UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

GCAD: Sistema Cliente/Servidor para la Gestión de Decisiones en Desarrollo Distribuido de Software.

Juan Andrada Romero

Febrero, 2012

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

DEPTO. DE TECNOLOGÍAS,
Y SISTEMAS DE INFORMACIÓN

PROYECTO FIN DE CARRERA

GCAD: Sistema Cliente/Servidor para la Gestión de Decisiones en Desarrollo Distribuido de Software.

Autor: Juan Andrada Romero

Director: Aurora Vizcaíno Barceló

Febrero, 2012

GCAD: Sistema Cliente/Servidor para la Gestión de Decisiones en Desarrollo Distribuido de Software.

TRIBUNAL:

Presidente: _____

Vocal 1: _____

Vocal 2: _____

Secretario: _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Resumen

Abstract

... english version for the abstract ...

Agradecimientos

Escribir agredecimientos

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Índice de algoritmos

Capítulo 1

Motivación y Objetivos del Proyecto

En este capítulo se presenta la motivación para el desarrollo del presente PFC, así como los objetivos que deben alcanzarse tras concluir el desarrollo del mismo.

1.1 Motivación

La globalización y competitividad de los mercados ha obligado, y está obligando, a que las empresas vayan evolucionando y adaptando sus procesos de negocio y sus procesos de gestión de proyectos para adaptarse a esta nueva tendencia de mercados globales. Estas nuevas estrategias requieren de cooperación intensiva entre las compañías, de nuevos procesos de gestión de proyectos, de mejoras en las comunicaciones entre organizaciones que se encuentran en diferentes husos horarios, etc. Todo ello ha provocado que aparezcan nuevas organizaciones, llamadas *organizaciones virtuales*, que son aquellas que se encuentran deslocalizadas, con proyectos en los que participan empleados de cualquier parte del mundo y que utilizan las tecnologías de la información para sus procesos de comunicación, coordinación y control [14].

Anteriormente a este marco de globalización, las empresas gestionaban sus proyectos de manera co-localizada, es decir, en el mismo lugar y al mismo tiempo. Sin embargo, ahora se ven obligadas a desarrollar procesos de gestión que permitan la gestión de *proyectos virtuales*, es decir, proyectos que ya no están centralizados en un único lugar y se realizan de manera distribuida. En la Figura 1.1 se muestran las dimensiones de los proyectos que las compañías deben gestionar [14].

Sin embargo, este marco de *proyectos virtuales* y, por tanto, del desarrollo de los proyectos

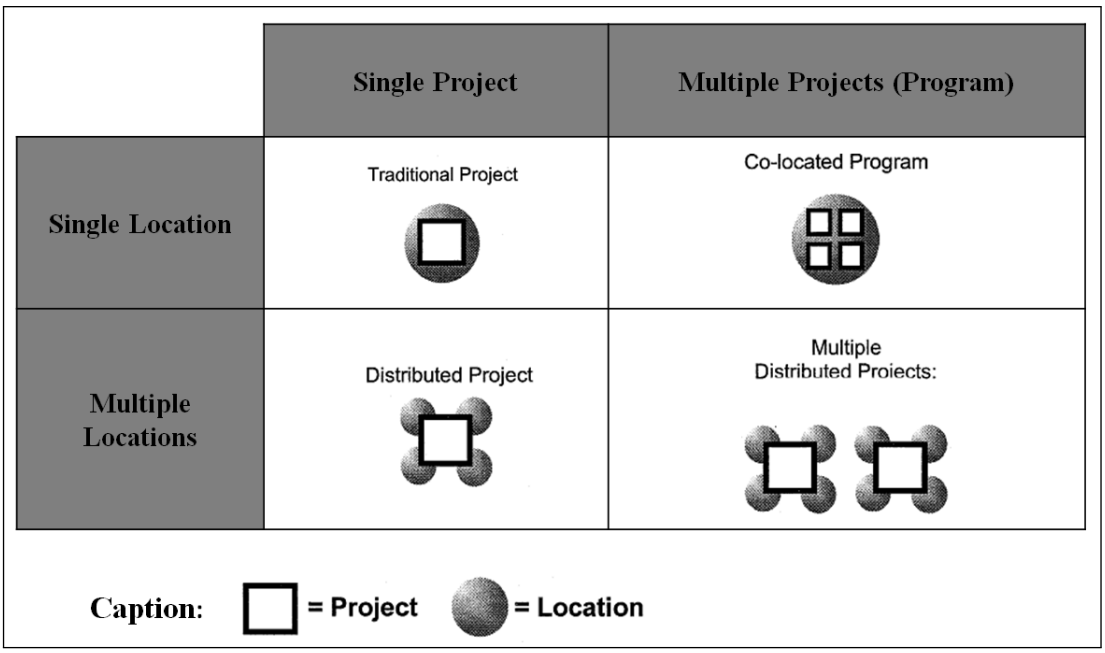

Figura 1.1: Proyectos según su localización

de manera distribuida, acarrea una serie de dificultades, siendo las más destacables los problemas de comunicación, coordinación y control entre las diferentes localizaciones de los centro de desarrollo de las organizaciones.

Otro de los problemas encontrados, muy relacionado con el anterior, es como las empresas, que ahora se encuentran en localizaciones diferentes y que tienen mayor problema para la comunicación y control, pueden crear, organizar y gestionar su conocimiento organizacional, recurso intangible imprescindible para que la empresa pueda innovar y ser competitiva.

Centrándonos en empresas dedicadas al sector de desarrollo software, gran parte de este conocimiento organizacional lo componen las decisiones tomadas durante el ciclo de vida de desarrollo de un proyecto software, tales como decisiones de diseño, decisiones de análisis del proyecto, etc., que con frecuencia no se documentan de manera adecuada o se documentan por un equipo de desarrollo y el resto no son conscientes de que dicha documentación existe, dificultando la posterior reutilización de dicho conocimiento. Este es un problema importante del desarrollo global de software, ya que al generarse conocimiento, información y toma de decisiones en los diferentes equipos de desarrollo, ésta queda distribuida en numerosas ocasiones si no es guardada previamente en un repositorio global. Es por este motivo que se produce una pérdida de dicha información y no se aprende de las decisiones tomadas en los proyectos.

Por tanto, la motivación de este PFC viene dada por la necesidad de resolver o minimizar algunos de los problemas que aparecen en las compañías que desarrollan software de manera global. Particularmente, la herramienta estará enfocada a la gestión de decisiones de proyectos software en desarrollo global, intentando minimizar todo lo posible los problemas de comunicación y control y falta de reutilización de "lecciones aprendidas" que aparecen en esta nueva forma de desarrollo deslocalizado (ver Capítulo 2 para una revisión de la literatura acerca del Desarrollo Global, Gestión de Conocimiento, Gestión de Decisiones y *Design Rationale*).

1.2 Objetivos

El objetivo principal del PFC consistirá en el diseño y construcción de una **herramienta** basada en Java que permita dar soporte a la **gestión de decisiones en proyectos software** en el paradigma de **Desarrollo Global de Software**. Por tanto, dicha herramienta debe permitir la creación, almacenamiento, recuperación, transmisión y aplicación de decisiones abordadas en un proyecto software, realizado de manera deslocalizada. Además, debe permitirse gestionar también los proyectos software sobre los que se toman decisiones.

De este modo, se intenta reducir o eliminar algunas de las problemáticas que aparece en el desarrollo global, como es la falta de comunicación y control entre los equipos de desarrollo, así como la falta de reutilización del conocimiento adquirido al desarrollar proyectos previos.

A continuación, se enumeran los objetivos parciales que se deben cumplir para alcanzar el objetivo principal. Para ello, la herramienta deberá:

- Representar las decisiones tomadas en los proyectos software de una manera visual, clara e intuitiva, evitando malentendidos, puesto que aunque el inglés pueda utilizarse como medio de comunicación, con frecuencia no es la lengua nativa de todos los miembros de los equipos de desarrollo y esto da lugar a malentendidos. De este modo, se representan dichas decisiones de la misma manera para todos los equipos de desarrollo y empleados, facilitando su transmisión y evitando confusiones.

- Proveer formularios para crear y modificar decisiones, estandarizando el conocimiento almacenado y facilitando su posterior recuperación y comunicación.

- Crear un mecanismo de comunicación síncrona, es decir, que si un empleado se encuentra visualizando decisiones de un determinado proyecto en la aplicación y, en ese momento, otro empleado de otra localización diferente realiza un cambio sobre ese proyecto, dicho cambio debe ser notificado al primer empleado en tiempo real, actualizando su vista. Además, deberá crearse también una notificación o alerta de manera automática, favoreciendo de este modo la comunicación asíncrona.

- Adaptación a diferentes idiomas.

- Dar soporte a la gestión de proyectos software, proveyendo formularios para dar de alta a nuevos proyectos, así como para su posterior modificación. De este modo, se favorece una estructura común de proyectos y se facilita su control entre los diferentes equipos de desarrollo que trabajan en esos proyectos.

- Permitir aspectos de control de proyectos por parte de los jefes de proyecto, generando gráficos e informes.

- Desarrollar un mecanismo de reutilización de decisiones entre diferentes proyectos software, de modo que decisiones de proyectos ya pasados puedan aplicarse a nuevos proyectos, con características similares. Para ello, se utilizan técnicas de Inteligencia Artificial, como es el Razonamiento Basado en Casos.

1.2.1 Resumen de objetivos

A modo de síntesis, en la Tabla 1.1 se resumen los objetivos que el sistema a desarrollar debe cumplir para alcanzar su objetivo principal. En dicha tabla se muestran además los desafíos del Desarrollo Global de Software que se intentan mitigar con la compleción de cada uno de esos objetivos.

Destacar que esta herramienta es en cierto modo un sistema para la gestión de conocimiento, ya que como se detalla en el Capítulo 2, permite la creación, almacenamiento, recuperación, transmisión y aplicación de decisiones. En el caso concreto del presente PFC, dicho conocimiento son las decisiones y alternativas planteadas durante las fases del ciclo de vida de los proyectos software, siguiendo de este modo el enfoque de *Rationale* (ver capítulo 2).

Id. Objetivo	Objetivo	Desafío GSD
O1	Permitir el acceso al sistema desde diferentes localizaciones	Comunicación y Control
O2	Facilitar y favorecer la gestión de decisiones en los proyectos software, evitando malentendidos debidos a diferencias socio-culturales	Comunicación y Gestión Conocimiento
O3	Favorecer la representación y visualización de la información almacenada, de manera clara e intuitiva, evitando malas interpretaciones	Comunicación y Gestión Conocimiento
O4	Facilitar la comunicación entre los equipos de desarrollo, notificando posibles cambios al instante	Comunicación
O5	Adaptación a diferentes idiomas	Comunicación
O6	Facilitar la gestión de proyectos software, utilizando una estructura común para su creación, modificación, etc.	Comunicación y Control
O7	Favorecer aspectos de control de proyectos, tales como la situación actual del proyecto, desarrolladores que en ellos trabajan, decisiones tomadas, etc.	Control
O8	Facilitar la reutilización de información entre proyectos software, aconsejando decisiones de proyectos similares	Control y Gestión Conocimiento

Tabla 1.1: Objetivos a cumplir en el PFC

Para terminar, a continuación se resumen los términos de mayor relevancia y que con más frecuencia aparecen en el presente documento, para facilitar la lectura y comprensión de posteriores capítulos:

- **Empresa**: el término *empresa*, o *compañía*, en el contexto de este PFC hace referencia a una organización dedicada y/o involucrada en el desarrollo de productos software.
- **Proyecto**: el término *proyecto* hace referencia a un proyecto software, desarrollado por los empleados de una compañía y cuyo desarrollo originará un producto software.
- **Decisión**: este término aparece mencionado en numerosas ocasiones a lo largo del presente documento. Dicho término hace referencia a las diferentes decisiones que los ingenieros informáticos realizan sobre un proyecto software, en cualquiera de las fases de su ciclo de vida. Así, dichas decisiones podrían ser decisiones de diseño del proyecto software, decisiones realizadas en el análisis, decisiones acerca de qué tecnologías utilizar, etc.
- **Usuario**: este término se refiere a cualquier empleado de una compañía dedicada al desarrollo software, independientemente del rol que dicho empleado tenga dentro de la compañía, y que puede utilizar el sistema.
- **Notificación**: este término, nombrado en algunas ocasiones como *alerta*, hace referencia a un aviso que el sistema genera para informar a los usuarios de cambios producidos en los proyectos.
- **Vista**: este término hacer referencia a una parte de la interfaz gráfica de usuario que presenta elementos visuales con información que es relevante al usuario.

Capítulo 2

Estado del Arte

La forma de desarrollar software está evolucionando hacia el Desarrollo Global de Software, donde los proyectos software se desarrollan por diferentes equipos de desarrollo que se encuentran deslocalizados en diversos países. Este nuevo paradigma de desarrollo implica una serie de ventajas, pero también introduce una serie de desafíos, siendo uno de estos desafíos la Gestión de Conocimiento.

En Desarrollo Global, la información proviene de fuentes muy diversas, en diferentes formatos e incluso idiomas, lo que dificulta conocer las decisiones tomadas en proyectos software, su documentación y, en general, la gestión de conocimiento y su reutilización. Es por ello que existe una tendencia a usar el método *Rationale*, el cuál proporciona mecanismos para capturar y representar las decisiones tomadas en cualquier fase del ciclo de vida del desarrollo del proyecto software, así como todo el razonamiento que se hace para tomar esas decisiones. Por tanto, gracias a *Rationale*, se facilita la generación, almacenamiento y representación de decisiones y su razonamiento.

Además, una vez que este tipo de información ha sido capturada y almacenada, otro aspecto destacado a tener en cuenta, y que también aparece en la Gestión de Conocimiento, es la recuperación y aplicación de dicho conocimiento. Por tanto, es importante que las decisiones capturadas y almacenadas por el método *Rationale* en los proyectos, puedan recuperarse y reutilizarse en otros proyectos con características similares. Para ello, puede utilizarse una técnica de Inteligencia Artificial, como es el Razonamiento Basado en Casos, que puede proporcionar soluciones a distintos problemas en base a experiencias pasadas.

En la imagen 2.1 se refleja de manera visual cómo estos conceptos están relacionados entre sí, según se ha detallado anteriormente.

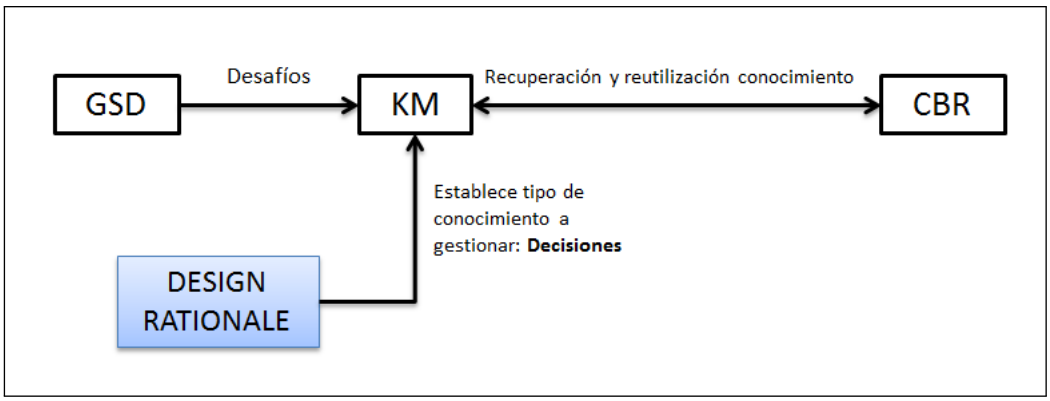

Figura 2.1: Relaciones entre Desarrollo Global, Gestión del Conocimiento, *Rationale* y Razonamiento Basado en Casos

De este modo, en las siguientes secciones se presenta el estado del arte sobre Desarrollo Global de Software, así como se detalla el método de *Rationale* y el Razonamiento Basado en Casos.

2.1 Desarrollo Global de Software

En los últimos años, la globalización económica está provocando que la industria y el comercio se adapten a nuevos modelos de negocios, en búsqueda de aumentar la competitividad y minimizar los costes, por lo que las empresas se han visto obligadas a evolucionar en su modo de trabajo. Así, se ha cambiado el concepto de *mercado nacional* por el de *mercado global* [11] [18].

Centrándose en el desarrollo de software, hasta hace unos años el desarrollo era llevado a cabo por Centros de Desarrollo Software (en adelante CDS) que estaban co-localizados en un mismo edificio. Sin embargo, debido a este nuevo modelo de economía global, la manera de desarrollar software está evolucionando en los últimos años, tendiendo a un desarrollo distribuido del mismo. Los CDS comenzaron a distribuirse en diferentes ciudades e incluso diferentes provincias de un mismo país, hasta llegar al punto en que los CDS se encuentran distribuidos en diferentes países y continentes. Esto es lo que se conoce como **Desarrollo Global de Software**, denominado a partir de este punto como GSD, por sus siglas en inglés Global Software Development. En la Tabla 2.1 se muestran los diferentes modelos de desarrollo de software existentes, en base a la dispersión geográfica.

EL GSD puede definirse como: "*el desarrollo de software que se realiza en localizaciones separadas geográficamente más allá de fronteras nacionales, de manera coordinada e involucrando participación en tiempo real (síncrona) e interacción asíncrona*" [17].

Localización	Tipo de desarrollo
Mismo lugar	Desarrollo tradicional (Co-localizado)
Mismo país	Desarrollo Distribuido de Software (DSD)
Distintos países	Desarrollo Global de Software (GSD)

Tabla 2.1: Modelos de desarrollo de software según dispersión geográfica

En este nuevo modelo de GSD, además de tener en cuenta que los *stakeholders*[1] se encuentran distribuidos geográficamente, hay que contar con el número de compañías que colaboran en un mismo proyecto. Un mismo proyecto bien puede ser desarrollado por una misma compañía, pero con CDS distribuidos en diferentes países, o bien puede llevarse a cabo por un conjunto de compañías que colaboran o están subcontratadas por la primera [28]. Así, surgen los conceptos de outsourcing, offshoring y nearshoring.

Por **outsourcing** se entiende la externalización o subcontratación de uno o varios servicios a terceros, ya que resulta más rentable, en términos de disminución de costes, que la compañía subcontratada realice estos servicios, ya que la mano de obra puedes ser más barata en un país diferente al país de la compañía contratante [27].

Relacionado con el *outsourcing*, está el concepto de **offshoring**, entendido como la relocalización de actividades de negocio, mediante empresas filiales de otros países [5]. Si estos países implicados en el proceso son relativamente cercanos al país de origen de la compañía que realiza el *outsourcing*, se habla de **nearshoring** [27].

En GSD, estas prácticas son cada vez más utilizadas, debido a la rentabilidad que proporcionan a la compañía. En la Figura 2.2 se puede observar el crecimiento del *offshoring* de los últimos años y la reducción de costes que conlleva.

Como se puede apreciar, el GSD y sus prácticas introducen una serie de beneficios o ventajas, pero también de dificultades o inconvenientes a tener en cuenta. En el siguiente apartado se desarrollan los más importantes.

[1] Stakeholders: *aquellos que pueden afectar o son afectados por las actividades de una empresa* [16].

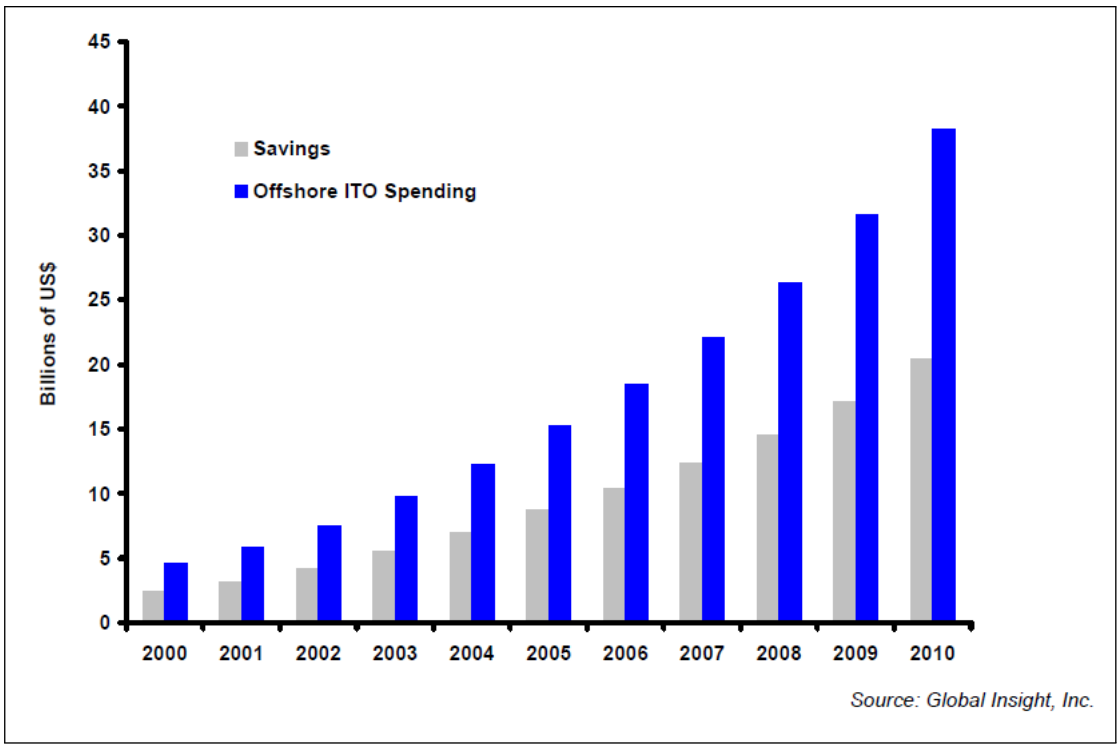

Figura 2.2: Incremento en actividades de *offshoring* y ahorro que conlleva [21]

2.1.1 Beneficios del Desarrollo Global de Software

A continuación se describen los beneficios más destacables del Desarrollo Global de Software, según [2] y [9].

2.1.1.1 Reducción de costes

La diferencia de salarios entre países puede ser enorme. Por ejemplo, el salario de un ingeniero de software en Estados Unidos puede llegar a ser varias veces más alto que el salario de un ingeniero similar en Asia o Sudamérica, por lo que las compañías buscan localizaciones alternativas que permitan recortar el coste de la mano de obra.

Gracias al GSD, las compañías tienen acceso a mano de obra más barata, debido a su deslocalización en diferentes países, pudiendo encontrar salarios mucho menores que en el país de origen. Ya que las compañías siempre buscan reducir el coste, éste ha sido el principal motivo que ha impulsado el GSD.

2.1.1.2 Aumento de la competitividad

El GSD provee la posibilidad de poder encontrar mano de obra más cualificada o más experimentada en diferentes países, de modo que las compañías pueden contratar esos recursos personales y expandir su desarrollo a zonas más cualificadas y competitivas.

2.1.1.3 Disminución de tiempo de lanzamiento al mercado

En GSD, se puede aplicar el modelo de desarrollo "*follow-the-sun*". Esto significa que, gracias a que existen varios centros de desarrollo en diferentes países, una compañía puede aprovechar las diferencias horarias entre sus diferentes centros de desarrollo para estar desarrollando software las 24 horas del día (ver Figura 2.3). De este modo, se consigue maximizar la productividad y disminuir el tiempo de salida al mercado (*time-to-market)* de un producto software.

De este modo, cada uno de los CDS localizados en cada país, pueden ir realizando diferentes actividades, como, por ejemplo, diseño, implementación, pruebas, etc. Para que este modelo de trabajo sea efectivo, debe existir una coordinación entre todos los CDS, que es una de las dificultades del GSD, como se comentará posteriormente.

Figura 2.3: Modelo de desarrollo *Follow-the-sun* [7]

2.1.1.4 Proximidad al mercado y al cliente

Al establecer filiales en aquellos países donde se localizan potenciales clientes, el GSD permite el desarrollo de software de una manera más cercana a sus clientes, aprovechando el conocimiento del mercado local y, por tanto, sus necesidades.

2.1.1.5 Mejoras en la innovación

Al disponer de CDS en diferentes países, se puede tomar ventaja de las diferentes nacionalidades, culturas, experiencias y habilidades de cada uno de ellos, pudiendo así evolucionar, innovar y enriquecer la compartición de conocimiento.

Como se ha visto, el GSD acarrea una serie de ventajas y beneficios que lo han convertido en un nuevo modelo de desarrollo utilizado por muchas empresas del sector. Sin embargo, existen también una serie de dificultades y problemas derivados de la deslocalización de los recursos, que se detallan en el siguiente apartado.

2.1.2 Desafíos del Desarrollo Global de Software

El GSD provee a las compañías y organizaciones unos beneficios muy prometedores. Sin embargo, existen dificultades a las que las compañías tienen que hacer frente, todas ellas debidas a la distancia y a la deslocalización (ver Figura 2.4) [9] [15].

Según [15], existen tres tipos de distancia que dificultan el GSD:

- **Distancia geográfica**. En GSD, se define como "*la medida de esfuerzo que un individuo necesita realizar para visitar otro punto, alejado del primero*". Por ejemplo, dos lugares dentro del mismo país con un enlace aéreo directo y vuelos regulares, se pueden considerar relativamente cercanos, aunque estén separados por grandes distancias kilométricas. Sin embargo, no se puede decir lo mismo de dos lugares que están cerca geográficamente (separación de pocos kilómetros) pero con poca infraestructura de transporte. Este último caso tendría una elevada distancia geográfica.

- **Distancia temporal**. En GSD, se define como "*la medida de la deslocalización en tiempo, experimentada por dos individuos que desean interactuar*". Esta distancia

normalmente va unida a la anterior, provocando que existan husos horarios diferentes entre dos puntos debidos a la distancia geográfica existente entre dichos puntos.

- **Distancia socio-cultural**. Se define como "*la medida en que un individuo comprende las costumbres (símbolos, normas y valores sociales) y cultura de otro individuo*". Aunque esta distancia es muy frecuente en el GSD, también aparece en equipos de desarrollo co-localizados, ya que cada miembro del equipo puede tener nacionalidades y culturas diferentes. Este tipo de distancia es el mayor desafío a superar en el GSD, ya que con mucha frecuencia se producen conflictos y malentendidos entre los diferentes equipos de desarrollo, debido al carácter multicultural y multinacional del GSD.

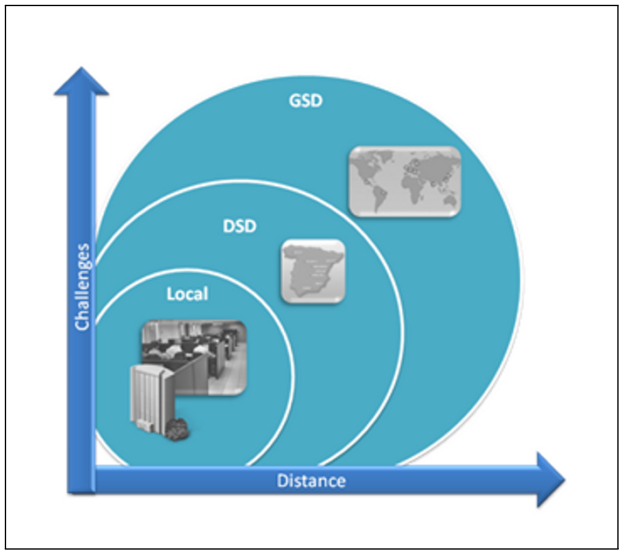

Figura 2.4: Influencia de la distancia en el desarrollo de software [15]

Estas distancias afectan a los tres grandes procesos que intervienen en el proceso de desarrollo de software: **comunicación**, **coordinación** y **control**, siendo los de comunicación y control los más problemáticos en GSD, según [2].

Por tanto, los desafíos y dificultades del GSD se agrupan en tres categorías, lo que se conoce como las tres "C" [10]:

- Desafíos en la ***Comunicación***, es decir, en el proceso de intercambio de conocimiento e información entre individuos.

- Desafíos en la ***Coordinación***, relacionados con objetivos e intereses comunes.

- Desafíos en el ***Control***, relacionados con la gestión del proyecto, informes, progreso, etc.

En la Tabla 2.2 se observa como afecta cada una de las distancias a los procesos de desarrollo software.

Proceso	**Distancia**		
	Geográfica	Temporal	Socio-Cultural
Comunicación	Dependiente de la tecnología	Uso de comunicación asíncrona	Malentendidos
Coordinación	Falta de conciencia de equipo	Modificación de calendarios laborales	Falta de confianza
Control	Incrementa la dificultad de gestión y control	Control asíncrono sobre rescursos remotos	Diferencias culturales en mecanismos de control

Tabla 2.2: Procesos de desarrollo software afectados por las distancias en GSD

En los siguientes apartados se resumen los desafíos más destacables:

2.1.2.1 Desafíos en la comunicación

Debido a la distancia geográfica, temporal y socio-cultural entre los distintos CDS, aparecen las siguientes dificultades en la comunicación [15] [9] [10].

2.1.2.1.1 Falta de comunicación

La posibilidad de un intercambio de información *face-to-face* (cara a cara, en español) desaparece o es prácticamente nula debido a la deslocalización de los equipos de desarrollo. Por ello, se depende casi en la totalidad de herramientas de comunicación, que son muy variadas y pueden no seguir estándares de comunicación, por lo que se provocan malentendidos.

2.1.2.1.2 Aumento del esfuerzo en la comunicación

Debido al uso de herramientas de comunicación, los tiempos de respuesta suelen ser altos y con frecuencia no se conoce personalmente a los miembros que intervienen en la comunicación ni su disponibilidad, por lo que la comunicación cada vez se vuelve más escasa y de menor calidad.

2.1.2.1.3 Interrupciones

Al encontrarse muchas veces los CDS en diferentes husos horarios, resulta muy complicado el poder contactar con individuos de otros equipos de trabajo y, en multitud de ocasiones, se produce la comunicación en horarios no adecuados o se producen interrupciones en el trabajo de un equipo, lo que aumenta el malestar en el trabajo.

2.1.2.1.4 Malentendidos debidos a diferencias lingüísticas

En GSD con frecuencia se producen malentendidos al comunicarse, debido a que los equipos de desarrollo hablan idiomas nativos diferentes. Estos fallos en la comunicación repercutirán posteriormente en el desarrollo, sobre todo si son malentendidos producidos durante la fase de análisis de requisitos del proyecto.

2.1.2.1.5 Malentendidos culturales

Al establecer una comunicación entre personas que no conocen totalmente las costumbres y cultura de la otra parte, puede provocar situaciones incómodas de manera involuntaria. Por ejemplo, en USA, prefieren especificar cada detalle en un documento, utilizar el teléfono y el e-mail de manera informal, mientras que en Japón prefieren el uso de medios más formales y prefieren la comunicación verbal a documentos escritos [9].

Por otro lado, en GSD el marco socio-cultural puede influir en opiniones diferentes sobre la propia naturaleza de llevar a cabo los procesos de desarrollo.

2.1.2.2 Desafíos en el control

Debido a la distancias existentes entre los distintos CDS, aparecen las siguientes dificultades en el control [15] [13].

2.1.2.2.1 Asignación de roles y estructuras de equipo

La deslocalización de los equipos de desarrollo complica y aumenta los costes en la asignación de roles y estructuras de equipos.

2.1.2.2.2 Gestión de los artefactos del proyecto

Cuando un proyecto involucra miembros de diferentes compañías o de diferentes CDS de la misma organización, hacer cumplir los procesos y normas del artefacto software es particularmente importante para mantener la consistencia e interoperabilidad entre los diferentes artefactos del proyecto.

2.1.2.2.3 Diferentes percepciones de la autoridad

La naturaleza de la autoridad dentro de un proyecto puede variar entre las diferentes culturas, por lo que pueden surgir problemas en el control de un proyecto si los equipos en diferentes lugares esperan que se maneje de manera diferente, y si estas expectativas diferentes no se identifican desde el principio.

2.1.2.3 Desafíos en la Gestión de Conocimiento

En la **Gestión del Conocimiento** también aparecen dificultades debidas a las distancias y desafíos presentes en el Desarrollo Global de Software, comentados anteriormente.

La gestión de conocimiento (en adelante KM, por sus siglas en inglés *Knowledge Management*) puede definirse como "*un proceso que permite crear, capturar, 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*" [6] [20] [22].

En los siguientes apartados se comentan los desafíos encontrados en cada una de los cuatro procesos que componen la Gestión de Conocimiento dentro de una empresa, mostrados en la Figura 2.5.

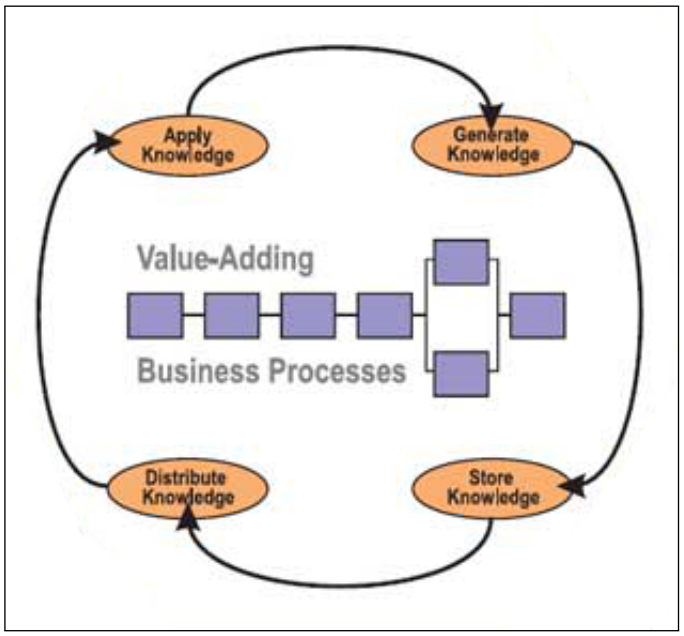

Figura 2.5: Ciclo de gestión de conocimiento [19]

2.1.2.3.1 Creación de conocimiento

La creación de conocimiento organizacional involucra crear nuevo conocimiento, o reemplazar ciertos contenidos con nuevo conocimiento tácito y explícito, obtenido tanto a través de colaboraciones e interacciones sociales, como a partir del propio proceso cognitivo de cada empleado [4].

En GSD, este proceso de creación de nuevo conocimiento se ve dificultado, ya que la información puede provenir de numerosas fuentes y lugares, con formatos y de tipos muy diferentes, en diferentes idiomas, etc. Además, las interacciones sociales se ven dificultadas por los desafíos de la comunicación, así como los procesos cognitivos de cada empleado, debido a las distancias socio-culturales. Por lo tanto, es difícil generar nueva información consistente y evitando duplicidades.

2.1.2.3.2 Almacenamiento y recuperación del conocimiento

Estrechamente ligado con el proceso anterior, se encuentra el almacenamiento y recuperación del conocimiento. Debido a que en GSD se manejan múltiples fuentes de conocimiento, y existen varias organizaciones que lo generan y utilizan, resulta complicado crear una base de conocimiento global y común para todas esas organizaciones que colaboran en GSD, debido a la multitud de información y formatos existentes (vídeos, documentos, procesos de negocio, etc.).

Para intentar mitigar los problemas que aparecen en estos procesos de KM, existe una tendencia a utilizar el enfoque de *Rationale*, o *Design Rationale*, para indicar qué tipo de información generar y almacenar, proveyendo así un tipo de información concreta, con un formato común y consistente que las empresas que colaboran en el desarrollo de proyectos software en GSD puedan crear, almacenar y recuperar, para su posterior aplicación. Este concepto de *Rationale* será presentado en la sección 2.2

2.1.2.3.3 Transmisión del conocimiento

El objetivo primordial de KM es que el conocimiento pueda ser utilizado y conocido por las empresas, para que puedan ser competitivas en el marco de globalización y sociedad de la información en el que nos encontramos.

La transmisión de conocimiento principalmente se produce por procesos de socialización, donde un grupo de individuos, normalmente con intereses comunes, comparten el conocimiento e intentan aprender y resolver problemas juntos [30].

En GSD, este proceso se ve dificultado debido a los desafíos que aparecen en la comunicación, ya que las herramientas de comunicación pueden no estar siempre disponibles y pueden impedir o dificultar esta transmisión de conocimiento. Además, debido a las diferencias socio-culturales, este proceso de compartición del conocimiento es más difícil de llevar a cabo, ya que dicho conocimiento puede ser interpretado de manera diferente en diferentes países, además de existir la barrera del idioma.

2.1.2.3.4 Aplicación del conocimiento

A la hora de aplicar el conocimiento recuperado, también existen dificultades, debido a todos los problemas que ya se han comentado, destacando la falta de consenso a la hora de aplicar dicho conocimiento, debido a las diferentes formas de gestionar y usar el conocimiento en las diversas organizaciones que colaboran en GSD, ya que cada una tendrá sus propios procesos de negocio y gestión.

2.2 *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 un proyecto software*" [12]. Dicho método puede ser aplicado en cualquier fase del ciclo de vida 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 del proyecto, y las razones de por qué son tomadas esas decisiones. Así, *Rationale* se centra en capturar [12]:

- Decisiones tomadas.
- Las razones detrás de la decisión tomada.
- Argumentos a favor o en contra de las decisiones tomadas.
- Decisiones evaluadas y su resultado.

Este enfoque de *Rationale* se plantea como una posible solución para mitigar los desafíos que aparecen en la gestión de conocimiento en GSD, comentados anteriormente. Gracias a esto, el conocimiento que en este caso interesa generar, almacenar, transmitir, reutilizar y aplicar son las decisiones tomadas en el desarrollo de un proyecto software. Por tanto, las diferentes organizaciones que participan en el desarrollo global de proyectos software utilizarán un mismo tipo de información, común y consistente.

De este modo, *Rationale* proporciona las siguientes ventajas a la hora de aplicarse en GSD, concretamente en la gestión de conocimiento:

- Proporciona un mecanismo común a las organizaciones para la captura y almacenamiento de las decisiones tomadas en el desarrollo de proyectos software, ya que se sigue el mismo formato y se define el tipo de información a generar.
- Facilita la representación de las decisiones y, por tanto, su transmisión, mejorando la calidad de futuras decisiones y la comunicación entre equipos de desarrollo.
- Facilita la recuperación de las decisiones, al estar bien definido el tipo de información que se ha almacenado. Del mismo modo, se facilita también su reutilización.

En lo que respecta a la captura de decisiones, en *Rationale* existen métodos muy variados. Sin embargo, algunos de los métodos que pueden emplearse para la captura de decisiones son [24] [8]:

- **Reconstrucción**: captura la información de una forma *cruda*, sin procesar, y luego la reconstruye de una manera más estructurada.
- **Método "Record and replay"**: las decisiones son capturadas de manera síncrona utilizando vídeo-conferencias, o de manera asíncrona, a través de discusiones por e-mail.
- **Método del "Aprendiz"**: se capturan las decisiones haciendo preguntas lógicas que coinciden, o no, con el punto de vista del diseñador.
- **Generación automática**: las decisiones se generan automáticamente a través de un historial. Este método tiene la capacidad para mantener las decisiones consistentes y actualizadas.
- **Método del "Historiador"**: es un método por el cual una persona u ordenador observa todas las acciones de los diseñadores, pero sin hacer ninguna sugerencia, y después describe todas esas acciones observadas.

En lo que respecta a representación del conocimiento, existen, principalmente, dos enfoques [12]:

- **"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 2.6 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 2.7 puede observarse un ejemplo. Este tipo de grafo se basa en los sistemas **IBIS** (**I**ssue-**B**ased **I**nformation **S**ystems, o Sistemas de Información Basados en Preguntas), donde el conocimiento se almacena y representa de manera jerárquica, en forma de decisiones/preguntas y justificaciones de éstas. Este será el método de representación utilizado en la aplicación a desarrollar en el presente PFC.

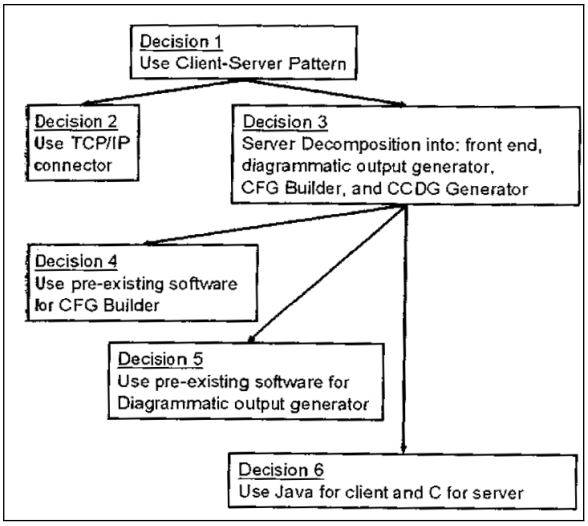

Figura 2.6: "Causal Graph" utilizado para representar decisiones en *Rationale* [1]

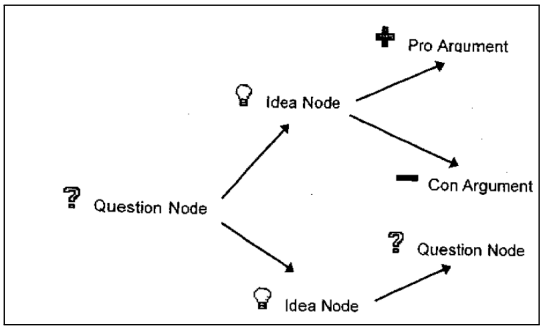

Figura 2.7: "Dialogue Map" utilizado para representar decisiones en *Rationale* [26]

2.3 Razonamiento basado en casos

Estrechamente relacionado con la gestión de conocimiento, está el concepto de **Razonamiento Basado en Casos** (en adelante CBR, por sus siglas en inglés Case-Based Reasoning), utilizado para la recuperación de conocimiento, o experiencias previas, y su aplicación y reutilización en nuevos casos. Por tanto, el CBR puede plantearse como una solución para mitigar los problemas de recuperación y aplicación del conocimiento en GSD, comentados con anterioridad. Además, en este caso, el proceso de CBR se simplifica ya que, gracias al uso de *Rationale*, el formato y tipo del conocimiento a recuperar y reutilizar es el mismo: decisiones tomadas en proyectos software.

El CBR puede definirse como "*un proceso en el que experiencias específicas son recuperadas, reutilizadas, revisadas y almacenadas para utilizarse en la solución de problemas similares*" [3]. 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 [23].

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 [29].

Conceptualmente, el CBR se describe como un ciclo con cuatro grandes etapas [25] [29]:

- **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.

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 en el CBR es la función de similitud que se va a utilizar para buscar casos 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.

Capítulo 3

Consecución de Objetivos

El objetivo principal del PFC era desarrollar una aplicación que facilite y permita la gestión de decisiones tomadas en proyectos software, así como la reutilización de las mismas para aprovechar la experiencia obtenida en proyectos previos, aplicada al ámbito del Desarrollo Global de Software.

Consideramos que este objetivo ha sido conseguido gracias al diseño y construcción de la aplicación, basada en Java, que ha sido detallada y comentada a lo largo del documento, especialmente en el capítulo **??**.

A continuación, se explica la consecución del resto de sub-objetivos planteados en el capítulo 1, y resumidos en la Tabla1.1.

3.1 O1: Acceso desde diferentes localizaciones

La aplicación desarrollada ha sido diseñada siguiendo una arquitectura cliente-servidor, donde el servidor centraliza la lógica de dominio y control del sistema, y el cliente presenta la interfaz gráfica de usuario, que realiza peticiones al servidor y muestra los resultados relevantes para el usuario del sistema. Por ello, y gracias a esta arquitectura definida, se puede utilizar la aplicación cliente de manera distribuida y, por tanto, desde diferente ubicaciones.

Además, la aplicación cuenta con una funcionalidad de *login*, permitiendo el acceso a los usuarios de la misma. Esto corresponde al grupo funcional **F1** (ver Tabla **??** en capítulo **??**).

3.2 O2: Facilitar y favorecer la gestión de decisiones

Para facilitar la gestión de decisiones tomadas en proyectos software, se han diseñado e implementado funcionalidades para la creación de decisiones, para su modificación, eliminación, etc. Por tanto, el sistema provee un mecanismo que facilita y favorece dicha gestión, además de la comunicación entre equipos de desarrollos, ya que al utilizar formularios y estructuras comunes a todos ellos, se minimizan los malentendidos y ambigüedades.

Por tanto, este objetivo queda cubierto con los requisitos funcionales que componen el grupo funcional **F2** (ver Tabla **??** en capítulo **??**).

3.3 O3: Favorecer la representación y visualización de la información almacenada

La aplicación obtenida como resultado del desarrollo del PFC permite la representación de las decisiones tomadas en proyectos software, además de otra información asociada, de una manera gráfica, visual e intuitiva, lo que facilita también la comunicación, ya que dicha información puede ser entendida de una manera rápida, visual y sin ambigüedades.

Por tanto, este objetivo queda cubierto con los requisitos funcionales que componen el grupo funcional **F3** (ver Tabla **??** en capítulo **??**).

3.4 O4: Facilitar la comunicación

Para facilitar la comunicación entre los usuarios de los equipos de desarrollo distribuidos, el sistema implementa mecanismos de comunicación síncrona y asíncrona.

En lo referente a la comunicación asíncrona, la aplicación genera automáticamente alertas cuando se ha producido cualquier cambio sobre las decisiones de un proyecto. Dichas alertas están disponibles para su lectura, en cualquier momento, para todos aquellos empleados que en ese proyecto trabajan. De este modo, este objetivo queda cubierto con los requisitos funcionales que componen el grupo funcional **F4** (ver Tabla **??** en capítulo **??**).

En cuanto a la comunicación síncrona, gracias a la arquitectura cliente-servidor y a que los clientes deben acceder al sistema, registrándose en el servidor, cuando el cliente realiza cualquier petición relacionada con la gestión de decisiones, el servidor se encarga de notificar al resto de clientes conectados este cambio, para que puedan reflejarlo visualmente en ese mismo instante, en tiempo real. Por tanto, este objetivo también queda cubierto.

3.5 O5: Adaptación al idioma local

Gracias a la creación de ficheros de recursos, escritos en varios idiomas, la aplicación soporta la internacionalización y puede adaptarse al idioma elegido por el usuario. Ésto corresponde al requisito funcional **F8** (ver Tabla **??** en capítulo **??**), cubriendo este objetivo.

3.6 O6: Facilitar la gestión de proyectos software

Para facilitar la gestión de proyectos software, se han diseñado e implementado funcionalidades para la creación y modificación de dichos proyectos, proveyendo formularios para realizar estas tareas, de modo que se siga una estructura común y se eviten errores. Además, la aplicación también permite asignar o modificar los usuarios que en dichos proyectos trabajan, favoreciendo el control de éstos.

Por tanto, este objetivo queda cubierto con los requisitos funcionales que componen el grupo **F5** (ver Tabla **??** en capítulo **??**).

3.7 O7: Favorecer aspectos de control de proyectos

En lo que respecta al control de proyectos, la aplicación desarrollada implementa funcionalidades que favorecen dicho control, como es la exportación a archivos XML de la información de los proyectos y sus decisiones; la generación de informes en formato PDF, y la generación de gráficos estadísticos. De este modo, los jefes de proyectos pueden llevar un control acerca de los proyectos y toda su información asociada, generando diferentes informes y gráficos de una manera sencilla y visual.

Por tanto, este objetivo queda cubierto con los requisitos funcionales que componen los grupos funcionales **F6**, **F7** y **F9** (ver Tabla **??** en capítulo **??**).

3.8 O8: Facilitar la reutilización de información

Para satisfacer este objetivo, propuesto por la necesidad de recuperar y reutilizar decisiones (y toda su información relacionada) de proyectos finalizados, en nuevos proyectos semejantes, se ha diseñado e implementado en la aplicación un mecanismo basado en técnicas de inteligencia artificial (CBR, en este caso) para poder comparar proyectos, recuperar y reutilizar decisiones de dichos proyectos similares.

Por tanto, este objetivo queda cubierto con el requisito funcional **F5.5** (ver Tabla **??** en capítulo **??**).

Capítulo 4

Conclusiones y Propuestas

En las secciones posteriores de este capítulo se desarrollan las conclusiones obtenidas tras el desarrollo del PFC, así como algunas propuestas para realizar en un futuro.

4.1 Conclusiones

Tras haber realizado y finalizado el desarrollo del PFC, se ha podido comprobar como gracias a utilizar una metodología de desarrollo iterativa e incremental, como es el PUD, el sistema ha ido aumentando en funcionalidad de manera progresiva, facilitando las tareas de análisis, diseño, implementación y pruebas de cada iteración, que daban como resultado un nuevo incremento en la funcionalidad de la aplicación.

Además, gracias a este carácter iterativo del PUD y a su división en fases, se ha podido resolver de una manera sencilla y rápida la incorporación de los nuevos requisitos que fueron detectados al comienzo de la fase de Elaboración, puesto que fueron identificados en iteraciones muy tempranas del ciclo de vida y, por tanto, no causaron un retraso ni impacto importante en el desarrollo del sistema.

En cuanto a las tecnologías y *frameworks* utilizados, Hibernate ha permitido gestionar las operaciones relacionadas con las bases de datos de una manera transparente, permitiendo cambiar el SGBD utilizado sin afectar de ningún modo al sistema desarrollado.

Por otra parte, la elección de RMI ha supuesto una serie de ventajas a la hora de realizar el sistema distribuido:

- Los objetos remotos pueden ser manejados como si fueran locales.

- Gracias a la utilización de interfaces para comunicar los subsistemas, éstos son totalmente independientes de su implementación, y se pueden extender con nuevo métodos de una manera sencilla.

- El servicio de registro de RMI, *rmiregistry*, permite fácilmente localizar e invocar los objetos remotos por su nombre.

- Al estar basado e integrado en Java, resulta sencillo y transparente al programador implementar el modelo de objetos distribuidos.

Como principal inconveniente de utilizar RMI, se puede destacar el problema de utilizarlo junto a Hibernate, ya que al serializar y transferir los objetos remotos, estas referencias no son las mismas que las referencias que mantiene Hibernate para gestionar su modelo de objetos, por lo que se encontraron problemas a la hora de, por ejemplo, modificar objetos en la base de datos cuando el cliente había realizado cambios en ellos. Sin embargo, este problema quedó solucionado clonando los objetos que fueron serializados por RMI y que deben modificarse en la base de datos, usando Hibernate.

Por otro lado, gracias a la arquitectura definida e implementada, y al uso de RMI, se ha conseguido independencia y extensibilidad en el sistema cliente-servidor. De este modo, la implementación del cliente podría cambiar sin verse afectado el servidor, y se podrían añadir nuevas funcionalidades al servidor sin verse afectado el funcionamiento actual de los clientes. Además, se podrían implementar nuevos subsistemas clientes y comunicarse con el servidor de manera sencilla, gracias a las interfaces y RMI, por lo que el sistema puede extenderse sin dificultades.

Para concluir, con el desarrollo de este PFC se han adquirido conocimientos acerca de los temas que en él se abordan, gracias a la revisión de la literatura existente, especialmente en la gestión de decisiones según *Rationale*, y a la comprensión de un método de Inteligencia Artificial, como es el CBR, pudiendo diseñar e implementar un algoritmo de CBR en el sistema. Además, también se ha adquirido o se ha profundizado en el conocimiento y manejo de las herramientas y tecnologías utilizadas para la construcción de la aplicación, destacando el uso de RMI, Hibernate, iText, JUNG, servicios Web de geoposicionamient y cómo diseñar y generar interfaces gráficas de usuario extensibles, flexibles y adaptables, utilizando para ello, además de en otros casos, el API de reflexión de Java.

4.2 Trabajo Futuro

Como trabajo futuro, se propone crear una aplicación Web aprovechando las funcionalidades del servidor actual, y sus interfaces. De este modo, el sistema contaría con una aplicación de escritorio, compuesta por los clientes actualmente implementados, y por una aplicación Web, todo ello gestionado por el servidor. Así, se puede extender su utilización por parte de los usuarios, ya que podría utilizarse desde la aplicación de escritorio y desde la Web.

Por otra parte, se propone extender la funcionalidad del sistema, incorporando un control de los proyectos software y sus decisiones más amplio, generando diferentes informes, tablas, gráficos, etc. Es por ello que se podría incorporar **BIRT**, que es un sistema de informes basado en Java (*Reporting System*, en inglés) con capacidad de generación de tablas dinámicas a partir de fuentes de datos, así como de gráficos e informes avanzados.

Otra consideración a tener en cuenta como trabajo futuro sería incorporar a la aplicación un soporte colaborativo, especialmente a la gestión de decisiones, para que los usuarios puedan interactuar en tiempo real, tengan conciencia de lo que otros usuarios están realizando en ese momento y puedan comunicarse en ese instante, añadiendo, por ejemplo, un chat.

Para terminar, se propone incorporar también una funcionalidad que permita exportar los datos de los proyectos y los empleados que en ellos trabajan desde definiciones de Microsoft Project, pudiendo importar esos datos en la aplicación, creando y almacenando nuevos proyectos con dichos datos.

Bibliografía

[1]

[2] Ågerfalk, P. *et al.*: *Benefits of global software development: the known and unknown*. Making Globally Distributed Software Development a Success Story, pages 1–9, 2008.

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

[4] Alavi, M. and D.E. Leidner: *Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues*. MIS quarterly, pages 107–136, 2001.

[5] Aspray, W. *et al.*: *Globalization and offshoring of software*. Report of the ACM Job Migration Task Force, Association for Computing Machinery, 2006.

[6] Bornemann, M. *et al.*: *An Illustrated Guide to Knowledge Management*, 2003.

[7] Carmel, E., J.A. Espinosa, and Y. Dubinsky: *" follow the sun" workflow in global software development*. Journal of Management Information Systems, 27(1):17–38, 2010.

[8] Chen, A. *et al.*: *Design history knowledge representation and its basic computer implementation*. In *Proceedings of the Design Theory and Methodology Conference, ASME*, pages 175–184, 1990.

[9] Cho, J.: *Globalization and global software development*. Issues in information systems, 8(2):287–290, 2007.

[10] Conchúir, E.Ó. *et al.*: *Global software development: where are the benefits?* Communications of the ACM, 52(8):127–131, 2009.

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

[12] Dutoit, A.H. *et al.*: *Rationale management in software engineering*. Springer, 2006.

[13] Evaristo, J.R. and R. Scudder: *Geographically distributed project teams: a dimensional analysis*. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, pages 11–pp. IEEE, 2000.

[14] Evaristo, R. and P.C. van Fenema: *A typology of project management: emergence and evolution of new forms*. International Journal of Project Management, 17(5):275–281, 1999.

[15] Fitzgerald, B. *et al.*: *Global software development: a multiple-case study of the realisation of the benefits*. 2010.

[16] Freeman, R.E.: *Strategic Management: A Stakeholder Approach*. Cambridge University Press, 2010, ISBN 9780521151740.

[17] Heijstek, Werner: *Global software development*. Leiden University, 2011.

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

[19] Holzner, B. and J.H. Marx: *Knowledge application: The knowledge system in society*. Allyn and Bacon Boston, 1979.

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

[21] Insight, G.: *Inc (2005)." executive summary: The comprehensive impact of offshore software and it services outsourcing on the us economy and the it industry."*. Information Technology Association of America.

[22] Kanagasabapathy, K.A., R. Radhakrishnan, and 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.

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

[24] Lee, J.: *Design rationale systems: understanding the issues*. IEEE Expert, 12(3):78–85, 1997.

[25] Lopez De Mantaras, R. *et al.*: *Retrieval, reuse, revision and retention in case-based reasoning*. The Knowledge Engineering Review, 20(03):215–240, 2005, ISSN 0269-8889.

[26] Rooksby, J., I. Sommerville, and M. Pidd: *A hybrid approach to upstream requirements: Ibis and cognitive mapping*. In *Rationale management in software engineering*, pages 137–153. Springer, 2006.

[27] Sahay, Sundeep, Brian Nicholson, and Srinivas Krishna: *Global IT Outsourcing: Software Development across Borders*. Cambridge University Press, November 2003.

[28] Sangwan, R. *et al.*: *Global software development handbook (auerbach series on applied software engineering series)*. 2006.

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

[30] Wenger, E.: *Communities of practice: Learning, meaning, and identity*. Cambridge Univ Pr, 1998.