Desarrollo de la competencia "Pensamiento Analítico" mediante tácticas de arquitecturas software

José Miguel Cañete Valdeón
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
E.T.S. de Ingeniería Informática. Avenida de la Reina Mercedes, S/N. 41012 Sevilla
jmcv@us.es

Resumen

La competencia "Pensamiento Analítico" se define como el comportamiento mental que permite distinguir y separar las partes de un todo hasta llegar a conocer sus principios o elementos. La inferencia de una arquitectura software a partir de la documentación (normalmente escasa) de un sistema real y de su código fuente es una actividad intelectual compleja que requiere un entrenamiento avanzado de esta competencia en el alumno (niveles superiores en la escala propuesta por los profesores Villa y Poblete de la Universidad de Deusto). Este artículo analiza el empleo de un catálogo de tácticas arquitectónicas como guía para los alumnos en dicha tarea y señala las ventajas e inconvenientes de esta aproximación. El uso de catálogos de tácticas puede generalizarse al contexto de otras asignaturas.

1. Introducción

La competencia genérica instrumental "Pensamiento Analítico" se define como "el comportamiento mental que permite distinguir y separar las partes de un todo hasta llegar a conocer sus principios o elementos. El pensamiento analítico es el pensamiento del detalle, de la precisión, de la enumeración y de la diferencia" [8]. Aunque resulta razonable desarrollar estas cualidades en cualquier alumno universitario, tradicionalmente los estudios de Informática han estado orientados hacia el diseño. Incluso la actividad comúnmente conocida como "análisis de requisitos" tiende a desembocar en la toma de decisiones sobre el software a construir en vez de consistir en un estudio preciso del problema a resolver. Como señala Jackson [4] el problema a resolver mediante software no reside en la interfaz con la máquina sino que se encuentra inmerso en el mundo real y por tanto merece un análisis detallado por parte del informático en el que se estructure el problema y se determinen sus características y dificultades. Sin embargo, además de la ingeniería de requisitos, existen otras áreas dentro de los estudios de Informática donde también se puede desarrollar el pensamiento analítico del alumno.

Así pues, es habitual considerar la enseñanza sobre arquitecturas software como una disciplina centrada en el diseño donde únicamente tienen cabida competencias como el pensamiento creativo o el pensamiento divergente. Sin embargo, resulta muy común encontrar sistemas en explotación donde la arquitectura no ha sido documentada por diferentes motivos (desconocimiento, falta de tiempo o de personal, etc). Sin un documento donde se describa la arquitectura resulta difícil determinar si un sistema informático satisface los requisitos funcionales o los atributos de calidad exigidos en el contrato. Por tanto, la reconstrucción (más precisamente, la inferencia) de arquitecturas software a partir del código fuente y de la documentación disponible es una actividad que tiene interés en sí misma desde una perspectiva profesional. Desde el punto de vista docente resulta una actividad intelectual compleja que requiere un entrenamiento avanzado de la competencia "Pensamiento Analítico" en el alumno. Por "entrenamiento avanzado" entendemos aquel que abarca los niveles de dominio dos y tres en la escala propuesta por los profesores Villa y Poblete [8] para las competencias.

El objetivo de este artículo es la exposición de una experiencia de desarrollo del pensamiento analítico en el alumno a través de la reconstrucción racional de arquitecturas software

en una asignatura de Máster Oficial. Para ello contaremos con el catálogo de tácticas arquitectónicas recopiladas en el Software Engineering Institute (SEI) a partir de experiencias en sistemas reales [1]. Nuestra hipótesis de partida es que el mencionado catálogo de tácticas junto con estrategias de razonamiento adecuadas permitirán al alumno un análisis en profundidad de los sistemas, obteniendo un desarrollo de esta competencia. La evaluación de los resultados se efectuará atendiendo a los indicadores y descriptores propuestos por Villa y Poblete [8].

El resto de este artículo se estructura del siguiente modo. La sección 2 presenta una introducción a las herramientas de análisis proporcionadas al alumno. La sección 3 describe la metodología empleada. Los indicadores para evaluar la competencia y los resultados obtenidos se muestran en la sección 4. El artículo finaliza en la sección 5 con las conclusiones y la bibliografía.

2. Herramientas para el pensamiento analítico en arquitecturas

En esta sección describimos las herramientas propuestas para que los alumnos lleven a cabo la reconstrucción de arquitecturas a partir de código fuente. La organización de las herramientas se apoya en los tres niveles de dominio de la escala propuesta por Villa y Poblete [8] para la competencia del pensamiento analítico. Asociamos cada herramienta con un indicador dentro del nivel, lo que permite conocer hacia qué aspecto del pensamiento analítico está orientada.

2.1. Nivel 1: Ontología arquitectural

En el primer nivel de dominio encontramos, entre otros, el siguiente indicador: "agrupa y describe conjuntos de elementos en categorías preestablecidas". El disponer de categorías al comienzo del análisis es útil para los alumnos, ya que se enfrentan con la totalidad del código fuente y deben comenzar el proceso de abstracción. Wieringa [9] define una *ontología arquitectural* como una meta-clasificación de los distintos tipos de componentes de una arquitectura. Desde el punto de vista estático, el concepto de *Módulo* es el que se suele utilizar para estructurar una arquitectura [1]. Tomaremos la clasificación de módulos propuesta por Wieringa, quien distingue tres categorías: *Transformaciones*, *Almacenes de datos* y *Objetos*. Desde el punto de vista dinámico podemos hablar de *Procesos* en la arquitectura (o hebras "lógicas") y desde la perspectiva del despliegue identificamos *Nodos* [1]. Estas categorías permiten al alumno una primera aproximación hacia la comprensión de los múltiples elementos contenidos en el código.

2.2. Nivel 2: Relaciones (estructuras, estilos arquitectónicos y patrones de diseño)

En el segundo nivel de dominio de esta competencia hallamos, entre otros, el siguiente indicador: "establece relaciones que ordenan los elementos cualitativos". En nuestro caso los "elementos cualitativos" son los módulos hallados, los procesos y los nodos. Bass et al. [1] proponen diversas *estructuras* para relacionar estos elementos, obtenidas a partir de la observación de sistemas reales. La Tabla 1 muestra las más comunes.

Estructuras modulares	Descomposición. Uso. Generalización.
Estructuras de componentes y conectores	Procesos comunicantes. Concurrencia. Datos compartidos. Cliente-servidor.
Estructuras de asignación	Despliegue. Implementación. Asignación del trabajo.

Tabla 1. Estructuras típicas que relacionan los elementos de una arquitectura [1]

Además de estas estructuras existen otros tipos de relaciones entre los módulos. En primer lugar podemos identificar relaciones para organizar módulos con el fin de resolver algún problema más o menos sencillo aunque recurrente: los llamados *patrones de diseño*. Aunque en sentido estricto no pertenecen a la arquitectura y su nivel de abstracción es todavía bajo, los patrones de diseño van a ser hallados por los alumnos durante el análisis del código fuente y pueden ayudarles a identificar otras relaciones entre módulos. Algunos ejemplos de patrones son *Fachada*, *Factoría* y *Cadena de responsabilidades*.

El otro tipo de relaciones organizativas entre módulos son los llamados *estilos* o *patrones*

arquitectónicos, situados en un nivel de abstracción mayor que los patrones de diseño. Como explicaremos más adelante los estilos contribuyen a la satisfacción de los atributos de calidad del sistema ya que son el resultado de aplicar una o varias tácticas. La Tabla 2 muestra algunos de los estilos arquitectónicos recopilados por Wieringa [9].

Estilos principales	Descomposición. Capas.
Estilos de descomposición	Flujos de datos: - Lotes. - Tuberías y filtros. Von Neumann. Orientación a objetos.

Tabla 2. Algunos estilos arquitectónicos [9]

2.3. Nivel 3: Relaciones causa-efecto (tácticas arquitectónicas y atributos de calidad)

En el tercer nivel de domino de la competencia "Pensamiento Analítico" hallamos, entre otros, el siguiente indicador: "establece relaciones causa-efecto o elabora conceptos a partir de elementos cualitativos". Distinguimos dos tipos de relaciones causa-efecto: (1) entre los módulos arquitectónicos y las tácticas y (2) entre las tácticas y los atributos de calidad. En esta sección describiremos ambas relaciones.

Según Bass et al. el diseño de una arquitectura software debe estar dirigido por los atributos de calidad que se hayan determinado para el sistema, y de hecho el método de los autores da preferencia a la asignación de los atributos a los módulos sobre la asignación de las responsabilidades funcionales a los módulos [1, cap. 7]. Los autores argumentan [1, pg. 72] que la funcionalidad se puede conseguir mediante diversas estructuras; de hecho el sistema podría existir incluso como un único módulo monolítico si la funcionalidad fuera el único requisito. Es la necesidad de satisfacer los atributos de calidad lo que conduce al diseño de una arquitectura software.

Los autores distinguen seis atributos de calidad: rendimiento, disponibilidad, seguridad, facilidad de uso (*usability*), capacidad de modificación (*modifiability*) y capacidad para realizar pruebas (*testability*). Para cada uno proponen un número de tácticas que permiten satisfacer el atributo, teniendo en cuenta que una táctica que sea beneficiosa para un atributo puede ser perjudicial para otro. Las tácticas han sido descubiertas por los investigadores del SEI a partir de su experiencia en el diseño de sistemas software, y de hecho Bass et al. acompañan su ensayo con la identificación de tácticas en varios casos de estudio reales (aviónica, control de tráfico aéreo, aplicaciones móviles, J2EE/EJB, etc). La siguiente tabla muestra, a modo de ejemplo, algunas tácticas para la satisfacción de los atributos de calidad.

Atributo de calidad	**Tácticas arquitectónicas**
Rendimiento	Reducir la sobrecarga computacional. Introducir concurrencia.
Disponibilidad	*Ping* / *echo*. Redundancia activa.
Seguridad	Limitar la exposición. Limitar el acceso.
Facilidad de uso	Mantener un modelo del usuario. Separar la interfaz de usuario del resto de la aplicación.
Capacidad de modificación	Generalizar el módulo. Anticipar los cambios esperados.
Capacidad para realizar pruebas	*Record* / *playback*. Monitores internos.

Tabla 3. Atributos de calidad en arquitecturas y tácticas que los satisfacen [1]

Las tácticas se encuentran en un nivel de diseño más abstracto que el de los llamados estilos arquitectónicos. De hecho cada estilo implementa una o varias tácticas [1, pg. 124]. Por ejemplo el estilo *Capas* (*Layers*) [9] puede ser beneficioso para la capacidad de modificación del sistema pero al mismo tiempo puede perjudicar su rendimiento al introducir intermediarios para la comunicación de módulos entre capas no adyacentes.

En este nivel el alumno debe ser capaz de elaborar una relación causa-efecto entre los elementos y relaciones hallados anteriormente (módulos, patrones de diseño, estilos, etc) y las tácticas arquitectónicas que éstos implementan. La relación se puede enunciar de manera más precisa mediante la hipótesis 1:

El conjunto C de elementos y relaciones es una implementación de la táctica T. (1)

La formulación de hipótesis de este tipo exige un dominio importante del pensamiento analítico dado el alto nivel de abstracción de las tácticas arquitectónicas frente al resto de elementos.

Por otro lado, existe una relación causa-efecto entre las tácticas y los atributos de calidad. Como se indicó anteriormente, una táctica puede favorecer un atributo pero ser negativa para la obtención de otro. Por ejemplo, el uso de un intérprete (táctica "generalizar el módulo") favorece la capacidad de modificación en tiempo de ejecución pero tiene una fuerte influencia negativa sobre el rendimiento del sistema [1, pg. 159]. Podemos enunciar la relación como las hipótesis 2 y 3.

La táctica T permite satisfacer el atributo de calidad A, el cual es coherente con los requisitos de este sistema. (2)

La táctica T_1 tiene una influencia negativa sobre el atributo de calidad A, que o bien es irrelevante en este sistema o bien se compensa mediante la táctica T_2. (3)

2.4. Nivel 3: Falta de coherencia en la argumentación (argumento de la ingeniería de sistemas)

Otro de los indicadores que se distinguen en el tercer nivel de domino de la competencia "Pensamiento Analítico" es el siguiente: "identifica lagunas de información o falta de coherencia en la argumentación en textos escritos". El diseño de un sistema informático puede verse como parte de un argumento lógico que ha sido descrito por diversos autores. Wieringa [9] lo denomina "el argumento de la ingeniería de sistemas" y lo formula del siguiente modo:

$$A \wedge S \models E$$

donde *A* denota las afirmaciones contenidas en el modelo (inducido por el ingeniero) acerca del mundo real donde estará contenido el sistema; *S* representa las propiedades del sistema a diseñar; y *E* denota las propiedades que se desea que emerjan en el sistema compuesto (software y entorno). La fórmula es un argumento de que el sistema ha sido implementado correctamente ya que relaciona el software con el mundo real (donde reside el problema, como se explicó en la Introducción).

En nuestro caso podemos restringir el argumento anterior al contexto de las arquitecturas. El alumno puede de este modo comprobar si las propiedades *S* del sistema (los atributos de calidad identificados) conducen o no a que emerjan las propiedades deseadas *E* en el entorno (aquella parte de la realidad donde se va a desplegar el software).

3. Metodología

En esta sección describimos la metodología empleada. En primer lugar explicamos en líneas generales la aplicación del Aprendizaje Basado en Proyectos a la asignatura y a continuación nos centramos en nuestro método para fomentar el desarrollo del pensamiento analítico en el alumno.

3.1. Aprendizaje basado en proyectos

La metodología empleada en la asignatura ha sido descrita en sus líneas generales en [3] por lo que en esta sección sólo resumiremos sus características principales.

Nuestro método combina las clases magistrales con la metodología basada en proyectos de investigación colaborativa [5, 6, 7] y el análisis de casos. La distribución temporal sigue el modelo de la Universidad de Aalborg [5] por lo que se dedica aproximadamente el 50% del tiempo total del alumno al desarrollo del proyecto. Este tiempo se divide en horas de clase dedicadas al proyecto, horas de tutorías y trabajo personal del alumno en el proyecto (aprendizaje investigativo). El otro 50% se reparte entre exposiciones interactivas (en el aula), análisis de casos (en el aula), tutorías y por supuesto el tiempo que el alumno debe dedicar al estudio de los conocimientos teóricos.

La asignatura es cuatrimestral y consta de cinco créditos ECTS lo que equivale a 150 horas de trabajo del alumno. La siguiente tabla muestra el reparto porcentual del esfuerzo del alumno y se ha obtenido a partir de una distribución previa de las 150 horas totales.

	Clases	Tutorías	Estudiar	Total
Proyecto	5.3	6.7	41.3	53.3
Exp.interactiv.	10	4	27.4	41.4
Análisis casos	1.3	0.7	3.3	5.3
Total	16.6	11.4	72	100

Tabla 4. Esfuerzo del alumno (en porcentajes)

Las clases de exposiciones interactivas constan de cinco seminarios, una sesión de exposiciones de proyectos y otra para las conclusiones finales. Las clases de proyectos se refieren al trabajo del alumno sobre el proyecto en el aula, con la guía del profesor. Estas clases se alternaron semanalmente con las exposiciones interactivas y el análisis de casos.

La evaluación del alumno es continua, por un lado mediante exámenes repartidos a lo largo del curso y por el otro mediante las reuniones con los alumnos en tutorías, donde podemos observar el grado de asimilación de las competencias.

Consideramos que los proyectos realizados por los alumnos (reconstrucción de arquitecturas) son trabajos de investigación ya que sus resultados no pueden encontrarse en ninguna referencia y constituyen una aplicación original de una teoría. Por tanto el aula se convierte en un escenario de generación de conocimiento, más allá del transmitido por el profesor. Las memorias de los proyectos son publicadas en la web para que todos los alumnos tengan acceso a las mismas. Durante las exposiciones, los alumnos ejercitan la capacidad de hablar en público y además transmiten a sus compañeros el conocimiento que ellos mismos han descubierto. Para que esto funcione el profesor debe haber corregido previamente las memorias y debe indicar a los alumnos qué tácticas no han sido correctamente identificadas y por qué ya que de otro modo estarían transmitiendo un conocimiento erróneo a sus compañeros.

3.2. Desarrollo del pensamiento analítico

En los dos años de vida de la asignatura su número de alumnos ha sido de veintidós en el primero y de diecisiete en el segundo (no se incluyen aquellos matriculados que no han asistido a clase). Los propios estudiantes se organizan en grupos de trabajo de entre tres y cinco miembros, aunque en ocasiones se han permitido grupos unipersonales. En el segundo año se han aplicado las técnicas preventivas de resolución de conflictos propuestas por Del Canto et al. [2]. Tan sólo ocurrió un conflicto en un grupo (abandono de un miembro y continuas ausencias de otro) que los alumnos solucionaron con éxito. Cada grupo debe eligir un sistema real para analizar su arquitectura. Para ello el profesor propone una lista de sistemas de código abierto. De este modo los alumnos pueden estudiar la implementación así como la documentación disponible: comentarios en el código fuente, sitios web de los sistemas, artículos científicos y libros. Los sistemas propuestos en el curso 2009/10 han sido *Apache Tomcat* (servidor de *servlets*), *Jena* (repositorio y motor de inferencia sobre ontologías), *Sesame* (repositorio y motor de inferencia sobre RDF) y *OpenLaszlo* (servidor para la compilación y ejecución de código LZX).

Dentro de los grupos cada miembro es libre de estudiar el módulo o módulos arquitectónicos que desee, aunque el profesor debe determinar si el tamaño del objeto de estudio es el adecuado.

En todos los sistemas propuestos se pueden encontrar diagramas más o menos formales (en los sitios web, en los libros, etc) bajo el título "arquitectura del sistema". Estos diagramas pueden ser útiles para los alumnos a la hora de hacer un primer reparto de responsabilidades pero se les avisa de que su propósito es el de ofrecer una visión general del sistema a los usuarios o lectores y por tanto carecen del rigor que se requiere en la asignatura.

Los alumnos analizan el código fuente empleando las herramientas descritas en la sección 2. Existen al menos dos esquemas de razonamiento: de abajo arriba y viceversa. El primero consiste en partir del código (por ejemplo, seleccionar un paquete o algunas clases) y estudiar varias trazas de ejecución. Se hace hincapié a los alumnos en que permitan guiarse por su curiosidad y su afán de conocimiento a la hora de estudiar el código. Si llegan a un punto del código que les resulta incomprensible es preferible anotarlo y continuar en otro. Durante este reconocimiento preliminar los alumnos pueden identificar elementos interesantes y clasificarlos mediante la ontología arquitectural. También resulta habitual que hallen numerosos patrones de diseño y comiencen a identificar las estructuras que relacionan los módulos entre sí, especialmente las de tipo "usa", lo que dará una indicación del estilo arquitectónico predominante

(encapsulación o capas). Algunos patrones de diseño también darán pistas sobre estilos arquitectónicos secundarios (por ejemplo el patrón *Cadena de responsabilidades* señala un estilo *Flujo de datos*).

La comunicación de los alumnos entre sí en paralelo a sus análisis individuales les llevará a descubrir relaciones entre módulos e indicios de los estilos existentes. Mientras que avanzan en su comprensión del código irán identificando composiciones de módulos para la consecución de las tácticas del catálogo. Esta identificación sólo es posible si previamente han estudiado el catálogo de tácticas. Para comprobar este punto realizamos un examen a mitad de curso centrado exclusivamente en el catálogo.

El esquema de razonamiento de arriba abajo comienza por formular conjeturas acerca de los atributos de calidad que sea razonable que posea el sistema a estudiar, dadas las características de éste. Por ejemplo, de un servidor como *Tomcat* se espera que esté siempre en ejecución por lo que podemos conjeturar que el atributo "disponibilidad" estará entre sus requisitos. A partir de los atributos se puede consultar el catálogo y obtener las tácticas más comunes para conseguirlos en sistemas reales. Seguidamente se inspecciona el código con el objetivo de localizar elementos que apoyen las conjeturas iniciales y que por tanto implementen alguna de las tácticas. Por ejemplo, una táctica de disponibilidad es el uso de excepciones para localizar defectos. Por tanto el alumno buscaría dichas excepciones en el código, sabiendo que no todas las excepciones Java se emplean para la notificación de defectos en el sistema.

Durante el proceso de reconstrucción racional de una arquitectura es habitual emplear ambos esquemas de razonamiento.

Como se ha indicado previamente, realizamos un seguimiento continuo del desarrollo analítico de los alumnos: en el aula (sesiones de proyectos) y en las tutorías obligatorias. Resulta frecuente que los estudiantes acudan a las tutorías con relaciones de tácticas identificadas y nos pregunten si son correctas. Ante esto optamos por requerirles que hagan el esfuerzo de construir un argumento coherente que justifique la existencia de cada una de las tácticas. Una vez expuesto razonamos con ellos acerca de la corrección del mismo.

Consideramos fundamental el que los alumnos se habitúen a exponer sus resultados según el estilo científico: formulación de hipótesis de partida, exposición de datos empíricos, análisis razonado de los datos y obtención de conclusiones que respalden o rechacen las hipótesis. El estilo científico estimula a los alumnos a organizar sus pensamientos y a describirlos formalmente. Para los profesores resulta también muy interesante ya que los enunciados son claros y las conclusiones pueden ser evaluadas de manera precisa comprobando los datos y la corrección de los argumentos. Los alumnos de Informática no suelen estar habituados a este modo de escritura por lo que se les proporciona una plantilla de documento a la que deben ajustarse. La Figura 1 muestra un resumen de la misma.

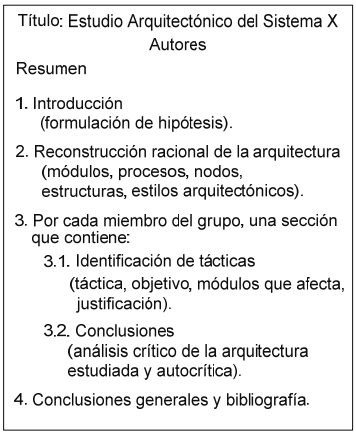

Figura 1. Plantilla para la memoria del proyecto

4. Indicadores y resultados

La evaluación del pensamiento analítico se realiza atendiendo a los indicadores propuestos por Villa y Poblete [8] para dicha competencia. Como ya se ha comentado con anterioridad, los autores proponen tres niveles de dominio; cada uno contiene entre cinco y seis indicadores. En una asignatura de Máster es razonable concentrarse en los dos niveles superiores; de los once indicadores asociados a dichos niveles hemos seleccionado cinco por su relevancia en el contexto de la reconstrucción racional de arquitecturas. Asimismo hemos considerado interesante

seleccionar un indicador de primer nivel. Seguidamente hemos adaptado el conjunto resultante de indicadores a las herramientas de análisis (sección 2). Cada uno tiene asociado cinco descriptores o grados de asimilación o de progreso por parte del alumno aunque, por motivos de espacio, no hemos incluido la descripción de los mismos. Las personas interesadas pueden consultarlos en la referencia original [8, pgs. 65-67]. A continuación describimos los indicadores empleados:

- *Indicador 1 (nivel 1)*: agrupa y describe conjuntos de elementos del código en categorías preestablecidas (ontología arquitectural).
- *Indicador 2 (nivel 2)*: identifica módulos principales y secundarios.
- *Indicador 3 (nivel 2)*: establece relaciones entre los módulos, procesos y nodos (estructuras, patrones de diseño, estilos arquitectónicos).
- *Indicador 4 (nivel 3)*: establece relaciones causa-efecto entre los módulos y las tácticas o entre las tácticas y los atributos de calidad.
- *Indicador 5 (nivel 3)*: identifica lagunas de información o falta de coherencia en el argumento de la ingeniería de sistemas.
- *Indicador 6 (nivel 3)*: al expresar sus ideas y conclusiones se apoya en datos y en la relación entre los mismos.

		Descriptores				
Nivel	Indicador	1	2	3	4	5
Primero	1	7	4	3	3	0
Segundo	2	1	2	10	4	0
	3	2	3	11	1	0
Tercero	4	0	3	6	8	0
	5	8	2	1	6	0
	6	1	8	8	0	0

Tabla 1. Distribución de los resultados de la evaluación

La Tabla 5 muestra los resultados de la evaluación de los distintos indicadores en los alumnos del curso 2009/10, el primero en el que hemos utilizado estas métricas. Los descriptores de progreso están numerados del 1 (mínimo progreso) al 5 (máximo). Un número *n* en la intersección del indicador *i* y del descriptor *d* indica que *n* alumnos consiguieron un grado de asimilación *d* en el indicador *i* de la competencia. Por tanto cada fila suma diecisiete (el total de alumnos).

De los datos mostrados podemos extraer varias conclusiones. El indicador 1 señala que es necesario fomentar más la capacidad de abstracción de los alumnos ya que siete de ellos no supieron aplicar correctamente la ontología arquitectural y realizaron sus razonamientos al nivel del código fuente.

Observando el indicador 2 podemos concluir que en general los alumnos han demostrado ser capaces de identificar correctamente los módulos principales y secundarios de la arquitectura.

El indicador 3 nos muestra que prácticamente todos los alumnos han sido capaces de identificar relaciones significativas: estructuras, patrones de diseño y estilos arquitectónicos.

El indicador 4 señala que la gran mayoría de los alumnos (catorce sobre diecisiete) ha sido capaz de establecer correctamente relaciones causa-efecto entre los módulos identificados y las tácticas arquitectónicas.

El indicador 5 nos muestra que más de la mitad de los alumnos (diez de diecisiete) no ha podido identificar críticas al diseño original del sistema elegido o, si lo ha hecho, éstas han sido pobres. Sin embargo seis alumnos han sido capaces de identificar problemas de diseño en el sistema estudiado y de proponer mejoras satisfactorias.

Observando el último indicador, podemos afirmar que la capacidad de razonamiento para la obtención de conclusiones se encuentra dividida. Ocho alumnos se apoyan en datos pero sólo tienen en cuenta los que avalan su opinión, (descriptor 2) mientras que otros tantos sí utilizan todos los datos y sus relaciones como argumentos al exponer sus ideas (descriptor 3).

Llama la atención que ha sido en el nivel más bajo de dominio donde los alumnos han demostrado tener más problemas. Muchos han saltado directamente desde la identificación de los conceptos principales del código fuente a la identificación de patrones de diseño y estilos y de ahí a las tácticas. Esto ha tenido una consecuencia negativa: les ha obligado a razonar empleando conceptos de bajo nivel, con el consiguiente deterioro de los dos indicadores que están directamente relacionados con el razonamiento: por un lado el indicador 5, donde más de la mitad de los alumnos no ha podido identificar críticas al

diseño original; y por otro lado el indicador 6, que mide la capacidad de expresión de ideas y conclusiones, ha quedado fuertemente polarizado.

En resumen, podemos afirmar que la experiencia ha sido en general positiva, ya que gracias al conocimiento de las tácticas los alumnos han sido capaces de realizar un análisis complejo como es la reconstrucción de una arquitectura a partir del código fuente. Sin embargo resulta evidente que debemos invertir más esfuerzo en la mejora de la capacidad de abstracción de los estudiantes y en que se acostumbren a razonar con conceptos más generales, dejando a un lado las ontologías que proporcionan los lenguajes de programación.

5. Conclusiones

En este artículo hemos presentado una experiencia en el desarrollo del pensamiento analítico a través de proyectos de investigación donde los alumnos deben inferir una arquitectura software a partir del código fuente. Para ello ha sido imprescindible el empleo de un catálogo de tácticas arquitectónicas extraídas de sistemas reales. La evaluación de los alumnos se ha realizado siguiendo la escala propuesta por los profesores Villa y Poblete [8] y los resultados avalan la eficacia de nuestro método, pese a que se pueden identificar problemas que no han sido resueltos en relación a la capacidad de abstracción de los alumnos y el razonamiento con ontologías de alto nivel.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el I Plan Propio de Docencia de la Universidad de Sevilla (acciones del curso 2009/10).

Referencias

[1] Bass, L., Clements, P., y Kazman, R. *Software Architecture in Practice (2nd ed).* Addison Wesley, 2003.

[2] Canto, P.; Gallego, M.I.; López, J.M.; Mora, F.J.; Reyes, M.A.; Rodríguez, E.; Kanapathipillai, J.; Santamaría, E.; Valero, M. Conflictos en el trabajo en grupo: dos casos representativos. *Actas de las XV Jornadas de Enseñanza Universitaria de la Informática.* Barcelona, 2009.

[3] Cañete Valdeón, J.M. y Martín Díaz, O. Una experiencia en el diseño y la impartición de una asignatura en torno a la metodología del aprendizaje basado en proyectos. *Actas de las XV Jornadas de Enseñanza Universitaria de la Informática*. Barcelona, 2009.

[4] Jackson, M.A. *Problem Frames: Analysing and Structuring Software Development Problems.* Addison-Wesley, 2001.

[5] Kolmos, A. Estrategias para desarrollar currículos basados en la formulación de problemas y organizados en base a proyectos. *Educar*, 33, págs. 77-96, 2004.

[6] López Ruiz, J.I. La docencia basada en proyectos de investigación: una experiencia en la Educación Superior. En Zabalza, M.A. (Ed.). *La calidad de la docencia en la Universidad*. Ágora, 2000.

[7] Reverte Bernabeu, J.R., Gallego Sánchez, A.J, Molina Carmona, R. y Satorre Cuerda, R. El aprendizaje basado en proyectos como modelo docente. Experiencia interdisciplinar y herramientas groupware. *Actas de las XIII Jornadas de Enseñanza Universitaria de la Informática*. Teruel, 2007.

[8] Villa, A. y Poblete, M. *Aprendizaje basado en competencias. Una propuesta para la evaluación de las competencias genéricas.* Ediciones Mensajero, 2007.

[9] Wieringa, R.J. *Design Methods for Reactive Systems.* Morgan-Kaufmann, 2003.