UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA

EN INFORMÁTICA

ANTEPROYECTO

GCAD: Plug-in de Eclipse para la **G**estión de **C**onocimiento en las fases de **A**nálisis y **D**iseño de desarrollo distribuido

Alumno: Juan Andrada Romero
Directora: Aurora Vizcaíno Barceló

Marzo, 2011

ÍNDICE

1 Introducción **1**

2 Objetivos **2**
2.1 Objetivos del proyecto 2
2.2 Fundamentos teóricos 3

3 Método y Fases de trabajo **7**

4 Medios que se pretenden utilizar **8**

Bibliografía **9**

1. INTRODUCCIÓN

La manera de desarrollar software está evolucionando en los últimos años, tendiendo a un desarrollo distribuido del mismo. Ésto se debe, en cierta medida, a la globalización del mercado, de tal modo que se ha cambiado el concepto de *mercado nacional* por el de *mercado global* [5] [11]. De este modo, y cada vez con más frecuencia, el desarrollo del software es llevado a cabo por diferentes equipos, que se pueden encontrar distribuidos en diferentes continentes. Es lo que se conoce como **Desarrollo Global de Software** (DGS).

Una de las principales ventajas del DGS es que puede aumentar la productividad a la hora de desarrollar el software, ya que la jornada laboral puede ser extendida debido a las diferencias horarias existentes en los diferentes países donde se encuentran los equipos de desarrollo. Otras de las ventajas del DGS es que se pueden abaratar costes en mano de obra de diferentes países y se puede mejorar la presencia en el mercado internacional, aumentando la competitividad.

Para aprovechar al máximo las ventajas que proporciona el DGS, es imprescindible una correcta **Gestión de Conocimiento**, que permita registrar, almacenar y organizar todo el conocimiento (ideas, decisiones, experiencias, artefactos, etc.) generado por los diferentes equipos de desarrollo distribuidos, para que dicho conocimiento pueda ser utilizado por otros en cualquier momento y lugar [11].

Por otro lado, la deslocalización de los equipos de desarrollo es uno de los principales inconvenientes del DGS, pues surgen problemas de comunicación y coordinación [8] [11] [22], que derivan en problemas a la hora de plantear decisiones y alternativas o de llegar a acuerdos. Además, al estar los equipos de desarrollo distribuidos en diferentes países, aparecen también problemas asociados a malentendidos que pueden provocarse por las diferencias culturales, ya que los empleados de las empresas de cada país tendrán su propia forma de trabajar, el idioma es diferente, etcétera. Todo ello provoca que sea más fácil cometer errores a la hora de gestionar conocimiento asociado al desarrollo de un proyecto, así como se dificulta también la utilización de dicho conocimiento.

Para intentan paliar los problemas comentados anteriormente, se propone el desarrollo de una herramienta visual para dar soporte a la gestión de conocimiento en la fase de análisis y diseño de un producto software, aplicándose al desarrollo global de software. De este modo, gracias al uso de esta herramienta, se puede recoger y visualizar el conocimiento de un modo intuitivo, evitando los problemas de comunicación y malentendidos debidos a las diferencias culturales entre los diferentes países.

2. OBJETIVOS

En esta sección se presentará el objetivo que se pretende alcanzar con el desarrollo del Proyecto Fin de Carrera (en adelante PFC).

2.1. Objetivos del proyecto

El objetivo principal del PFC consistirá en el diseño y construcción de una herramienta visual que permita dar soporte a la gestión de decisiones en desarrollo software. Concretamente, la herramienta debe dar soporte a las decisiones planteadas en las fases de análisis y diseño de un producto software.

Dicha herramienta se desarrollará en forma de plug-in para el IDE Eclipse [9], uno de los más utilizados por las empresas en el ámbito de desarrollo de software.

Como se ha comentado en la sección 1, con esta herramienta visual se intenta reducir o eliminar una de las problemáticas que aparece en DGS, como es la falta de comunicación y control, así como los malentendidos que pueden surgir a la hora de tomar decisiones en el desarrollo de un producto software.

Por esta razón, la herramienta debe cumplir con los siguientes objetivos:

- Almacenar conocimiento en forma de alternativas y decisiones planteadas en las fases de análisis y diseño de un producto software.
- Mostrar de una forma visual e intuitiva las diferentes alternativas y decisiones que han sido tomadas en las fases de análisis y diseño de un producto software.
- Notificar nuevas contribuciones y conocimiento disponible.
- Proporcionar información acerca de quién aporta una alternativa o decisión. Dicha información puede ser el rol de esa persona, el país donde se encuentra trabajando, experiencia acumulada en desarrollo de software, antigüedad en la empresa, etcétera.
- Aconsejar decisiones de análisis y diseño que hayan sido exitosas en otros proyectos con características similares. Para ello, se pretende utilizar técnicas de Razonamiento Basado en Casos (Cases Based Reasoning - CBR). [1]
- Informar sobre la contribución de cada uno de los miembros del proyecto, ofreciendo estadísticas que ayuden a conocer qué personas aportan más. Dichas estadísticas podrán visualizarse en forma de diagramas de barras, diagrama circular, etcétera.
- Exportar el conocimiento utilizando un estándar para el intercambio de información estructurada, como es XML [28]. También se permitirá exportar imágenes de los gráficos generados en formato PNG [24].

2.2. Fundamentos teóricos

Los conceptos teóricos que intervienen en este PFC son la Gestión de Conocimiento, *Design Rationale*, el Razonamiento Basado en Casos y el desarrollo distribuido de software, donde nos centraremos en el Desarrollo Global de Software. Éste último ya ha sido descrito brevemente en la sección 1, por lo que se presenta a continuación el resto de conceptos.

Gestión del Conocimiento

Aunque actualmente no existe un consenso acerca del concepto de gestión del conocimiento, se podría definir como un proceso que permite capturar, crear, identificar, almacenar, buscar, recuperar, compartir, transferir y diseminar el conocimiento existente dentro de una organización, con el fin de incrementarlo y evitar su pérdida y sub-utilización [4] [13] [18].

Por otra parte, el conocimiento se define como "*una mezcla fluida de experiencias enmarcadas, valores, información contextual, y pericia que provee de un marco para evaluar e incorporar nuevas experiencias e información. Esto se origina y es aplicado en la mente de quienes conocen. En las organizaciones, con frecuencia viene embebido no solo en documentos o repositorios, sino también en rutinas, procesos, prácticas o normas de la organización*" [6]. En el caso concreto de una organización dedicada al desarrollo software, el conocimiento que más interesa gestionar son las alternativas y decisiones planteadas en el análisis y diseño del producto software, los diferentes diagramas, la documentación, las pruebas del software, etc.

Centrándose en la gestión de conocimiento en organizaciones especializadas en desarrollo de software, se presenta en el siguiente apartado el concepto de *Design Rationale*.

Design Rationale

Design Rationale (o *Rationale* simplemente) es "*un método que permite capturar, representar y mantener registros de información acerca de las decisiones que son tomadas por los miembros de un equipo de desarrollo de software*" [7]. Dicho método puede ser aplicado en cualquier fase del desarrollo, desde el análisis de requisitos hasta las pruebas.

De este modo, se pueden capturar todas las decisiones y alternativas que se van produciendo a lo largo del desarrollo, pudiendo discutir y decidir sobre éstas. Además, dichas decisiones quedarán almacenadas como nuevo conocimiento para la empresa y se podrá reutilizar en proyectos futuros, sobre todo aquellas decisiones que hayan sido exitosas.

A la hora de representar el conocimiento, existen, principalmente, dos enfoques [7]:

- **Causal Graph**: es un grafo causal donde cada nodo representa una decisión y los arcos representan restricciones entre ellas, como por ejemplo que una decisión depende o proviene de otra anterior. En la Figura 2.1 puede observase un ejemplo de este tipo de grafo.

- **Dialogue Map**: es un grafo donde los nodos pueden representar una pregunta, una idea o un argumento a favor o en contra de dichas ideas y preguntas. De este modo, se puede ir discutiendo una decisión e ir añadiendo nuevas alternativas y justificaciones. En la Figura 2.2 puede observarse un ejemplo. Este tipo de grafo se basa en los sistemas **IBIS** (**I**ssue-**B**ased **I**nformation **S**ystems), donde el conocimiento se almacena y representa de manera jerárquica, en forma de decisiones/preguntas y justificaciones de éstas.

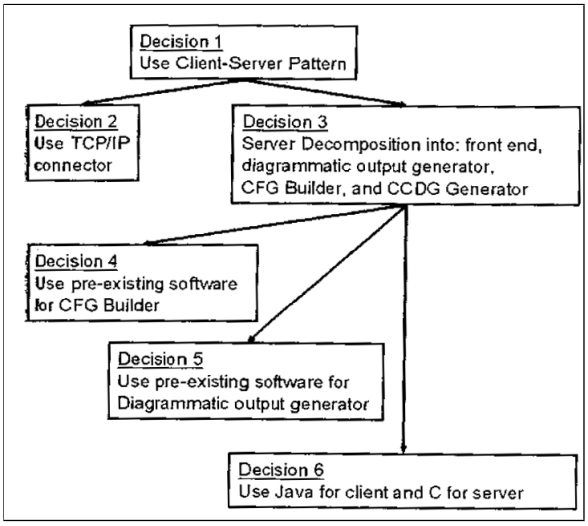

Figura 2.1: Causal Graph

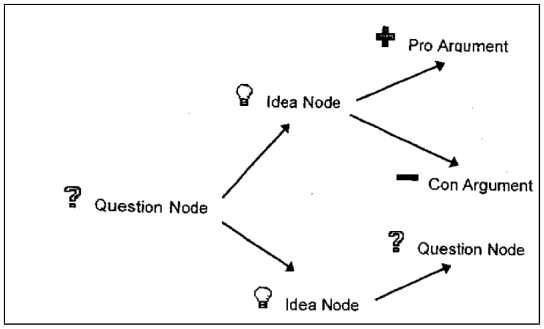

Figura 2.2: Dialogue Map

Razonamiento Basado en Casos

El Razonamiento Basado en Casos (Case-Based Reasoning, CBR en adelante) puede definirse como "*un proceso en el que experiencias específicas son recuperadas, reutilizadas, revisadas y almacenadas para utilizarse en la solución de problemas similares.*" [2] De este modo, se pueden encontrar soluciones a nuevos problemas basándose en soluciones de problemas similares anteriores. Dichas soluciones puede que sea necesario adaptarlas para resolver el nuevo problema y se almacenarán en forma de nuevas experiencias (casos) para

poder reutilizarlas en un futuro. Es por esto que un sistema CBR puede *"aprender"* a partir de nuevas soluciones. [19]

A diferencia de otras técnicas y algoritmos de Inteligencia Artificial, como pueden ser Razonamiento Basado en Reglas o Algoritmos Genéticos, CBR no se considera como una tecnología, sino más bien como una metodología, que indica cómo resolver problemas a partir de soluciones previas almacenadas en el sistema, pero sin detallar una tecnología concreta [27]. Por tanto, un sistema CBR se define por lo que hace, pero no cómo se hace o se implementa.

Conceptualmente, el CBR se describe como un ciclo (ver Figura 2.3) con cuatro grandes etapas [21] [27]:

- **Recuperación (Retrieve)** de casos similares al problema planteado.
- **Reutilización (Reuse)** de una solución propuesta por un caso similar.
- **Revisión (Revise)** de la solución propuesta, para adaptarla mejor a las condiciones del nuevo problema.
- **Retención (Retain)** de la nueva solución, pasando a formar un nuevo caso.

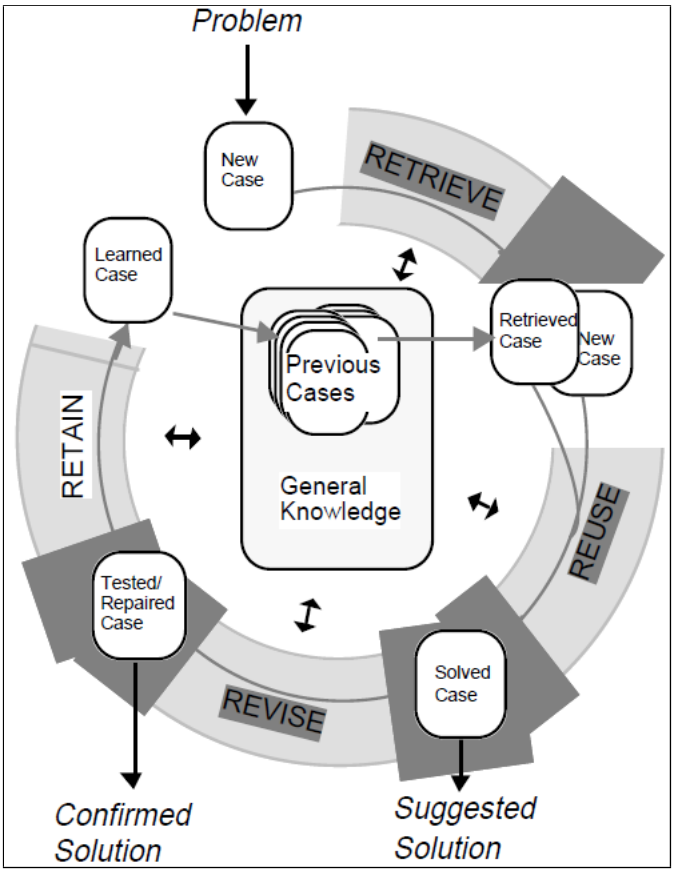

Figura 2.3: Ciclo del CBR

Estas cuatro etapas son las que constituyen la metodología del CBR. Así, para solucionar un nuevo problema, primero se debe obtener una descripción de éste, midiendo la similitud

del nuevo problema con otros problemas previos almacenados en el sistema. A continuación, se recuperan las soluciones de estos problemas similares y se reutiliza la solución de uno de estos casos, adaptándola si es necesario. Para terminar, este nuevo problema, junto a la solución encontrada, se almacena en el sistema, formando un nuevo caso.

Uno de los puntos críticos es la función de similitud que se va a utilizar para buscar problemas similares a uno dado. Por ejemplo, un problema se podría describir como una serie de atributos (o características) que pueden ser cuantificadas con un "peso" numérico, obteniendo un vector numérico, y utilizar como función de similitud la distancia Euclídea entre los vectores que representan cada problema.

Para terminar, el ciclo que se ha visto anteriormente encaja perfectamente con un sistema de Gestión del Conocimiento, pues en este tipo de sistemas también existen fases de recuperación, reutilización y almacenamiento, en este caso de conocimiento.

3. MÉTODO Y FASES DE TRABAJO

Para el desarrollo de la herramienta se ha optado por utilizar la metodología genérica descrita por el **Proceso Unificado de Desarrollo** (PUD). El PUD [14] es una evolución del Proceso Unificado de Rational (RUP), que define un "conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software". Es un marco genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos.

Las principales características del PUD son:

- **Dirigido por casos de uso**. Un caso de uso representa un requisito funcional al cuál el sistema debe dar soporte para proporcionar un resultado de valor al usuario. Los casos de uso guían el proceso de desarrollo, ya que basándose en los casos de uso, los desarrolladores crean una serie de modelos para poder llevarlos a cabo. Todos los casos de uso juntos constituyen el **modelo de casos de uso**.
- **Centrado en la arquitectura**. Un sistema software puede contemplarse desde varios puntos de vista. Por tanto, la arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema y debe estar profundamente relacionada con los casos de uso, ya que debe permitir el desarrollo de los mismos.
- **Iterativo e incremental**. El esfuerzo de desarrollar un proyecto de software se divide en partes más pequeñas, llamadas **mini-proyectos**. Cada mini-proyecto es una **iteración** que resulta en un **incremento**. Las iteraciones deben estar controladas y deben seleccionarse y ejecutarse de una forma planificada, siguiendo el esquema *requisitos, análisis, diseño, implementación y pruebas*, que es conocido como **flujo de trabajo**. En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, implementan el diseño mediante componentes y verifican que los componentes satisfacen los casos de uso.

Debido al enfoque iterativo e incremental que caracteriza al PUD, éste se divide en ciclos, donde cada ciclo se compone de cuatro fases y éstas, a su vez, se dividen en iteraciones que siguen el flujo de trabajo *requisitos, análisis, diseño, implementación y pruebas.*

Las cuatro fases que constituyen un ciclo son:

- **Iniciación**. Se obtiene un modelo de casos de uso simplificado, se identifican riesgos potenciales y se estima el proyecto de manera aproximada.
- **Elaboración**. Se especifican en detalle la mayoría de los casos de uso del producto software y se diseña la arquitectura, obteniendo la línea base de la arquitectura.
- **Construcción**. Se crea el producto en base a las dos fases anteriores. La línea base de la arquitectura crece hasta convertirse en el sistema completo.
- **Transición**. Implica la corrección de errores y el mantenimiento del sistema.

4. MEDIOS QUE SE PRETENDEN UTILIZAR

Para el desarrollo del PFC se utilizarán una serie de medios software, tanto para la especificación y definición del sistema, como para su implementación. Dichos medios son:

- **Eclipse** [9] (*versión 3.6 Helios*), ya que la herramienta a desarrollar es un plug-in para dicho IDE.
- **Java** [15].
- **Apache Commons Configuration** [3], para leer archivos de configuración de diferentes tipos (*Properties*, *XML*, etcétera).
- **JAXB (Java Architecture for XML Binding)** [16], para poder mapear y exportar clases Java a un fichero XML.
- **Zest: The Eclipse Visualization Toolkit** [29], para visualizar el conocimiento en forma de grafo o árbol.
- **JFreeChart** [17], para mostrar gráficos.
- **MySQL** [23], para la elaboración de la base de datos de conocimiento con la que debe comunicarse la herramienta.
- **Hibernate** [12], para gestionar la persistencia.
- **Visual Paradigm** [26], para la realización de los diferentes diagramas de la herramienta.
- **LaTeX** [20], para la elaboración de la documentación.
- **Subversion** [25], utilizado como sistema de control de versiones. Además, se ha utilizado **Google Code** [10] para alojar una copia del desarrollo del PFC, aprovechando el control de versiones que éste provee.

BIBLIOGRAFÍA

[1] Aamodt, A. y E. Plaza: *Case-based reasoning: Foundational issues, methodological variations, and system approaches.* AI communications, 7(1):39–59, 1994, ISSN 0921-7126.

[2] Aha, D., C. Marling y I. Watson: *Case-based reasoning commentaries: introduction.* The Knowledge Engineering Review, 20(03):201–202, 2005, ISSN 0269-8889.

[3] Apache Commons Configuration. http://commons.apache.org/configuration/.

[4] Bornemann, M., M. Graggober, E. Hartlieb, B. Humpl, P. Koronakis, A. Primus, K. Ritsch, H. Rollett, M. Sammer, J. Tuppinger, R. Willfort y K. Wöls: *An Illustrated Guide to Knowledge Management*, 2003.

[5] Damian, D. y D. Moitra: *Guest Editors' Introduction: Global Software Development: How Far Have We Come?* Software, IEEE, 23(5):17–19, 2006.

[6] Davenport, T. y P. L.: *Working Knowledge: How organizations manage what they know.* Harvard Business School Press, 2000.

[7] Dutoit, A., R. McCall, I. Mistrik y B. Paech: *Rationale management in software engineering.* Citeseer, 2006.

[8] Ebert, C. y P. De Neve: *Surviving global software development.* Software, IEEE, 18(2):62–69, 2002.

[9] Eclipse Home. http://www.eclipse.org/.

[10] Google Code Project Hosting. http://code.google.com/intl/es-ES/projecthosting/.

[11] Herbsleb, J. y D. Moitra: *Global Software Development.* Software, IEEE, 18(2):16–20, 2002.

[12] Hibernate Home Page. http://www.hibernate.org/.

[13] Hovland, I.: *Knowledge management and organisational learning: An international development perspective.* Overseas Development Institute Working Paper No. 224, Overseas Development Institute, London, UK.

[14] Jacobson, I., G. Booch y J. Rumbaugh: *El proceso unificado de desarrollo de software.* Addison Wesley, 2000.

[15] Java Home. http://www.java.com/es/.

[16] JAXB: Java Architecture for XML Binding. http://jaxb.java.net/.

[17] JFreeChart Home. http://www.jfree.org/jfreechart/.

[18] Kanagasabapathy, K., R. Radhakrishnan y S. Balasubramanian: *Empirical investigation of critical success factor and knowledge management structure for successful implementation of knowledge management system: A case study in process industry.* 2006.

[19] Kolodner, J.: *An introduction to case-based reasoning.* Artificial Intelligence Review, 6(1):3–34, 1992, ISSN 0269-2821.

[20] LaTeX – A document preparation system. http://www.latex-project.org/.

[21] Lopez De Mantaras, R., D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus *y cols.*: *Retrieval, reuse, revision and retention in case-based reasoning.* The Knowledge Engineering Review, 20(03):215–240, 2005, ISSN 0269-8889.

[22] Mohagheghi, P.: *Global Software Development: Issues, Solutions, Challenges.* Retrieved April, 7:2009, 2004.

[23] MySQL Home. http://www.mysql.com.

[24] PNG: Portable Network Graphics. http://www.w3.org/Graphics/PNG/.

[25] Subversion Project. http://subversion.tigris.org/.

[26] Visual Paradigm Home. http://www.visual-paradigm.com/.

[27] Watson, I.: *CBR is a methodology not a technology.* Research & Development in Expert Systems XV, págs. 213–223, 1998.

[28] XML: eXtensible Markup Language. http://www.w3.org/XML/.

[29] Zest: The Eclipse Visualization Toolkit. http://www.eclipse.org/gef/zest/.

ANEXO I. SOLICITUD DE EVALUACIÓN DEL ANTEPROYECTO

Escuela Superior de Informática

D. Juan Andrada Romero, con D.N.I. nº 05930408-L, alumno de la Escuela Superior de Informática

EXPONE:

– Que se encuentra matriculado del PFC o en condiciones de hacerlo en el siguiente periodo de matriculación por tener aprobado el primer ciclo completo.

SOLICITA:

– Que la Comisión Académica evalúe el anteproyecto de fin de carrera que acompaña, titulado **GCAD**: Plug-in de Eclipse para la **G**estión de **C**onocimiento en las fases de **A**nálisis y **D**iseño de desarrollo distribuido

Ciudad Real, a 16 de Marzo de 2011.

El alumno, VºBº La directora,

Fdo.: Juan Andrada Romero Fdo.: Aurora Vizcaino Barcelo

ILMO. SR. DIRECTOR DE LA ESCUELA SUPERIOR DE INFORMATICA