipatrones. Ventajas y desventajas de su utilización en el software orientado a objetos*. Cuadernos de la facultad nº4, 2009.
9. Cultivate. <http://sewiki.iai.uni-bonn.de/research/cultivate/start>. Último acceso: 05/08/2010.
10. CodeClinic. Herramienta facilitada por su autor, Adrian Trifu. Disponible en el CD que acompaña al proyecto.
11. SemmleCode. <http://semmle.com>. Último acceso: 05/08/2010.
12. CodeCrawler. <http://www.inf.usi.ch/faculty/lanza/codecrawler.html>. Último acceso: 05/08/2010.
13. X-ray. <http://xray.inf.usi.ch/xray.php>. Último acceso: 05/08/2010.
14. Optimal Advisor. <http://www.compuware.com>. Último acceso: 27/08/2010.
15. Goose. <http://esche.fzi.de/PROSTextern/software/goose>. Último acceso: 06/08/2010.
16. Sissy. <http://fast.fzi.de/index.php/sissy>. Último acceso: 06/08/2010.
17. ArgoUML. <http://argouml.tigirs.org>. Último acceso: 06/08/2010.
18. Together. <http://www.borland.com/us/products/together/index.html>. Último acceso: 06/08/2010.
19. Checkstyle. <http://checkstyle.sourceforge.net/index.html>. Último acceso: 07/08/2010.
20. Reek. <http://wiki.github.com/kevinrutherford/reek/>. Último acceso: 10/08/2010.
21. Roodi. <http://github.com/martinjandrews/roodi>. Último acceso: 10/08/2010.
22. FxCop. <http://msdn.microsoft.com/en-us/library/bb429476%28VS.80%29.aspx>. Último acceso: 10/08/2010.
23. StyleCop. <http://stylecop.codeplex.com>. Último acceso: 10/08/2010.
24. RevJAVA. <http://www.serc.nl/people/florijn/work/designchecking/RevJava.htm>. Último acceso: 11/08/2010.

25. Eclipse. <http://www.eclipse.org>. Último acceso: 11/08/2010.
26. NetBeans. <http://netbeans.org>. Último acceso: 11/08/2010.
27. Visual Studio. <http://msdn.microsoft.com/es-es/vstudio/default.aspx>. Último acceso: 11/08/2010.
28. jEdit. <http://www.jedit.org>. Último acceso: 11/08/2010.
29. PMD. <http://pmd.sourceforge.net>. Último acceso: 11/08/2010.
30. jDeodorant. <http://www.jdeodorant.com>. Último acceso: 12/08/2010.
31. ASG Wikipedia. http://en.wikipedia.org/wiki/Abstract_semantic_graph. Último acceso: 12/08/2010.
32. Nahuel Moha. *DECOR: Détection et correction des défauts des systèmes orientés objet*. PhD thesis, Université des Sciences et Technologies de Lille; Université de Montréal, August 2008.
33. RefactorIt. <http://refactorit.sourceforge.net>. Último acceso: 13/08/2010.
34. Eclipse Metrics Plugin. <http://eclipse-metrics.sourceforge.net>. Último acceso: 13/08/2010.
35. jRefactory. <http://jrefactory.sourceforge.net>. Último acceso: 14/08/2010.
36. Analyst4j. <http://www.codeswat.com/cswat/index.php>. Último acceso: 16/08/2010.
37. LOOSE Research Group. iPlasma. <http://loose.upt.ro/iplasma>. Último acceso: 16/08/2010.
38. Xref-tech. Xrefactory. <http://www.xref-tech.com/xrefactory/main.html>. Último acceso: 16/08/2010.
39. Ptidej. <http://ptidej.dyndns.org/material/inanutshell>. Último acceso: 16/08/2010.
40. Stan. <http://stan4j.com>. Último acceso: 16/08/2010.
41. William C. Wake. *Refactoring Workbook*. Addison Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
42. 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.
43. TTCN-3. TRex. <http://www.trex.informatik.uni-goettingen.de/trac>. Último acceso: 16/08/2010.
44. Zhenchang Xing and Eleni Stroulia. *Data-mining in support of detecting class co-evolution*. In Frank Maurer and Günter Ruhe, editors, *Proceedings of the sixteenth International Conference on Software Engineering & Knowledge Engineering*, pages 123-128, June 2004.
45. Salah Bouktif, Giuliano Antoniol, Ettore Merlo, and Markus Neteler. *A Novel Approach to Optimize Clone Refactoring Activity*. In GECCO'06: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1885-1892, New York, NY, USA, 2006.
46. Thierry Bodhuin, Gerardo Canfora, and Luigi Troiano. *SORMASA: A tool for Suggesting Model Refactoring Actions by Metrics-led Genetic Algorithm*. In 1st Workshop on Refactoring Tools (WRT 2007), pages 23-24.
47. Holger Baer, and Oliver Ciupke. *Exploiting Design Heuristics for Automatic Problem Detection*. In *ECOOP Workshop on Experiences in Object-Oriented Re-Engineering*, pages 73-74, London, UK, 1998.

48. Douglas Kirk, Marc Roper, Murray Wood. *A Heuristic-Based Approach to Code-Smell Detection*. In 1st Workshop on Refactoring Tools (WRT 2007), pages 55-56.
49. Argus Codewatch. <http://arguscodewatch.sourceforge.net>. Último acceso: 20/08/2010
50. CodePro Analytix. <http://www.instantiations.com>. Último acceso: 20/08/2010. Herramienta no disponible.
51. Al Vermeulen, Scott W. Ambler, Greg Bumgardner, Eldon Metz, Trevor Misfeldt, Jim Shur, Patrick Thompson. *The Elements of Java™ Style*, Cambridge University Press, 2000.
52. Swi-prolog. <http://www.swi-prolog.org>. Último acceso: 20/08/2010.
53. jTransformer. <http://sewiki.iai.uni-bonn.de/research/jtransformer/start>. Último acceso: 20/08/2010.
54. Mylyn. <http://www.eclipse.org/mylyn>. Último acceso: 20/08/2010.
55. Ramesh Seela, Ryan Miller, Derek Chang, Ali Shojaeddini, Ankit Sengar. *FxCop tool evaluation. Project Report*. Analysis of software artifacts. Carnegie Mellon University. 2008.
56. inCode. <http://www.intooitus.com/inCode.html>. Último acceso: 23/08/2010.
57. Nikolaos Tsantalis, Theodoros Chaikalis, Alexander Chatzigeorgiou. *JDeodorant: Identification and Removal of Type-Checking Bad Smells*. Department of Applied Informatics, University of Macedonia. 2008.
58. Marios Fokaefs, Nikolaos Tsantalis, Alexander Chatzigeorgiou. *JDeodorant: Identification and Removal of Feature Envy Bad Smells*. Department of Applied Informatics, University of Macedonia. 2007.
59. Nikolaos Tsantalis, Alexander Chatzigeorgiou. *Identification of refactoring opportunities introducing polymorphism*. The Journal of Systems and Software 83 (2010), pages 391–404.
60. JRefactory. <http://jrefactory.sourceforge.net>. Último acceso: 30/08/2010.
61. M2 Resource Standard Metrics. <http://msquaredtechnologies.com/m2rsm/index.htm>. Último acceso: 31/08/2010.
62. Allen Hsu, Somakala Jagannathan, Sajjad Mustehsan, Session Mwamufiya, Marc Novakous. *Analysis tool evaluation: PMD. Project Report*. Carnegie Mellon University. April 24th, 2007.
63. Structural Analysis for Java (SA4J). <http://www.alphaworks.ibm.com/tech/sa4j>. Último acceso: 18/10/2010
64. STAN. <http://stan4j.com>. Último acceso: 20/09/2010.
65. Structure101. <http://www.headwaysoftware.com/products/structure101/index.php>. Último acceso: 20/09/2010.
66. PRQA (Structure 101 para C/C++). <http://www.programmingresearch.com/structure101.html>.
67. Stan. White Paper Fall 2009.
68. Headway Software. *XS – A Measure of Structural Over-Complexity*. White Paper. Enero 2006.
69. Together. <http://www.borland.com/us/products/together/index.html>. Último acceso: 22/09/2010.
70. Roodi. <http://blog.martyandrews.net/2009/05/roodi-140-released.html>. Último acceso: 06/10/2010.
71. Marty Andrews. *Code Quality Analysis*. Cogentconsulting.co.au.

72. Stefano Maggioni. *Design Pattern Detection and Software Architecture Reconstruction: an Integrated Approach based on Software Micro-structures*. Università degli Studi di Milano-Bicocca. Dipartimento di Informatica, Sistemistica e Comunicazione. Dottorato di Ricerca in Informatica.
73. Benjamin Zeiss, Helmut Neukirchen, Jens Grabowski, Dominic Evans, Paul Baker. *TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications*. Technische Berichte des Instituts für Informatik an der Georg-August-Universität Göttingen.
74. Benjamin Zeiss, Helmut Neukirchen, Jens Grabowski, Dominic Evans, Paul Baker *TRex. An Open-Source Tool for Quality Assurance of TTCN-3 Test Suites*.
75. Danny Dig, Michael Cebulla. *1st Workshop on Refactoring tools (WRT'07)*. July 31, 2007, TU Berlin, Germany.
76. Martin Bisanz. *Pattern-Based Smell Detection in TTCN-3 Test Suites*. Am Institute für Informatik Gruppe Softwaretechnik für Verteilte Systeme. Bachelor- und Masterarbeiten des Zentrums für Informatik an der Georg-August-Universität Göttingen. 19 Dezember 2006.
77. Jens M. Nödler. *An XML-based Approach for Software Analysis Applied to Detect Bad Smells in TTCN-3 Test Suites*. Software Engineering for Distributed Systems Group. Institute for Computer Science University of Göttingen, Germany. 26/07/2007.
78. Adrian Trifu, Mircea Trifu. *SISSy: Catalog of Detected Problem Patterns*. FZI Forschungszentrum Informatik, Karlsruhe, Germany.
79. Mircea Trifu, Adrian Trifu. *SISSy – a Tool for Structural Investigation of Object-Oriented Software Systems*. FZI Forschungszentrum Informatik, Software Engineering Group, Karlsruhe, Germany.
80. jCosmo. <http://old-www.cwi.nl/projects/renovate/javaQA/>. Último acceso: 18/10/2010.
81. Emacs. <http://www.gnu.org/software/emacs/>. Último acceso: 18/10/2010.
82. xEmacs. <http://www.xemacs.org>. Último acceso: 18/10/2010.
83. M.J. Escalona, N. Koch. *Ingeniería de Requisitos en Aplicaciones para la Web: Un estudio comparativo*. Trabajo financiado por Deutscher Akademischer Austauschdienst (DAAD) y Ministerio de Educación y Ciencia y los fondos FEDER.
84. REM. http://www.lsi.us.es/descargas/descarga_programas.php?id=3.
85. Adrian Trifu. *Towards Automated Restructuring of Object Oriented Systems*. Dissertation, Universität Karlsruhe (TH), Fakultät für Informatik, 2008.
86. Hammurapi. <http://www.hammurapi.com/dokuwiki/doku.php>. Último acceso: 18/10/2010.
87. Pavel Vlasov. *Hammurapi, code review platform*. <http://www.hammurapi.org>. January 2007.
88. Eva van Emden, Leon Moonen. *Java Quality Assurance by Detecting Code Smells*. CWI, The Netherlands, <http://www.cwi.nl>.
89. William C. Wake. *Refactoring Workbook*. Addison-Wesley, 2003.
90. Proyecto09. Javier Sobrino y Alberto Mendo, práctica de la asignatura Ingeniería del Software II. E.T.S.I. Informática. Universidad de Valladolid. Curso 2008/2009.
91. Alejandro Ramirez *et al.* ArgoUML: Manual de usuario.
92. Wamp. <http://www.wampserver.com/>. Último acceso: 13/01/2011.

93. Wiki Eclipse. Eclipse AST. http://wiki.eclipse.org/FAQ_What_is_an_AST%3F.
94. Ondrej Míkle. *SiSSy/Recoder/Java2PCM*. Distributed systems research group. <http://dsrg.mff.cuni.cz>. Charles University in Prague. Faculty of Mathematics and Physics. DSRG Seminar, 27 Nov 2007.
95. CMS Wikipedia. http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenidos. Ultimo acceso: 13/01/2011.
96. Joomla! Wikipedia. <http://es.wikipedia.org/wiki/Joomla!>. Ultimo acceso: 13/01/2011.
97. Lucene. <http://lucene.apache.org/java/docs/index.html>.
98. jFreechart. <http://www.jfree.org/jfreechart>.
99. jHotdraw. <http://www.jhotdraw.org/>.
100. QuickGraph. <http://quickgraph.codeplex.com/>.
101. Ganttproject. <http://www.ganttproject.biz/>.
102. CrocoPat. <http://www.sosy-lab.org/~dbeyer/CrocoPat/>.

APÉNDICE I

CONTENIDO DEL DISCO

El contenido del disco que acompaña a esta memoria tiene la siguiente estructura:

- Sitio Web: Directorio que contiene los archivos relacionados con el portal web.
 - Web: Directorio que contiene las carpetas que componen el sitio web que se ha desarrollado.
 - localhost.sql: Archivo en formato sql que contiene la base de datos del sitio web.
- Herramientas: Directorio en el que se incluyen las herramientas de libre distribución que se han analizado en este trabajo. Existen varios formatos:
 - Ejecutables: Permiten la instalación directa de la herramienta.
 - Directorios: Contienen la herramienta ya instalada dentro del IDE Eclipse.
 - Archivos Comprimidos: Contienen el código fuente de la herramienta.
- Source Code: Directorio que contiene los diferentes códigos fuente utilizados para efectuar los análisis.
- Memoria: Directorio en el que se encuentra la memoria del proyecto en formato PDF.

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Figura 1. Planificación inicial del proyecto.	15
Figura 2. Seguimiento general del proyecto.	18
Figura 3. Seguimiento específico de la parte de desarrollo del sitio web.	19
Figura 4. Raíz del diagrama de características.	23
Figura 5. Característica Entorno de Trabajo y sus sub-características.	24
Figura 6. Característica Target Artefact. Subcaracterística de Design Smell Management.	25
Figura 7. Característica Design Smell. Subcaracterística de Design Smell Management.	27
Figura 8. Característica Actividad. Subcaracterística de Design Smell Management.	30
Figura 9. Característica Técnica. Subcaracterística de Actividad.	31
Figura 10. Característica Resultado. Subcaracterística de Actividad.	33
Figura 11. Analyst4j. Comparación manual entre dos versiones.	43
Figura 12. Analyst4j. Ejemplo de comparación de cifras.	44
Figura 13. Analyst4j. Umbral de métricas usado para identificar The Blob.	45
Figura 14. Analyst4j. Umbral de métricas usado para identificar Spaghetti Code.	45
Figura 15. Analyst4j. Umbral de métricas usado para identificar Swiss Knife Classes.	45
Figura 16. Analyst4j. Actividad Visualización en la que se muestra el resultado de un análisis.	46
Figura 17. ArgoUML. Elementos problemáticos detectados tras un análisis con código java.	50
Figura 18. ArgoUML. Sugerencias de corrección mostradas para un determinado problema.	50
Figura 19. Argus CodeWatch. Detección de errores.	53
Figura 20. Argus CodeWatch. Preferencias Argus CodeWatch.	54
Figura 21. Argus CodeWatch. Sugerencias de corrección.	55
Figura 22. CodePro Analytix. Creación de un nuevo conjunto de reglas a partir de libros.	60
Figura 23. Cultivate. Conversión a proyecto Cultivate.	64
Figura 24. Cultivate. Vistas que incluye la herramienta.	65
Figura 25. Cultivate. Selección del modo de marcar el error en Cultivate.	66
Figura 26. Eclipse. Bad Coding Style en Eclipse.	72
Figura 27. Eclipse. Cuadro de corrección en Eclipse.	73
Figura 28. FxCop. Análisis de programa usando la herramienta FxCop.	78
Figura 29. FxCop. Sugerencia de corrección mostrada al hacer doble clic en un error.	79
Figura 30. inCode. Preferencias de Análisis en las que se ve la lista de Disharmonies que soporta.	83
Figura 31. inCode. Niveles a los que la herramienta trabaja para realizar gráficos.	84
Figura 32. inCode. Uso de la herencia mientras se carga inCode Overview.	84
Figura 33. inCode. Vista inCode Overview en la que se ve las métricas y los errores.	85
Figura 34. inCode. Listado de errores que aparecen en el proyecto ganttproject2.	85
Figura 35. inCode. Detección de Disharmonies. Sugerencias de corrección.	86
Figura 36. jDeodorant. Detección de type checking.	90
Figura 37. jDeodorant. Corrección mediante refactorización.	91
Figura 38. jDeodorant. Análisis de impacto.	91
Figura 39. JRefactory. Coding Standards. Posibles errores en código y sugerencias de refactorización.	96
Figura 40. JRefactory. Opciones de refactorización.	97
Figura 41. RSM. Interfaz de M2 Resource Standard Metrics.	102
Figura 42. RSM. Informe originado tras ejecutar la opción de detección.	103
Figura 43. PMD. Vista general en la que se muestran las violaciones de código detectadas.	108
Figura 44. PMD. Opción de Especificación donde se ven las diversas posibilidades.	109
Figura 45. Reek. Muestra de smells detectados en un directorio con archivos escritos en Ruby.	114
Figura 46. RevJava. Ventana principal con el código de proyecto09 cargado.	119

Índice de figuras

Figura 47. RevJava. Ventana Critics.	120
Figura 48. Roodi. Ejecución de Roodi con varios códigos diferentes.	125
Figura 49. STAN. Metrics Warnings en Stan.	129
Figura 50. STAN. Visualización del tangle.	130
Figura 51. Stan. Proceso de especificación de las métricas.	131
Figura 52. Stan. Muestra de metrics warnings detectados.	131
Figura 53. SA4j. Muestra de dependencias entre las clases.	135
Figura 54. SA4j. Muestra de antipatrones.	136
Figura 55. SA4j. Muestra del antipatrón local hub.	136
Figura 56. SA4j. Visualización skeleton.	137
Figura 57. SA4j. Ejecución de what if.	138
Figura 58. Structure101. Funcionamiento de la aplicación web con un repositorio de versiones.	142
Figura 59. Structure101. Gráfico de dependencias en el que se señalan los ciclos.	143
Figura 60. Structure101. Vista Overview en la que se calculan y muestran métricas.	144
Figura 61. Structure101. Muestra del Impact Analysis que la herramienta incluye.	145
Figura 62. StyleCop. Ejecución de la herramienta sobre código C#.	149
Figura 63. StyleCop. Resultados obtenidos tras efectuar un análisis.	150
Figura 64. StyleCop. Ventana de selección de reglas para realizar un análisis.	151
Figura 65. Together. Preferencias para Audits.	156
Figura 66. Together. Muestra de bad coding style.	157
Figura 67. Together. Muestra de métricas.	157
Figura 68. Together. Ventana con las opciones de especificación.	158
Figura 69. Together. Actividad de Corrección.	159
Figura 70. TRex. Cadena de acciones de la herramienta. Imagen extraída de [73].	163
Figura 71. TRex. Resultado del análisis realizado a código TTCN-3.	164
Figura 72. XRefactory. Dead Code en XRefactory integrado en Emacs.	170
Figura 73. CodeClinic. Vista lógica de la arquitectura del sistema [85].	173
Figura 74. CodeClinic. Error generado por el Eclipse que actúa como lanzador de la aplicación.	173
Figura 75. CodeClinic. Error obtenido en la ejecución de la herramienta.	174
Figura 76. jCosmo. Muestra de los problemas encontrados en la instalación.	176
Figura 77. Sissy. Pasos que sigue para analizar código.	177
Figura 78. Sissy. Listado de problemas de código que busca la herramienta.	177
Figura 79. XRay. Visualización de XRay en Eclipse.	179
Figura 80. Proceso de ingeniería de requisitos.	195
Figura 81. Interacción Web y usuarios.	198
Figura 82. Instalación de Joomla!. Lanzar phpMyAdmin en Wampserver.	208
Figura 83. Instalación de Joomla!. Vista general de phpMyAdmin.	208
Figura 84. Instalación de Joomla!. Ventana de selección de idioma para la instalación.	209
Figura 85. Instalación de Joomla!. Ventana de comprobación previa.	210
Figura 86. Instalación de Joomla!. Ventana de configuración de la base de datos.	210
Figura 87. Instalación de Joomla!. Ventana de configuración principal del sitio web.	211
Figura 88. Instalación de Joomla!. Último paso.	211
Figura 89. Design Smell Management Tools. Página de inicio del sitio web.	212
Figura 90. Design Smell Management Tools. Vista del análisis de una herramienta.	214
Figura 91. Design Smell Management Tools. Login administración.	215
Figura 92. Design Smell Management Tools. Vista principal administración.	215
Figura 93. Design Smell Management Tools. Menu Principal.	216
Figura 94. Design Smell Management Tools. Items del Menú.	217
Figura 95. Design Smell Management Tools. Artículo nuevo.	218
Figura 96. Design Smell Management Tools. Instalar.	219

ÍNDICE DE TABLAS

ÍNDICE DE TABLAS

Tabla 1. Catálogo de Antipatterns. De Abstraction Inversion a Junkyard Coding.	9
Tabla 2. Catálogo de Antipatterns. De Kill Two Birds With One Stone a Zero Means Nulls.	10
Tabla 3. Catálogo de Bad Smells utilizado. Del grupo Bloater al grupo O-O Abusers.	10
Tabla 4. Catálogo de Bad Smells utilizado. Del grupo Change Prevents a Not Defined.	11
Tabla 5. Catálogo de Dishamonies utilizado. Identity Dishamonies.	11
Tabla 6. Catálogo de Dishamonies utilizado. Collaboration Dishamonies.	12
Tabla 7. Listado Inicial. De ArgoUML a Structure101.	13
Tabla 8. Listado Inicial. De Tahvildari and Kontogiannis a XRefactory.	14
Tabla 9. Listado de herramientas tras la fase de búsqueda.	38
Tabla 10. Analyst4j. Tabla de la característica Target Artefact.	43
Tabla 11. Analyst4j. Tabla de la característica Design Smell.	46
Tabla 12. Analyst4j. Tabla de la característica Activity.	46
Tabla 13. ArgoUML. Tabla de la característica Target Artefact.	49
Tabla 14. ArgoUML. Tabla de la característica Design Smells.	49
Tabla 15. ArgoUML. Tabla de la característica Activity.	50
Tabla 16. Argus CodeWatch. Tabla de la característica Target Artefact.	53
Tabla 17. Argus CodeWatch. Tabla de la característica Design Smells.	54
Tabla 18. Argus CodeWatch. Tabla de la característica Activity.	55
Tabla 19. CodePro Analytix. Tabla de la característica Target Artefact.	59
Tabla 20. CodePro Analytix. Tabla de la característica Design Smells.	59
Tabla 21. CodePro Analytix. Tabla de la característica Activity.	60
Tabla 22. Cultivate. Tabla de la característica Target Artefact.	64
Tabla 23. Cultivate. Tabla de la característica Design Smells.	66
Tabla 24. Cultivate. Tabla de la característica Activity.	67
Tabla 25. Eclipse. Tabla de la característica Target Artefact.	71
Tabla 26. Eclipse. Tabla de la característica Design Smells.	72
Tabla 27. Eclipse. Tabla de la característica Activity.	73
Tabla 28. FxCop. Tabla de la característica Target Artefact.	77
Tabla 29. FxCop. Tabla de la característica Design Smells.	78
Tabla 30. FxCop. Tabla de la característica Activity.	79
Tabla 31. inCode. Tabla de la característica Target Artefact.	83
Tabla 32. inCode. Tabla de la característica Design Smells.	84
Tabla 33. inCode. Tabla de la característica Activity.	86
Tabla 34. jDeodorant. Tabla de la característica Target Artefact.	89
Tabla 35. jDeodorant. Tabla de la característica Design Smells.	89
Tabla 36. jDeodorant. Tabla de la característica Activity.	92
Tabla 37. JRefactory. Tabla de la característica Target Artefact.	95
Tabla 38. JRefactory. Tabla de la característica Design Smells.	96
Tabla 39. JRefactory. Tabla de la característica Activity.	97
Tabla 40. M2 Resource Standard Metrics. Tabla de la característica Target Artefact.	101
Tabla 41. M2 Resource Standard Metrics. Tabla de la característica Design Smells.	103
Tabla 42. M2 Resource Standard Metrics. Tabla de la característica Activity.	103
Tabla 43. PMD. Tabla de la característica Target Artefact.	107
Tabla 44. PMD. Tabla de la característica Design Smells.	107
Tabla 45. PMD. Tabla de la característica Activity.	109
Tabla 46. Reek. Tabla de la característica Target Artefact.	113
Tabla 47. Reek. Tabla de la característica Design Smells.	115
Tabla 48. Reek. Tabla de la característica Activity.	115

Tabla 49. RevJava. Tabla de la característica Target Artefact.	119
Tabla 50. RevJava. Tabla de la característica Design Smells.	121
Tabla 51. RevJava. Tabla de la característica Activity.	121
Tabla 52. Roodi. Tabla de la característica Target Artefact.	125
Tabla 53. Roodi. Tabla de la característica Design Smells.	125
Tabla 54. Roodi. Tabla de la característica Activity.	126
Tabla 55. STAN. Tabla de la característica Target Artefact.	129
Tabla 56. STAN. Tabla de la característica Design Smells.	130
Tabla 57. STAN. Tabla de la característica Activity.	132
Tabla 58. SA4J. Tabla de la característica Target Artefact.	135
Tabla 59. SA4j. Tabla de la característica Design Smells.	137
Tabla 60. SA4J. Tabla de la característica Activity.	138
Tabla 61. Structure101. Tabla de la característica Target Artefact.	141
Tabla 62. Structure101. Tabla de la característica Design Smells.	144
Tabla 63. Structure101. Tabla de la característica Activity.	145
Tabla 64. StyleCop. Tabla de la característica Target Artefact.	149
Tabla 65. StyleCop. Tabla de la característica Design Smells.	150
Tabla 66. StyleCop. Tabla de la característica Activity.	151
Tabla 67. Together. Tabla de la característica Target Artefact.	155
Tabla 68. Together. Tabla de la característica Design Smells.	158
Tabla 69. Together. Tabla de la característica Activity.	159
Tabla 70. TRex. Tabla de la característica Target Artefact.	163
Tabla 71. TRex. Tabla de la característica Design Smells.	164
Tabla 72. TRex. Tabla de la característica Activity.	165
Tabla 73. XRefactory. Tabla de la característica Target Artefact.	169
Tabla 74. XRefactory. Tabla de la característica Design Smells.	169
Tabla 75. XRefactory. Tabla de la característica Activity.	170
Tabla 76. Tabla que recoge la característica Target Artefact de Analyst4j a StyleCop.	183
Tabla 77. Tabla que recoge la característica Target Artefact de Structure101 a XRefactory.	184
Tabla 78. Tabla que recoge la característica Design Smells de Analyst4j a Structure101.	185
Tabla 79. Tabla que recoge la característica Design Smells de StyleCop a XRefactory.	186
Tabla 80. Tabla que recoge la característica Activity de Analyst4j a FxCop.	187
Tabla 81. Tabla que recoge la característica Activity de inCode a Roodi.	188
Tabla 82. Tabla que recoge la característica Activity de SA4J a XRefactory.	189



Universidad de Valladolid

