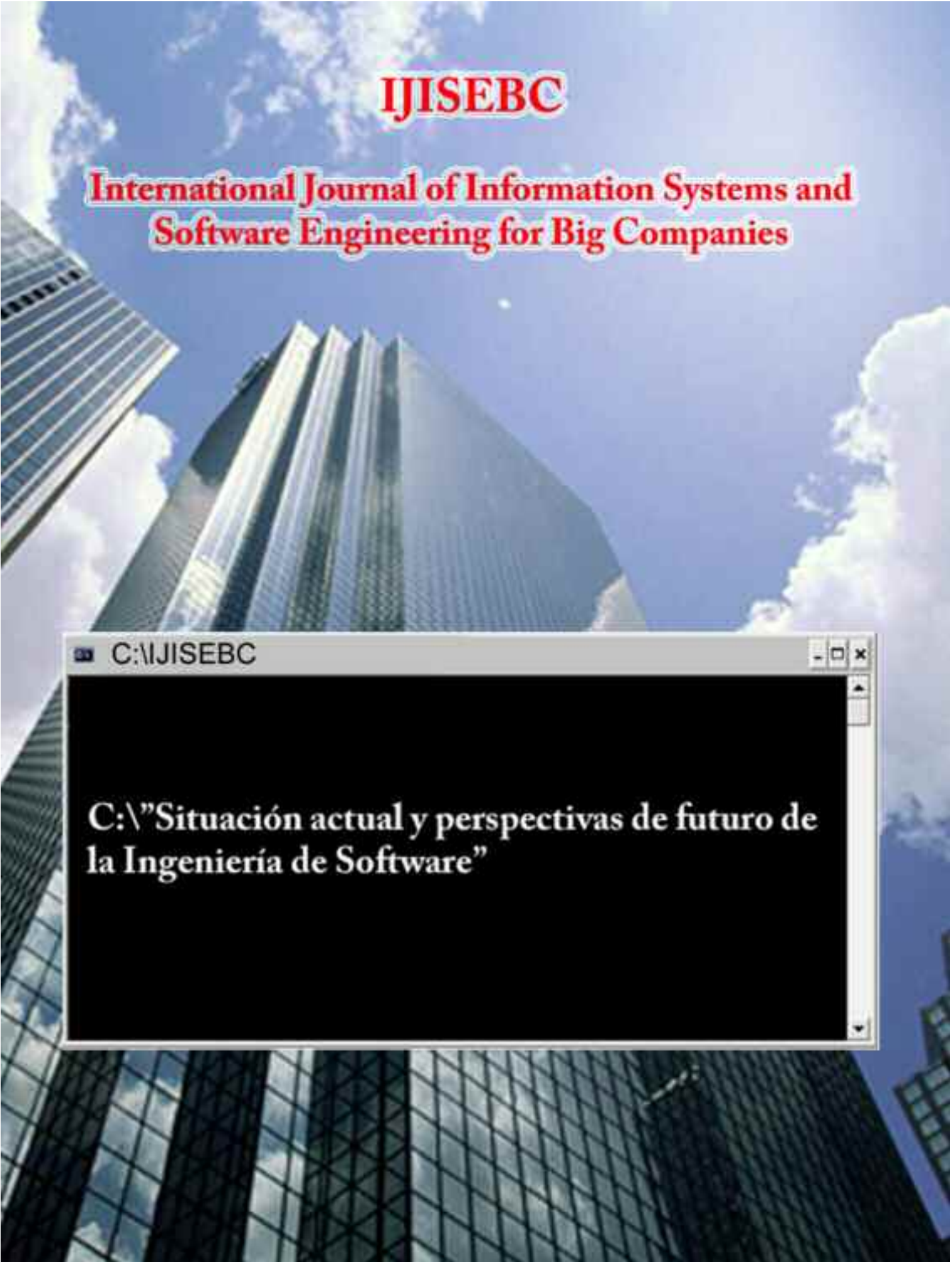
IJISEBC

International Journal of Information Systems and Software Engineering for Big Companies



International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)

Revista Internacional de Sistemas de Información e Ingeniería del Software para Grandes Corporaciones

© IJISEBC; VOL I; 01

INTERNATIONAL JOURNAL OF INFORMATION SYSTEMS AND SOFTWARE ENGINEERING FOR BIG COMPANIES

REVISTA INTERNACIONAL DE SISTEMAS DE INFORMACIÓN E INGENIERÍA DEL SOFTWARE PARA GRANDES CORPORACIONES

ISSN: 2387-0184
Huelva (Spain), vol. I; nº 01
2º semestre, diciembre de 2014

GITICE

Grupo de Investigación de las Tecnologías de la Información y las Comunicaciones en la Empresa

- Research Group of Information Technology and Communication in Business -

www.gitice.com

IJISEBC

INTERNATIONAL JOURNAL OF INFORMATION SYSTEMS AND SOFTWARE ENGINEERING FOR BIG COMPANIES

REVISTA INTERNACIONAL DE SISTEMAS DE INFORMACIÓN E INGENIERÍA DEL SOFTWARE PARA GRANDES CORPORACIONES

IJISEBC, 01, I, 2014

01, I

EDITORES (Editors)

Dr. Francisco José Martínez López
Universidad de Huelva, España

Dr. Alfonso Infante Moro
Universidad de Huelva, España

EDITOR ADJUNTO (Assistant Editor)

• D. Juan Carlos Infante Moro, Universidad de Huelva, España

COMITÉ EDITORIAL (Editorial Board)

• Dra. Mercedes García Ordaz, Universidad de Huelva, España
• D. Juan Carlos Infante Moro, Universidad de Huelva, España
• Dra. Rocío Arteaga Sánchez, Universidad de Huelva, España
• Anca Zavate, University Alexandru Ioan Cuza, Rumania

GESTIÓN COMERCIAL Y DISEÑO (Commercial Manager):
• Juan Carlos Infante Moro, Universidad de Huelva, España

COMITÉ CIENTÍFICO (Advisory Board)

• Dra. Mercedes García Ordaz, Universidad de Huelva, España
• Dra. Rocío Arteaga Sánchez, Universidad de Huelva, España
• Dr. Mario Piattini, Universidad de Castilla la Mancha, España
• Ph.D. David Garlan, University Carnegie Mellon, USA
• Dra. Paula Luna, Universidad de Sevilla, España
• D. Manuel Galán, Steelmood, España
• D. Fernando Ruiz-Falcó, Steelmood, España
• D. Alfonso Royo, Steelmood, España
• Dr. Andrea Carignani, IULM, Italia
• D. Antonio José Redondo García, Universidad de Huelva, España
• Ph.D. Antonio Padilla, Universidad de Málaga, España
• Ph.D. Aknin Noura, Abdelmalek Essaâdi University, Marruecos
• Dra. Ana Rosa del Águila Obra, Universidad de Málaga, España
• D. César Montes de Oca, Quarksoft, Mexico
• Ph.D. Carlos Sousa, Universidade do Algarve, Portugal
• Dr. Antonio Peregrín, Universidad de Huelva, España
• Dra. Lorena Pichardo Flores, Universidad Nacional Autónoma de Mexico, Mexico
• Dr. Luis Ignacio López, Universidad de Huelva, España
• Dra. Albetina Dias, Universidade Nova, Portugal
• Kamal EL KADIRI, Abdelmalek Essaâdi University, Marruecos
• Dr. Manuel Ángel Serrano Martín, Universidad de Castilla la Mancha, España

© ISSN: 2387-0184

S U M A R I O • C O N T E N T S

IJISEBC, 01, I, 2014

Situación actual y perspectivas de futuro de la Ingeniería de Software.

Current status and future prospects of Software Engineering.

PRELIMINARES (FOREWORD)

Sumario (Contents) 3/4
Editorial 5/6
Francisco J. Martínez y Alfonso Infante

COORDINADOR/COORDINATOR

Dr. Mario Piattini Velthuis

Universidad de Castilla la Mancha, España. Doctor y Licenciado en Informática por la UPM. Licenciado en Psicología por la UNED. CISA, CISM, CRISC y CGEIT por la ISACA. Auditor Jefe ISO 15504 por AENOR. Ha trabajado como consultor para numerosas organizaciones (MINER, MAP, Siemens-Nixdorf, Unisys, Hewlett-Packard, Oracle, ICM, Atos-Ods, Indra/Soluziona, STL, Alhambra-Eidos, etc.). Socio fundador de las empresas Cronos Ibérica, S.A. y Kybele Consulting, S.L. Ha sido Coordinador del Área de Ciencias de la Computación y Tecnología Informática de la ANEP y Director del Centro Mixto de I+D de Software UCLM-INDRA Software Labs. Catedrático de Universidad de Lenguajes y Sistemas Informáticos en la ESI de la UCLM, dirige el grupo de investigación Alarcos. Director del Instituto de Tecnologías y Sistemas de Información de la UCLM y Socio-Director de AQC, S.L.

ARTÍCULOS (PAPERS)

• Visión General del Desarrollo Global de Software 08/22
Overview of Global Software Development
Aurora Vizcaíno, Félix García y Mario Piattini. Ciudad Real (España)

• Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software . 24/36
Indra's technological proposal to tackle the immediate challenges of Software Engineering
Gabriel Sánchez, Carlos García y Yolanda Hernández. Madrid (España)

• Software Development by Steelmood 38/50
El Desarrollo de Software por Steelmood
Fernando Ruiz-Falcó. Madrid (España)

• On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE? . . 52/68
Sobre la adopción industrial del Model Driven Engineering (MDE) ¿Está su empresa preparada para MDE?
Antonio Vallecillo. Málaga (España)

• Software Engineering: Reflections on an Evolving Discipline 70/77
Ingeniería de Software: Reflexiones sobre una disciplina en evolución
David Garlan. Pittsburgh (USA)

• 25 Challenges of Semantic Process Modeling 78/94
25 Desafíos de la Modelación de Procesos Semánticos
Jan Mendling. Vienna (Austria). Henrik Leopold. Amsterdam (Netherlands). Fabian Pittke. Vienna (Austria).

© ISSN: 2387-0184

Sobre la revista (about magazine)

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC) conforma el instrumento de divulgación internacional de los trabajos de investigación e innovación relativos al uso de la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC), con el fin de recoger experiencias y estudios que involucren o influyan a las grandes empresas del tejido empresarial internacional y nacional. Esta publicación incorpora todos los indicadores y parámetros propios de las publicaciones de carácter científico de relevancia. Para ello, cuenta con un prestigioso Comité Científico que ejercen como evaluadores bajo el sistema de evaluación externa denominado "doble-ciego", lo cual asegura la calidad de las publicaciones.

Normas de publicación (Submission guidelines)

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» es una revista que provee el acceso libre e inmediato a su contenido bajo el principio de hacer disponible gratuitamente la investigación al público, lo cual fomenta un mayor intercambio de conocimiento global.
Se rige por las normas de publicación de la APA (American Psychological Association) para su indización en las principales bases de datos internacionales.
Cada número de la revista se edita en versión electrónica.

TEMÁTICA Y ALCANCE

Artículos científicos: Contribuciones científicas originales sobre la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC). Los artículos generalmente tienen una extensión entre 3.000 y 10.000 palabras y son revisados por el sistema de pares ciegos.

Reseñas bibliográficas: Se recogen textos descriptivos y críticos sobre una publicación de interés actual.

APORTACIONES

Los trabajos deben ser originales, sin haber sido publicados en ningún medio ni estar en proceso de publicación, siendo responsabilidad de los autores el cumplimiento de esta norma y deben tratar un tema actual y de interés público.

Los manuscritos se presentarán en tipo de letra arial, cuerpo 11, interlineado simple, justificados completos y sin tabuladores ni retornos de carros entre párrafos. Sólo se separarán con un retorno los grandes bloques (autor, títulos, resúmenes, descriptores, créditos y apartados). La configuración de página debe ser de 2 cm. en todos los márgenes (laterales y verticales). Los trabajos han de venir en formato .doc, .docx o .odt.

La extensión estará comprendida entre 3.000 y 10.000 palabaras.

Es importante que los manuscritos no contengan ninguna información que pueda dar a conocer la autoría.

EVALUACIÓN DE MANUSCRITOS

El Consejo de Evaluadores Externos de «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» es un órgano colegiado esencial para poder garantizar la excelencia de esta publicación científica, debido a que la revisión ciega basada exclusivamente en la calidad de los contenidos de los manuscritos y realizada por expertos de reconocido prestigio internacional en la materia es la mejor garantía y, sin duda, el mejor aval para el avance de la ciencia y para preservar una producción científica original y valiosa.

La evaluación de manuscritos por expertos internacionales, en consecuencia, es la clave fundamental para seleccionar los artículos de mayor impacto para la comunidad científica.

Esta revisión permite también que los autores, una vez que sus manuscritos son estimados para ser evaluados, puedan contar con informes objetivables sobre los puntos fuertes y débiles de sus manuscritos, en virtud de criterios externos.
Todas las revisiones en «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» emplean el sistema estandarizado internacionalmente de evaluación por pares con «doble ciego» que garantiza el anonimato de los manuscritos, auditados dentro de la Plataforma «OJS», Open Journal System, generándose un promedio de cinco informes por cada manuscrito

Normas de publicación / guidelines for authors (español-english) en: www.ijisebc.com

Grupo editor (Publishing Group)

GITICE (Grupo de Investigación de las Tecnologías de la Información y las Comunicaciones en la Empresa)
www.uhu.es/gitice

© ISSN: 2387-0184

IJISEBC, 01, I, 2014

Editorial

Editorial

Dr. Francisco José Martínez López
Dr. Alfonso Infante Moro
Editores

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC) (ISSN: 2387-0184) es una revista científica de investigación multidisciplinar en relación con el uso de la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC). Con una doble vocación, recoger las experiencias de investigadores a título personal y las experiencias institucionales.

Esta revista científica de ámbito internacional para la reflexión, la investigación y el análisis de la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC), se publicará en Español e Inglés.

Editada desde diciembre de 2014, se presenta como una revista con periodicidad semestral y con rigurosa puntualidad cada semestre, los meses de junio y diciembre. La revista cuenta con un consejo científico asesor formado por investigadores prestigiosos nacionales e internacionales en este ámbito, pertenecientes tanto a instituciones universitarias como a centros de investigación e instituciones superiores de América y Europa esencialmente.

GITICE

www.gitice.com

© ISSN: 2387-0184

Editorial

Editorial

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), como revista científica que cumple los parámetros internacionalmente reconocidos de las cabeceras de calidad, incluye en todos sus trabajos resúmenes y abstracts, así como palabras clave y keywords en español e inglés. Todos los trabajos, para ser publicados, requieren ser evaluados por expertos, miembros de los comités asesores y de redacción de la publicación y se someten a revisión de pares con sistema «ciego» (sin conocimiento del autor). Sólo cuando reciben el visto bueno de dos expertos los mismos son aprobados. En cada trabajo se recoge la fecha de recepción y aceptación de los mismos.

En sus diferentes secciones, en las que prevalece la investigación, se recogen monografías sobre temáticas específicas de este campo científico, así como experiencias, propuestas, reflexiones, plataformas, recensiones, informaciones para favorecer la discusión y el debate entre la comunidad científica y profesional de la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC). En sus páginas, los investigadores cuentan con un foro de reflexión crítica, con una alta cualificación científica, para reflexionar y recoger el estado de la cuestión en esta parcela científica, a fin de fomentar una mayor profesionalización del uso de las mismas.

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC) recepciona trabajos de la comunidad científica (universidades, centros de educación superior), así como de profesionales de la Ingeniería de Software y las Tecnologías de la Información y la Comunicación (TIC) en las Grandes Corporaciones de todo el mundo. La revista es editada por el Grupo de Investigación de las Tecnologías de la Información y las Comunicaciones en la Empresa (GITICE), formado por profesionales, docentes e investigadores universitarios de las universidades de Huelva, Sevilla, Granada, Autónoma de Madrid, así como por profesores internacionales, tanto en América como en Europa. Este Grupo de Investigación funciona desde 1993, interesado en promover las Tecnologías de la Información y las Comunicaciones en el mundo empresarial, siendo sus principales líneas de investigación: el Análisis de los Sistemas de Información Empresariales (TPS, DSS, ERP,...) e Interempresariales, el E-Comercio, la E-Administración, los Nuevos Modelos de Internet, la WEB 2.0, 3.0 y 4.0, la transmisión del olor por Internet, el Intercambio Electrónico de Documentos, la robótica aplicada a la Dirección de Empresas, el comportamiento del consumidor en Internet, Internet y Turismo, Contabilidad Digital y Teletrabajo.

© ISSN: 2387-0184

© ISSN: 2387-0184

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)

IJISEBC, 01, I, 2014

Visión General del Desarrollo Global de Software

Overview of Global Software Development

Aurora Vizcaíno[1], Félix García[1], Mario Piattini[1]

[1] Instituto de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha, España

Aurora.Vizcaino@uclm.es, Felix.Garcia@uclm.es, Mario.Piattini@uclm.es

RESUMEN. Este artículo presenta una panorámica general del estado del arte y de la práctica del Desarrollo Global de Software (DGS), analizando las principales revisiones sistemáticas de la literatura e identificando un conjunto de áreas de gran interés en la actualidad.
El cual muestra que el DGS es un campo que empieza a alcanzar cierta madurez: cuya evolución ya no se encuentra limitada por factores críticos como las diferencias lingüísticas y culturales, sino que ésta depende más de factores como la motivación personal y las habilidades de los recursos humanos, y de la disponibilidad de funciones y responsabilidades bien definidas; y, al mismo tiempo, presenta nuevos desafíos centrados en importantes líneas de interés como: los Procesos para desarrollo y gestión, la Gestión de Proyectos DGS y los Equipos de Trabajo.

ABSTRACT. This paper presents an overview of the state of the art and the practical of Global Software Development (DGS), analyzing the main systematic reviews of the literature and identifying a set of areas of great interest today.
Which shows that the DGS is a field that begins to reach a certain maturity: whose evolution is no longer limited by critical factors such as language and cultural differences, but it depends more on factors such as personal motivation and skills of resources human, and the availability of clearly defined roles and responsibilities; and at the same time, presents new challenges focused on important areas of interest include: Processes for development and management, DGS Project Management and Task Forces.

PALABRAS CLAVE: Desarrollo Global de Software, DGS, Revisión sistemática, Estado del arte, Estados de la Práctica, Trabajos futuros.

KEYWORDS: Global Software Development, DGS, Systematic review, State of the Art, State of the Practical, Future studies.

Vizcaíno, A., García, F., y Piattini, M. (2014). Visión General del Desarrollo Global de Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 8-22. Consultado el [dd/mm/aaaa] en www.ijisebc.com

1. Introducción

El Desarrollo Global de Software (DGS) se ha consolidado como uno de los aspectos más relevantes en la investigación y práctica de la Ingeniería del Software en la década de 2010 como ya lo predijera (Boehm, 2006). Surge hace más de 20 años con las primeras prácticas de outsourcing, pero se oficializa como tal en 2006 con la celebración de la primera conferencia internacional sobre desarrollo global ICGSE (IEEE International Conference on Global Software Engineering).

En general, el DGS supone para las empresas una manera de disminuir costes de desarrollo intentando mantener el nivel de calidad (Audy et al., 2004). En particular, este tipo de desarrollo permite obtener los siguientes beneficios:

- Contar con profesionales a lo largo y ancho del mundo sin necesidad de afrontar el costo de traslado de esas personas (Kobitzsch et al., 2001). Como resultado se puede por ejemplo reducir el coste de contratación de desarrolladores de software cuyos salarios son más reducidos en ciertos países, mediante la subcontratación de una empresa o la creación de filiales de la misma empresa en otros países (Damian y Moitra, 2006).
- Producir software para clientes remotos sin necesidad de trasladar el equipo de desarrolladores, incrementando de esta forma las posibilidades de introducirse en nuevos mercados (Richardson et al., 2005; Damian y Moitra, 2006).
- Lograr jornadas de trabajo más extensas, y por ende mayor productividad, cuando los programadores se encuentran distribuidos en sitios con amplia diferencia horaria (Carmel y Agarwal, 2001; Ebert y Neve, 2001; Herbsleb y Moitra, 2001).
- Obtener ventajas de la diversidad de experiencias, conocimiento técnico y destrezas de los stakeholders distribuidos (Ebert y Neve, 2001; Richardson et al., 2005).

Existen otros factores que hacen interesante este modelo de desarrollo, como es el aumento de la innovación que nace de la diversidad cultural y de compartir experiencias entre sus miembros.

Como contrapartida el DGS ha producido un profundo impacto en la manera en que los productos software se conciben, diseñan, construyen, prueban y entregan a los clientes (Herbsleb y Moitra, 2001), para lo que se necesitan nuevos métodos de trabajo (Damian et al., 2004). El DGS también presenta una serie de problemas, según (Conchúir, 2010) los principales desafíos encontrados en DGS se pueden agrupar en las llamadas "3C's": desafíos en la comunicación, entendiendo el concepto de la comunicación como el intercambio de conocimientos e información; desafíos en la coordinación, relacionados con la realización de tareas para alcanzar objetivos e intereses comunes y; desafíos en el control, que se centran en la gestión del proyecto (cumplir calendarios de entregas, presupuestos, calidad, estándares, etc.). Dichos desafíos se acentúan debido a lo que se ha llamado en la literatura las tres distancias (Ågerfalk et al., 2005):

- Distancia geográfica, definida como "la medida de esfuerzo que un individuo necesita realizar para visitar otro punto, alejado del primero" (Conchúir, 2010). 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, definida como "la medida de la deslocalización en tiempo, experimentada por dos individuos que desean interactuar" (Conchúir, 2010). Esta distancia normalmente va unida a la anterior, ya que cuando existe distancia geográfica con frecuencia implica diferentes husos horarios, lo cual puede limitar o incluso impedir la comunicación síncrona porque no haya solape de horario entre dos equipos de trabajo que estén geográficamente distribuidos.
- Distancia socio-cultural definida como "la medida en que un individuo comprende las costumbres (símbolos, normas y valores sociales) y cultura de otro individuo" (Conchúir, 2010). Esta dis-

tancia aparece frecuentemente en DGS, ya que cada miembro del equipo puede tener una nacionalidad y cultura diferente. Este tipo de distancia puede provocar conflictos y malentendidos entre los diferentes miembros de los equipos de desarrollo y suele ser también la causa de retrasos en las entregas de productos, es por ello que es uno de los temas que con más frecuencia se trata en la literatura (Huang, 2007).

Como consecuencia de lo anterior en los estudios sobre DGS se reportan numerosos problemas, entre los que se encuentran la falta de solidez teórica (Betz, 2010), el desconocimiento de los riesgos que este tipo de desarrollos implican (Betz, 2010), el uso de los mismos métodos, procesos y herramientas usadas en desarrollos tradicionales (da Silva et al., 2011) y requisitos incompletos o pobremente especificados (Islam et al., 2009). Otros problemas que las empresas han encontrado en su aplicación son (Eskeli y Maurolagoitia, 2011): la curva de aprendizaje, de modo que las personas que no están familiarizadas con las nuevas tecnologías DGS se resisten al aprendizaje; la pobre interoperabilidad entre herramientas; y los roles y responsabilidades no definidos claramente, entre otros.

En este artículo se presenta una panorámica del estado del arte sobre DGS, analizando los avances realizados hasta la fecha e identificando los desafíos de futuro. Para ello se analizan las principales revisiones sistemáticas de la bibliografía y se resumen algunas de las líneas principales de interés. El artículo se estructura del siguiente modo: en el apartado 2 se presentan las principales revisiones sistemáticas sobre DGS. En el apartado 3 se analizan las principales áreas de interés identificadas a partir de las revisiones sistemáticas y de la experiencia práctica de los autores en proyectos con industria. Finalmente se presentan las principales conclusiones obtenidas.

2. Revisiones Sistemáticas sobre DGS

Debido a la actual tendencia hacia la globalización del desarrollo de software, se ha incrementado tanto el número de revisiones sistemáticas (SLRs: Systematic Literature Reviews) como de estudios de mapeo (Mapping Studies) que se consideran una manera de sintetizar la investigación existente de un modo riguroso e imparcial (Genero et al., 2014). En la tabla 1 se presenta el conjunto de revisiones sistemáticas que se han publicado desde 2008. En el caso de existir varias SLRs de los mismos autores que suponen una ampliación de la original, se ha considerado la más actual.

Tema	Subtema	Referencias
Investigación general sobre DGS	Análisis de estudios existentes sobre DGS	(Niknasser et al., 2009; Alzoubaei y Durvedi, 2010; Kroll et al., 2011; Verner et al., 2012; Raza et al., 2013)
	Estudio empírico en DGS	(Smite et al., 2008; Treude et al., 2009; Šmite y Wohlin, 2011; Himanyan et al., 2013)
	Soluciones en entornos DGS	(Schneider et al., 2013)
	Ontología sobre DGS	(Borges et al., 2013)
Arquitectura	Gestión de conocimiento: identificación del conjunto de entidades DDS y sus relaciones	(Noor, 2010)

Proceso de desarrollo	Scrum, factores de riesgo y estrategias	(Hossain et al., 2009; Almeida et al., 2011; Hossain et al., 2011)
	Cadena de búsqueda o snowballing como primer paso en el desarrollo del proceso software	(Jalali y Wohlin, Systematic literature studies: Database searches vs. backward snowballing, 2012)
	Casos prácticos sobre el uso de metodologías ágiles	(Jurado et al., 2011; Šmite, Šmitenga y Šmite, 2011; Jalali y Wohlin, "Global Software Engineering and Agile Practices: A Systematic Review," 2012)
	Gestión de Riesgos en DDS	(Jiménez Mosnaor y Piattini, 2009; Persson, 2010; Šmite y Wohlin, 2011)
	Modelos y Procesos	(Jiménez Mosnaor y Piattini, 2009; Prikladnicki y Audy, 2010; Rocha et al., 2011)
Gestión de proyectos	Herramientas para dar soporte al DGS	(Costa et al., 2010; Cataldo y Herbsleb, 2011; da Silva et al., 2011; Chauhan y Ali Babar, 2012; Portillo-Rodríguez et al., 2012)
	¿Qué factores hay que tener en cuenta para gestionar un proyecto software en DGS?	(Cataldo y Herbsleb, 2011)(53)
	Barreras en gestión de proyectos DGS	(Oliani et al., Challenges of project management in Global Software Development: Initial results, 2013)
	Estimación	(Britto et al., 2014)
	Gestión de riesgos en DGS	(Virtani et al., 2014)
Ingeniería de requisitos	¿Cuáles son los principales riesgos del análisis de requisitos en DGS? Modelos, técnicas y herramientas que dan soporte	(Ebling et al., 2009)
	Riesgos y desafíos para el análisis de requisitos en contextos de DGS. ¿Qué propuestas existen para solventarlas?	(Lopez et al., 2009; Khan et al., Situational factors affecting Requirement Engineering process in Global Software Development, 2011)
Gestión de la configuración	Problemas, soluciones y estudios realizados sobre la gestión de configuración.	(Faun et al., 2010; Costa y Murta, 2013)
Outsourcing	Selección de Proveedores: Barreras y factores que influyen.	(Prikladnicki et al., 2008; Khan et al., 2009; 2011)
	Factores que pueden ayudar a proveedores en la gestión exitosa de las diferentes actividades durante las etapas de un contrato	(Khan y Khan, 2014)
	Factores de éxito en colaboraciones	(Ali y Khan, 2014)

Colaboración, comunicación, control y distancias	Barreras, Riesgos y Factores que influyen	(Jiménez Moreno y Piattini, 2009; Khan et al., "Communication Risks and Best Practices in Global Software Development during Requirements Change Management: A Systematic Literature Review Protocol," 2013)
	Distancias culturales, geográficas y temporales. Control y coordinación.	(Noll et al., 2010; Conradi et al., 2012)
	Barreras que afectan a la colaboración entre equipos distribuidos. Soluciones para solventar estas barreras	(Nurdiani et al., 2011)
	Colaboración en el desarrollo software	(Rocha et al., 2011)
	Estudios sobre *awareness* realizados con el objetivo de dar apoyo al DGS.	(Stevenaarbe et al., 2010; Smite y Wohlin, 2011; Vivian et al., 2011)
	Follow the sun	(Kroll et al., 2013)
	Evidencia empírica sobre el impacto de la dispersión en coordinación, rendimiento de equipo y de proyecto.	(Nguyen-Duc et al., 2014)
	Perspectiva "social" en DGS	(Gutfraind y Dittrich, 2013; Niazi et al., Motivators of Adopting Social Computing in Global Software Development: Initial Results, 2013)
Enseñanza	Técnicas y casos prácticos que se han realizado para tratar de impartir docencia en el contexto del DGS	(Monasor et al., Preparing students and engineers for Global Software Development: A Systematic Review, 2010; Budgen et al., 2012)

Tabla 1. Categorías y subtemas de las revisiones sistemáticas sobre DGS.

Tal como se puede observar en la Tabla 1, las revisiones sistemáticas sobre DGS se han enfocado principalmente en las distintas problemáticas del DGS y en cómo resolverlas o disminuir su impacto. Además, un menor número de revisiones se enfocan en un tema importante cómo la tecnología puede dar soporte a las distintas actividades del DGS. También, se deduce una clara tendencia a la utilización de metodologías ágiles, que pueden suponer una mejora a determinados problemas. Por ello, se deduce que existe cierta madurez principalmente en la investigación sobre los factores que influyen negativamente en el DGS y en las propuestas de soluciones para evitarlos o disminuirlos. Sin embargo, todavía es necesaria más investigación en temas relacionados con las herramientas de apoyo a este tipo de desarrollo y en la validación empírica sobre cómo las metodologías ágiles benefician el DGS comparado con otras metodologías. La descripción de casos reales en empresas son poco frecuentes pero muy necesarios ya que las lecciones aprendidas en estos casos son más relevantes que las obtenidas por experimentos o casos de estudios realizados con alumnos.

Con todo ello, en el siguiente apartado se resume un conjunto de líneas representativas de interés en DGS.

3. Síntesis del Estado del Arte y de la Práctica sobre DGS

En este apartado se sintetizan los resultados sobre las principales áreas de interés en DGS a partir del análisis de las revisiones sistemáticas y de la bibliografía relacionada con cada área.

3.1. Procesos ágiles para gestión y desarrollo global de software

Una de las áreas que más interés suscita en DGS en los últimos años es la definición de procesos adecuados en entornos DGS. La tendencia se centra la búsqueda de una exitosa combinación de métodos ágiles y desarrollo de software distribuido, tal como se presenta en diversas propuestas (Kussmaul et al., 2004; Ambler, 2009; Batra, 2009; Lee y Yong, 2009).

En la revisión sistemática realizada por (Hossain et al., 2009), que abarca los artículos que tratan la aplicación de Scrum en DGS desde 2003 a 2008, se concluye que hay un creciente interés en la realización de estudios empíricos para validar Scrum en la práctica del DGS pero se deben considerar una serie de retos y posibles limitaciones, como la necesidad de considerar los factores del proyecto (complejidad, presupuesto, etc..) a la hora de extraer conclusiones, ya que los resultados de la aplicación de Scrum pueden estar condicionados por dichos factores; los retos a los que se enfrentan los equipos Scrum y la necesidad de soporte de herramientas para facilitarles la aplicación de prácticas de Scrum en un entorno global; la necesidad de extender o modificar las prácticas de Scrum para dar soporte a DGS. En la revisión sistemática realizada por (Jalali y Wohlin, "Global Software Engineering and Agile Practices: A Systematic Review," 2012) se establece como las prácticas ágiles más aplicadas en DGS son las reuniones diarias de Scrum, así como el desarrollo iterativo o en forma de sprints, seguido por la integración continua, la planificación de sprints y las reuniones de retrospectiva. En cuanto a la combinación de métodos de desarrollo y de gestión para DGS se observa un mayor número de estudios que reportan: XP-Equipos distribuidos, Ágil-Offshore, Scrum-Equipo distribuido y Scrum-Offshore. También destacan la necesidad de mayor colaboración academia-industria al identificarse distintas percepciones de ambos en los estudios reportados.

En estos últimos años se ha avanzado en algunos de los desafíos comentados anteriormente, tanto en el desarrollo de herramientas web de soporte a Scrum, como en propuestas de solución de las limitaciones. Por ejemplo (del Nuevo et al., 2011), proponen la metodología "Scrum4D" cuyo objetivo es proporcionar guías para la gestión y desarrollo de software en un entorno distribuido utilizando las ventajas proporcionadas por la integración de métodos ágiles como Scrum y metodologías tradicionales como el Proceso Unificado de Desarrollo. Más recientemente, la tendencia se centra en la optimización de sprints en base al valor de las historias de usuario en proyectos DGS (Sobiech et al., 2014), la adaptación de las tareas del propietario del producto y del Scrum Master cuando trabaja a grandes empresas en proyectos a gran escala (Bass, 2013; 2014); y la gestión de cadenas de equipos scrum co-dependientes (Vlietland y Van Vliet, 2014).

3.2. Congruencia Socio-Técnica

Los problemas de coordinación y comunicación en DGS son, probablemente, uno de sus mayores desafíos. Con el fin de controlar, o al menos poder medir, estas dos variables surge lo que en inglés se llama "Socio Technical Congruence" (STC) que podemos traducir como congruencia socio-técnica. La idea principal del STC proviene de la ley de Conway que afirma que la estructura de un producto software refleja la estructura física de la organización. La STC se suele definir como la comparación entre el esfuerzo de coordinación requerido en un determinado proyecto de desarrollo de software con la coordinación que realmente se está llevando a cabo. Actualmente las empresas intentan conseguir un buen nivel de congruencia entre los requisitos de coordinación y las actividades de coordinación que actualmente se realizan. Teóricamente, un alto nivel de congruencia implicaría una mejora en la productividad y en la calidad del software. Sin embargo, también puede traer asociado un incremento de riesgo y coste. Además, un elevado número de interacciones puede provocar una sobrecarga de trabajo. Por lo tanto es conveniente que las empresas traten de encontrar el equilibrio entre el número de interacciones y el tiempo que se requiere para realizarlas.

Existen estudios empíricos que demuestran los beneficios de medir la STC. Concretamente en (Cataldo et al., 2006) se indica que controlando la congruencia se puede reducir el tiempo destinado a realizar determinadas tareas, como por ejemplo las modificaciones. Además, midiendo la STC se pueden detectar la falta de comunicación y evitar los efectos negativos que esto puede conllevar, por ejemplo, se ha comprobado que cuando hay falta de comunicación se incrementa el número de cambios que hay que realizar en el código (Ehrlich et al., 2008). Por su parte, los jefes de proyecto pueden beneficiarse del conocimiento de esta medida para controlar distintos aspectos, por ejemplo, comprobar el alineamiento entre la coordinación que existe en su equipo con las dependencias técnicas (Kwan et al., 2009). También, puede utilizarse para realizar un ranking de las tareas de coordinación que han faltado, analizando cual es la más importante y prioritaria de solucionar (Kwan et al., 2009).

Para medir la congruencia se requiere obtener información sobre: la estructura del equipo, las interacciones de comunicación, los procesos y prácticas de trabajo, la coordinación que se ha producido bien personalmente o a través de herramientas, el conocimiento tácito, la asignación de tareas y la localización de las personas, entre otros aspectos (Sarma et al., 2008). Actualmente, existen herramientas que ayudan a obtener parte de esta información, por ejemplo los foros, la mensajería instantánea, los gestores de correo o los repositorios de software. Otra información deberá obtenerse mediante encuestas o entrevistas. Las técnicas para medir la congruencia socio-técnica suelen comparar los requisitos de coordinación con la coordinación que realmente se está dando en el proyecto. Esta comparación se suele realizar usando matrices o redes sociales. En el caso de (Cataldo et al., 2006; Cataldo et al., 2008) se calcula comparando la que se llama Matriz de Requisitos de Coordinación (Cr) con la Matriz de Coordinación Real (Ca). La primera es una matriz persona a persona en la que cada celda representa qué debe coordinar un trabajador con otro. La Ca es calculada multiplicando la Matriz de Dependencia de Tareas (Td) en la cual cada celda representa la dependencia entre dos tareas y la Matriz de Tareas Asignadas (Ta) en la que cada celda representa el hecho de que un trabajador es asignado a una determinada tarea.

(Kwan et al., 2009; Kwan et al., 2011) se basan en el enfoque de Cataldo pero añaden un peso en las celdas. Ambos enfoquen ayudan a detectar falta de interacción entre dos personas que debe coordinarse.

Otro enfoque es el de (Ehrlich et al., 2008), en este caso se centran en detectar falta de comunicación en lugar de falta de coordinación. Para calcular la congruencia utilizan información de la comunicación obtenida de las redes sociales y la traducen a un grafo.

En (Portillo-Rodríguez et al., 2014) se presenta una arquitectura completa multiagente diseñada para medir y gestionar la STC con el objetivo de mejorar la coordinación y comunicación en DGS. Para ello se utiliza una arquitectura de agentes encargada de detectar de forma autónoma problemas de coordinación, calcular su importancia y notificar al jefe de proyecto sobre los problemas detectados y su importancia.

3.3. Estimación de Proyectos DGS

La estimación de proyectos software constituye un aspecto fundamental dentro de la gestión de proyectos ya que permite ir desde el inicio del ciclo de vida tomando decisiones más adecuadas en cuanto a la distribución de recursos a lo largo del proyecto. Los métodos de estimación paramétricos tradicionales que más se han aplicado en el campo de estimación del desarrollo y mantenimiento de software son Puntos Función (estimación de tamaño) y COCOMO II (estimación de tamaño y esfuerzo). Estos métodos, junto con otras técnicas de estimación tradicionales basadas juicio de expertos o ágiles como Planning Poker se siguen aplicando en proyectos DGS (Peixoto et al., 2010). De aquí se deriva la necesidad desarrollar métodos de estimación específicos para DGS, dada la heterogeneidad de métodos usados en proyectos DGS, lo que dificulta la comparación entre los resultados de estimación.

Sin embargo, estas técnicas tradicionales no se pueden aplicar tal cual para realizar estimaciones en Desarrollo Global de Software, ya que surgen nuevos retos que se deben resolver, en especial relacionados con la forma de distribuir el trabajo entre los distintas localizaciones (Lamersdorf et al., 2009) o nodos, así como nuevos factores de complejidad que pueden surgir. También es importante reutilizar el conocimiento de asignación de tareas en proyectos anteriores de desarrollo global.

En esta línea de interés, es importante considerar como factor clave la necesidad en DGS de realizar una asignación de trabajo sistemática entre las distintas localizaciones, que tenga en cuenta el coste de dicha asignación para seleccionar las mejores alternativas. A partir de lo anterior, (Lamersdorf et al., Estimating the Effort Overhead in Global Software Development, 2010; Lamersdorf et al., A Rule-Based Model for Customized Risk Identification in Distributed Software Development Projects, 2010) proponen: un modelo de asignación de trabajo en base a las características de los proyectos, sus objetivos y recursos disponibles; un modelo de estimación de costes para evaluar cada alternativa de asignación que considera las características

del sitio o localización y sus relaciones con otras tareas asignadas en otras localizaciones; y un modelo de identificación de riesgos para valorar y predecir los riesgos que pueden surgir de dichas asignaciones de trabajo.

Por su parte, se observa del análisis de la bibliografía que los modelos paramétricos de estimación tradicionales deben ser extendidos para considerar factores adicionales de complejidad en los que cubran las características especiales de DGS. En particular, COCOMO II ya considera un factor de coste denominado "Multisite development (SITE)" que tiene en cuenta el desarrollo distribuido, pero está enfocado en la perspectiva de infraestructura de soporte a la comunicación, lo que requiere incorporar más factores específicos para este tipo de desarrollos. En este sentido, (Keil et al., 2006) proponen nuevos factores a incorporar en el modelo COCOMO II así como una adaptación de alguno de los existentes para encajar mejor en la naturaleza de proyectos DGS. (Madachy, 2007) propone una extensión a las fórmulas del modelo COCOMO II que consideren el esfuerzo específico por fase, la distribución de trabajo a los equipos que trabajan en múltiples localizaciones y los atributos de cada equipo. Por su parte en (Vizcaíno et al., 2014) se propone un modelo de estimación que extiende la técnica de puntos función considerando nuevos factores de complejidad a tres niveles (factores a nivel global, factores del sitio, factores entre sitios).

En definitiva, la estimación aplicada a proyectos DGS se ha convertido en una línea de trabajo de gran interés y se observan ciertos avances aunque queda un importante camino por recorrer sobre todo a la hora de validar y calibrar los modelos propuestos para su aplicación en las empresas. Ello se confirma en estudios como el realizado por (Britto et al., 2014).

3.4. Formación

Una de las principales estrategias que permitirían minimizar los problemas que aparecen en el Desarrollo Global de Software (DGS), consiste en proporcionar una formación adecuada. Dicha formación debe permitir al alumno adquirir conocimientos, actitudes, habilidades y destrezas, en resumen, competencias para afrontar estos retos de manera eficaz. Concretamente, la literatura se describen numerosas dificultades a las que se han de enfrentar los ingenieros en DGS (Monasor et al., 2009; Nunamaker et al., 2009) derivadas de la interacción con equipos multiculturales y multilingües, menos oportunidades para que los miembros del equipo puedan comunicarse con la consecuente reducción de conversaciones informales (Palacio et al., 2009), la toma de decisiones requiere más tiempo (Wainfan, 2005) y crece el miedo por parte de algunos participantes a intervenir en las reuniones (Casey y Richardson, 2008; Casey, 2010), entre otras.

Para paliar lo anterior, se requiere adquirir una serie de competencias generales que son:

- Resolución de conflictos, que incluye entre otras la capacidad para hacer frente a situaciones difíciles y conflictivas (Niederman y Tan, 2011); el diagnóstico temprano de conflictos en el equipo virtual (Wainfan, 2005); actitud positiva y capacidad de motivación (Parvathanathan et al., 2007; Ocker et al., 2009).
- Trabajo en equipo, considerando entre otras, la capacidad para pensar desde la perspectiva del interlocutor (Favela y Peña-Mora, 2001) o la habilidad para ganar la confianza del equipo (Nguyen et al., 2006; Parvathanathan et al., 2007), el uso de estructuras de gratificación y recompensa o respuesta apropiada ante críticas o sugerencias (Niederman y Tan, 2011).
- Destrezas comunicativas, como el dominio de la lengua común empleada por la organización (Ebert y Neve, 2001) o el conocimiento de protocolos de comunicación y costumbres de las diferentes culturas (Richardson et al., 2007; Gotel et al., 2008), entre otras.

Por su parte, reproducir entornos DGS en contextos educativos es una tarea compleja. La mayoría de los estudios que se han encontrado en la literatura describen problemas organizativos cuando se trata de lograr la colaboración entre estudiantes de diferentes países. Por otra parte, los estudiantes que participan en estas actividades, poseen diferentes niveles de conocimiento o habilidades, lo que hace que sea necesario ofrecerles diferentes estrategias de entrenamiento (Soller, 2001). La cultura y lenguaje nativo de cada participante debe,

igualmente tenerse en cuenta en el diseño de dicha formación (Clear y Kassabova, 2008). Otro de los problemas destacados en la literatura es la ineficacia de la comunicación a través de medios como el correo electrónico o el chat, en ocasiones, relacionados con cuestiones técnicas (Daniels et al., 1998), lo que generalmente conlleva al incumplimiento de los plazos fijados.

Con el fin de proporcionar algunas soluciones a los problemas antes mencionados y formar adecuadamente a los desarrolladores de software en destrezas comunicativas en entornos globales, existen propuestas de simuladores como el entorno VENTURE (Virtual ENvironment for Training cUlture and language problems in global softwaRe dEvelopment) (Monasor et al., A Framework for Training Skills for Global Software Development, 2010). Este entorno presta especial atención a las diferencias culturales y lingüísticas, pero también es posible diseñar simulaciones de escenarios que permitan entrenar destrezas de trabajo en equipo, resolución de conflictos, así como escenarios específicamente enfocados a las distintas actividades que se pueden dar en DGS, tales como la elicitación de requisitos, la gestión del proyecto, la resolución de problemas, etc., y que impliquen una comunicación entre los participantes implicados.

3.5. Herramientas de soporte al DGS

Como ya hemos señalado, los retos que deben afrontar las organizaciones cuando los proyectos se llevan a cabo en un entorno global están principalmente relacionados con la distancia, ya sea geográfica, temporal o socio-cultural, lo que provoca un descenso importante de interacciones personales dificultando la colaboración, control y coordinación. Para minimizar los efectos negativos de la distancia, actualmente, existen tecnologías y aplicaciones que tratan de dar soporte a equipos de desarrollo que trabajan de manera distribuida, como en el caso de un proyecto de desarrollo global (Dubey y Hudepohl, 2013). Sin embargo, no todas las tecnologías y aplicaciones disponibles son capaces de ofrecer las mismas características para hacer frente a la colaboración en un entorno distribuido. Teniendo en cuenta que el desarrollo software está dividido en fases, cada aplicación deberá ofrecer características adaptadas a la fase de desarrollo para la que ha sido diseñada y además ofrecer características que mejoren la colaboración distribuida.

Con todo ello a continuación se ofrece una visión general de las herramientas software disponibles para DGS. En primer lugar, es importante considerar el conjunto de características que se deberían tener en cuenta a la hora de diseñar aplicaciones para DGS:

- Soporte al awareness. Significa que las herramientas deben incluir mecanismos a través de los cuales el usuario pueda conocer (ser consciente de) las acciones o eventos relevantes que sus compañeros de trabajo están realizando o produciendo.
- Soporte a la comunicación informal. En un entorno como el global donde los trabajadores prácticamente no pueden realizar charlas "cara-a-cara" con los compañeros que se encuentran en otra localización geográfica, es importante que las herramientas ofrezcan mecanismos para poder realizar charlas de una manera informal, directa y cómoda.
- Soporte al control y a la coordinación. Es importante que las herramientas proporcionen opciones de seguimiento de las tareas, errores, cambios, etc. e integren toda la información de manera que se pueda controlar el estado del proyecto.
- Análisis de dependencias socio-técnicas, mediante herramientas que consideren las relaciones sociales entre los miembros del equipo para realizar un análisis socio-técnico que ayude a mejorar la coordinación.
- Integración de datos. Debido al gran número de actividades realizadas durante el desarrollo software, es necesario usar diferentes tipos de herramientas para cubrir las necesidades de todas las actividades. Así, por ejemplo, es normal hacer uso de herramientas de control de versión, de seguimiento de errores, de peticiones de cambios o de generación de documentación entre otras.
- Soporte a la gestión del conocimiento, dado que en DGS el conocimiento se encuentra distribuido a través de los miembros de los distintos equipos. Esto hace necesario el uso de sistemas que ayuden

a capturar y distribuir dicho conocimiento como por ejemplo a través de Wikis o comunidades de práctica.

- Uso de versiones web de herramientas. Dado el alto grado de distribución en un entorno global, se debe permitir la interacción entre miembros del equipo desde cualquier lugar.

En (Alarcos, 2014) se presentan un conjunto de herramientas representativas que pueden ser útiles para DGS, clasificadas, de acuerdo a los procesos de ciclo de vida de ISO/IEC 12207, en las áreas de Planificación del Proyecto, Control y Aseguramiento del Proyecto, Análisis de Requisitos, Diseño del Software, Construcción del Software, Pruebas del Software, Gestión de la Documentación y Gestión de la Configuración. Por su parte, (García et al., 2011) presenta un conjunto de herramientas representativas de soporte a Ingeniería de Procesos que pueden ser de interés en DGS.

También resulta de interés considerar las herramientas que han sido desarrolladas en proyectos de investigación, y que dan también soporte a distintos procesos o actividades del ciclo de vida, como:

- Diseño distribuido: SYSIPHUS (Bruegge et al., Sysiphus: Enabling informal collaboration in global software development, 2006).
- Gestión de la Configuración: ADAMS (Bruegge et al., Supporting Distributed Software Development with fine-grained Artefact Management, 2006), Augur (Froehlich y Dourish, 2004), RepoGuard (Legenhausen et al., 2009), WikiDev (Bauer et al., 2009).
- Supervisión de Proyecto y actividades: WorldView (Al-Ani et al., 2008), WorkSpace Activity Viewer (Al-Ani et al., 2008), IssuePlayer (Garousi y Leitch, 2010), Implementación Syde (Hattori y Lanza, 2010), Share (Assogba y Donath, 2010).
- Clasificación de sistemas: MUDABlue (Kawaguchi et al., 2006).
- Gestión de Documentación DOCTOR (Krishnamurthy y Subramani, 2008) 4everedit (Meisinger et al., 2006).
- Gestión de Procesos XCHIPS (Fernández et al., 2004), GENESIS (Aversano et al., 2004).
- Gestión del Conocimiento: iBistro (Braun et al., 2002).

4. Conclusiones y Trabajos Futuros

En este artículo se ha presentado una panorámica general del estado del arte y de la práctica del DGS, analizando las principales revisiones sistemáticas de la literatura e identificando un conjunto de áreas de gran interés en la actualidad.

Una importante conclusión obtenida del estudio, es que el DGS es un paradigma muy dinámico que está en constante evolución. Una prueba de ello es por ejemplo el estudio realizado por (Vizcaíno et al., 2013), en el que se analizan los factores que afectan al DGS. Tradicionalmente se ha considerado como factores más críticos para los expertos las diferencias lingüísticas y culturales, así como la distancia geográfica (Carmel, 1999; Ågerfalk et al., 2005; Damian et al., 2005; Herbsleb, 2007; Cuppen et al., 2010). Sin embargo en base a los resultados de dicho estudio estos factores no son percibidos ya como clave por los expertos. En cambio, actualmente se consideran como los factores más importantes la motivación personal y las habilidades de los recursos humanos, al igual que disponer de funciones y responsabilidades bien definidas. Este resultado puede ser debido a que hoy en día estas distancias se han reducido gracias al uso de la tecnología adecuada y mediante la mejora del proceso software en entornos deslocalizados. Ello implica que el DGS es un campo que empieza a alcanzar cierta madurez y al mismo tiempo presenta nuevos desafíos centrados en importantes líneas de interés como:

- **Procesos para desarrollo y gestión**. Se observa que la tendencia es hacia la aplicación de métodos de gestión ágiles, como Scrum, pero adaptados a las particularidades y especial complejidad de desarrollos DGS, lo que supone retos importantes para que un equipo que trabaja de forma des-localizada

pueda funcionar como un equipo cohesivo y auto-organizado para el cumplimiento de sus objetivos.

- **Gestión de Proyectos DGS**, enfocada en una adecuada estimación de esfuerzo y costes del proyecto DGS considerando dimensiones adicionales así como estrategias efectivas de asignación de trabajo en las distintas localizaciones o a los distintos equipos distribuidos.

- **Equipos de Trabajo**, aspecto muy importante que debe permitir la formación de equipos muy cohesionados que alcancen adecuadamente sus objetivos en un entorno en el que hay que mejorar su comunicación, coordinación y control. Hoy en día se ha mejorado en infraestructura tecnológica que facilita lo anterior y se plantean desafíos constantes en la mejora de la congruencia socio-técnica en dichos equipos así como la importancia de una adecuada formación, donde los simuladores de entrenamiento pueden aportar importantes soluciones. También cobra mucho interés la mejora de las capacidades de los equipos que trabajan de forma distribuida. Ello motiva la necesidad de evolucionar al desarrollo de comunidades de práctica, como puede verse en (Monasor et al., 2013), donde los profesionales pueden compartir experiencias, lecciones aprendidas o "buenas prácticas" (Davila y Oktaba, 2013).

Agradecimientos

Este artículo ha sido financiado por los proyectos: GEODAS-BC (Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01); SDGear (TSI-100104-2014-4), enmarcado en la iniciativa ITEA 2 (Call 7), y cofinanciado por el Ministerio de Industria, Energía y Turismo dentro del Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2013-2016 y por el Fondo Europeo de Desarrollo Regional (FEDER); INGENIOSO (Junta de Comunidades de Castilla La Mancha, PEII11-0025-9533); y GLOBALIA (Junta de Comunidades de Castilla La Mancha, PEII11-0291-5274).

Cómo citar este artículo / How to cite this paper

Vizcaíno, A., García, F., y Piattini, M. (2014). Visión General del Desarrollo Global de Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 8-22. Consultado el [dd/mm/aaaa] en www.ijisebc.com

Referencias

Ågerfalk, P. J., et al. (2005). A framework for considering opportunities and threats in distributed software development International Workshop on Distributed Software Development. A. C. Society. Paris, France 47-61.

Al-Ani, B., et al. (2008). Continuous coordination within the context of cooperative and human aspects of software engineering. the International Workshop on Cooperative and Human Aspects of Software Engineering. Leipzig, Germany, ACM: 1-4.

Alarcos (2014). "Global Software Development Tools." from https://sites.google.com/site/toolsgsd/.

Ali, S. y Khan, S. U. (2014). Critical Success Factors for Software Outsourcing Partnership (SOP): A Systematic Literature Review. International Conference on Global Software Engineering (ICGSE 2014). Shanghai, China: 153-162.

Almehida, L. E., et al. (2011). Applying multi-criteria decision analysis to global software development with scrum project planning. International Conference on Rough Sets and Knowledge Technology (RSKT 2011). Banff, Canada.

Alsudairi, M. A. y Dwivedi, Y. K. (2010). "A multi-disciplinary profile of IS/IT outsourcing research." Journal of Enterprise Information Management 23(2): 215-258.

Ambler, S. W. (2009). "The Distributed Agile Team." Dr. Dobb's Journal 34(1): 45-47.

Assogba, Y. y Donath, J. (2010). Share: a programming environment for loosely bound cooperation. Proceedings of the 28th international conference on Human factors in computing systems. Atlanta, Georgia, USA, ACM: 961-970.

Audy, J., et al. (2004). Distributed Analysis The Last Frontier? the 37th Annual Hawaii International Conference on Systems Sciences (HICSS), Big Island (Hawaii).

Aversano, L., et al. (2004). "Managing coordination and cooperation in distributed software processes: the GENESIS environment."

Software Process: Improvement and Practice 9(4): 239-263.
Bass, J. M. (2013). Agile Method Tailoring in Distributed Enterprises: Product Owner Teams. International Conference on Global Software Engineering (ICGSE 2013): 154-163.
Bass, J. M. (2014). Scrum Master Activities: Process Tailoring in Large Enterprise Projects. International Conference on Global Software Engineering (ICGSE 2014): 6-15.
Batra, D. (2009). "Modified agile practices for outsourced software projects." Communications of the ACM 52(9).
Bauer, K., et al. (2009). WikiDev 2.0: discovering clusters of related team artifacts. the Conference of the Center for Advanced Studies on Collaborative Research. Ontario, Canada, ACM: 174-187.
Betz, S. (2010). Knowledge Transfer in IT Offshore Outsourcing Projects: An Analysis of the Current State and Best Practices. 2010 5th IEEE International Conference on Global Software Engineering. A. Oberweis and Stephan, R. Princeton, New Jersey, USA. 0: 330-335.
Boehm, B. (2006). A view of 20th and 21st century software engineering. Proceedings of the 28th International Conference on Software Engineering (ICSE 2006). Shanghai, China: 12-29.
Borges, A., et al. (2013). Ontologies supporting the distributed software development: a systematic mapping study. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, Porto de Galinhas, Brasil.
Braun, A., et al. (2002). iBistro: A Learning Environment for Knowledge Construction in Distributed Software Engineering Courses. the Ninth Asia-Pacific Software Engineering Conference, IEEE Computer Society: 197-203.
Britto, R., et al. (2014). Effort Estimation in Global Software Development: A Systematic Literature Review. International Conference on Global Software Engineering (ICGSE 2014): 135-144.
Bruegge, B., et al. (2006). Sysiphus: Enabling informal collaboration in global software development. International Conference on Global Software Engineering (ICGSE'06) Florianopolis, Brazil 139-148.
Bruegge, B., et al. (2006). Supporting Distributed Software Development with fine-grained Artefact Management. Proceedings of the IEEE international conference on Global Software Engineering, IEEE Computer Society: 213-222.
Budgen, D., et al. (2012). What scope is there for adopting evidence-informed teaching in SE? International Conference on Software Engineering (ICSE 2012).
Carmel, E. (1999). Global software teams: collaborating across borders and time zones. Upper Saddle River, NJ (USA), Prentice Hall PTR.
Carmel, E. y Agarwal, R. (2001). "Tactical Approaches for Alleviating Distance in Global Software Development." IEEE Software 18(2): 22-29.
Casey, V. (2010). "Developing trust in virtual software development teams." Special Issue on Trust and Trust Management of the Journal of Theoretical and Applied Electronic Commerce Research 5(2): 41-58.
Casey, V. y Richardson, I. (2008). The Impact of Fear on the Operation of Virtual Teams. IEEE International Conference on Global Software Engineering, IEEE Computer Society: 163-172.
Cataldo, M. y Herbsleb, J. (2011). Factors leading to integration failures in global feature-oriented development: an empirical analysis. Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011): 161-170.
Cataldo, M., et al. (2008). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. Kaiserslautern, Germany, ACM: 2-11.
Cataldo, M., et al. (2006). Identification of coordination requirements: implications for the Design of collaboration and awareness tools. the 20th Anniversary Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada, ACM: 353-362.
Clear, T. y Kassabova, D. (2008). A Course in Collaborative Computing: Collaborative Learning and Research with a Global Perspective. Proceedings of the 39th ACM Technical Symposium on Computer Science Education. M. Guzdial and Fitzgerald, S. Portland, Oregon, USA, ACM: 63-67.
Conchúir, E. (2010). Global Software Development: A Multiple-Case Study of the Realisation of the Benefits. Limerick (Ireland), University of Limerick: 262.
Conradi, R., et al. (2012). Dispersion, coordination and performance in global software teams: a systematic review. Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement: 129-138.
Costa, C., et al. (2010). Models and tools for Managing Distributed Software Development: A systematic literature review. International Conference on Evaluation and Assessment in Software Engineering (EASE 2010). Keele University, UK.
Costa, C. y Murta, L. (2013). Version Control in Distributed Software Development: a Systematic Mapping Study. International Conference on Global Software Engineering (ICGSE 2013). Bari, Italia.
Cuppen, E., et al. (2010). "Q methodology to select participants for a stakeholder dialogue on energy options from biomass in the Netherlands." Ecological Economics 69(3): 579-591.
Chauhan, M. A. y Ali Babar, M. (2012). Cloud infrastructure for providing tools as a service: quality attributes and potential solutions. Proceedings of the WICSA/ECSA 2012: 5-13.
da Silva, F. Q. B., et al. (2011). "Research and practice of distributed software development project management: A systematic mapping study." Information and System Technology.
Damian, D., et al. (2005). "Requirements Engineering and Downstream Software Development: Findings from a Case Study." Empirical Software Engineering 10(3): 255-283.
Damian, D., et al. (2004). Workshop Introduction. 3rd International Workshop on Global Software Development. Co-located with ICSE 2004, Edinburgh (Scotland).
Damian, D. y Moitra, D. (2006). "Guest Editors' Introduction: Global Software Development: How Far Have We Come?" IEEE Software 23(5): 17-19.

Daniels, M., et al. (1998). RUNESTONE, an International Student Collaboration Project. IEEE Frontiers in Education Conference, Tempe, Arizona, IEEE.
Davila, M. y Oktaba, J. (2013). Run-Through Practice as a Collaboration Facilitator in Inter-organizational Software Construction. PARIS Workshop en International Conference on Global Software Engineering. Bari (Italy): 31-40.
del Nuevo, E., et al. (2011). Scrum-based Methodology for Distributed Software Development. International Conference on Global Software Engineering (ICGSE 2011): 66-74.
Dubey, A. y Hudepohl, J. P. (2013). Towards Global Deployment of Software Engineering Tools. International Conference on Global Software Development (ICGSE 2013): 129-133.
Ebert, C. y Neve, P. D. (2001). "Surviving Global Software Development." IEEE Software 18(2): 62–69
Ebling, T., et al. (2009). A Systematic Literature Review of Requirements Engineering in Distributed Software Development Environments. ICEIS '09, Milan, Italy.
Ehrlich, K., et al. (2008). "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development." Mining Software Repositories.
Eskeli, J. y Maurolagoitia, J. (2011). Global Software Development: Current challenges and solutions. the 6th International Conference on Software and Data Technologies (ICSOFT), Seville (Spain).
Fauzi, S. S. M., et al. (2010). Software Configuration Management in Global Software Development: A Systematic Map. 17th Asia Pacific Software Engineering Conference, Sydney, Australia.
Favela, J. y Peña-Mora, F. (2001). "An Experience in Collaborative Software Engineering Education." IEEE Software 18(2): 47-53.
Fernández, A., et al. (2004). "Guided support for collaborative modeling, enactment and simulation of software development processes." Software Process: Improvement and Practice 9(2): 95-106.
Froehlich, J. y Dourish, P. (2004). Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. the 26th International Conference on Software Engineering, IEEE Computer Society: 387-396.
García, F., et al. (2011). "Process Management Tools." IEEE Software 28(2): 15-18.
Garousi, V. y Leitch, J. (2010). "IssuePlayer: An extensible framework for visual assessment of issue management in software development projects." Journal of Visual Languages & Computing 21(3): 121-135.
Genero, M., et al. (2014). Métodos de Investigación en Ingeniería del Software, Ra-Ma.
Giuffrida, R. y Dittrich, Y. (2013). "Empirical Studies on the Use of Social Software in Global Software Development - a Systematic Mapping Study." Information and Software Technology: 23.
Gotel, O., et al. (2008). Integration Starts on Day One in Global Software Development Projects. IEEE International Conference on Global Software Engineering (ICGSE'08). Bangalore, India, IEEE Computer Society: 244-248.
Hattori, L. y Lanza, M. (2010). Syde: a tool for collaborative software development. the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2. Cape Town, South Africa, ACM: 235-238.
Herbsleb, J. D. (2007). Global Software Engineering: The Future of Socio-technical Coordination. International Conference on Software Engineering: Future of Software Engineering (FOSE'07). Minneapolis, MN, USA, IEEE Computer Society: 188-198.
Herbsleb, J. D. y Moitra, D. (2001). "Global software development." IEEE Software 18(2): 16-20.
Hossain, E., et al. (2009). Using Scrum in Global Software Development: A Systematic Literature Review. Fourth IEEE International Conference on Global Software Engineering (ICGSE'09). Limerick, Ireland, IEEE Computer Society: 175-184.
Hossain, E., et al. (2011). Towards an understanding of tailoring scrum in global software development: a multi-case study. Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP 2011): 110-119.
Huang, H. (2007). Cultural influences and globally distributed information systems development: experiences from Chinese IT professionals. the ACM SIGMIS CPR Conference on Computer personnel research: The global information technology workforce. St. Louis, Missouri (USA): 36-45.
Humayun, M., et al. (2013). An empirical study on investigating the role of KMS in promoting trust within GSD teams. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE 2013): 207-211.
Islam, S., et al. (2009). Goal and Risk Factors in Offshore Outsourced Software Development from Vendor's Viewpoint. the 4th IEEE International Conference on Global Software Engineering (ICGSE 2009).
Jalali, S. y Wohlin, C. (2012). "Global Software Engineering and Agile Practices: A Systematic Review." Journal of Software: Evolution and Process 24(6): 643-659.
Jalali, S. y Wohlin, C. (2012). Systematic literature studies: Database searches vs. backward snowballing. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM: 29-38.
Jiménez Monasor, M. y Piattini, M. (2009). A Systematic Review of Distributed Software Development: Problems and Solutions. Software Engineering Approaches for Offshore and Outsourced Development (SEAFOOD 2008). Zurich, Switzerland.
Junius, B. A., et al. (2011). The impact of media selection on stakeholder communication in agile global software development: a preliminary industrial case study. Proceedings of the 49th SIGMIS annual conference on Computer personnel research: 131-139.
Kawaguchi, S., et al. (2006). "MUDABlue: An automatic categorization system for Open Source repositories." Journal of Systems and Software 79(7): 939-953.
Keil, P., et al. (2006). Cost estimation for global software development. Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research (EDSER '06). ACM. New York, NY, USA: 7-10.
Khan, A. A., et al. (2013). "Communication Risks and Best Practices in Global Software Development during Requirements Change Management: A Systematic Literature Review Protocol." Research Journal of Applied Sciences, Engineering and Technology 6(19): 3514.
Khan, A. W. y Khan, S. U. (2014). Critical challenges in execution of offshore software outsourcing contract from vendors' perspective: A systematic literature review. International Conference on Information and Communication Systems (ICICS 2014).

Khan, H. H., et al. (2013). Situational factors affecting Requirement Engineering process in Global Software Development. IEEE Conference on Open Systems (ICOS),: 118-122.
Khan, S. U., et al. (2009). Critical Success Factors for Offshore Software Development Outsourcing Vendors: A Systematic Literature Review. Fourth IEEE International Conference on Global Software Engineering (ICGSE'09). Limerick, Ireland, IEEE Computer Society: 207-216.
Khan, S. U., et al. (2011). "Barriers in the selection of offshore software development outsourcing vendors: An exploratory study using a systematic literature review." Information and Software Technology 53(7): 693-706.
Kobitzsch, W., et al. (2001). "Outsourcing in India." IEEE Software 18(2): 78-86.
Krishnamurthy, T. y Subramani, S. (2008). Ailments of Distributed Document Reviews and Remedies of DOCTOR (DOCument Tree ORganizer Tool) with Distributed Reviews Support. IEEE International Conference on Global Software Engineering (ICGSE 2008). Bangalore, India: 210-214
Kroll, J., et al. (2013). A Systematic Literature Review of Best Practices and Challenges in Follow-the-Sun Software Development. International Conference on Global Software Engineering Workshops (ICGSEW 2013). Bari, Italia: 18 - 23.
Kroll, J., et al. (2011). Mapping the Evolution of Research on Global Software Engineering - A Systematic Literature Review. Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS 2011). Beijing, China. 3.
Kussmaul, C., et al. (2004). Outsourcing and Offshoring with Agility: A Case Study (Experience Paper). Proceedings of XP/Agile Universe: 147-154.
Kwan, I., et al. (2009). A Weighted Congruence Measure. Workshop on SocioTechnical Congruence 1-4.
Kwan, I., et al. (2011). "Does Socio-Technical Congruence Have An Effect on Software Build Success ? A Study of Coordination in a Software Project." IEEE Computer Society: 1-20.
Lamersdorf, A., et al. (2009). A Survey on the State of the Practice in Distributed Software Development: Criteria for Task Allocation. International Conference on Global Software Engineering, Limerick, Ireland.
Lamersdorf, B. A., et al. (2010). Estimating the Effort Overhead in Global Software Development. 5th International Conference on Global Software Engineering (ICGSE 2010): 267-276.
Lamersdorf, B. A., et al. (2010). A Rule-Based Model for Customized Risk Identification in Distributed Software Development Projects. 5th International Conference on Global Software Engineering (ICGSE 2010): 209-218.
Lee, S. y Yong, H. S. (2009). "Distributed agile: project management in a global environment." Empirical Software Engineering: 1-14.
Legenhausen, M., et al. (2009). RepoGuard: A Framework for Integration of Development Tools with Source Code Repositories. the Fourth IEEE International Conference on Global Software Engineering, IEEE Computer Society: 328-331.
Lopez, A., et al. (2009). Risks and Safeguards for the Requirements Engineering Process in Global Software Development. International Conference on Global Software Engineering (ICGSE 2009), Limerick, Ireland.
Madachy, R. (2007). "Distributed Global Development Parametric Cost Modeling."
Meisinger, M., et al. (2006). "4everedit — Team-based Process Documentation Management." Software Process: Improvement and Practice 11(6): 627-642.
Monasor, M. J., et al. (2009). "Challenges and Improvements in Distributed Software Development: A Systematic Review." Advances in Software Engineering: 1-16.
Monasor, M. J., et al. (2010). A Framework for Training Skills for Global Software Development. International Conference on Global Software Development (ICGSE 2010), Princeton, NJ, USA.
Monasor, M. J., et al. (2010). Preparing students and engineers for Global Software Development: A Systematic Review. the International Conference on Global Software Development (ICGSE 2010), Princeton, NJ, USA.
Monasor, M. J., et al. (2013). Towards a Global Software Development Community Web: Identifying Patterns and Scenarios. PARIS Workshop en International Conference on Global Software Engineering. Bari (Italy): 41-46.
Nguyen-Duc, A., et al. (2014). "The impact of global dispersion on coordination, team performance and software quality – A systematic literature review." Information and Software Technology.
Nguyen, P. T., et al. (2006). Critical factors in establishing and maintaining trust in software outsourcing relationships. the 28th International Conference on Software Engineering. Shanghai, China, ACM: 624-627.
Niazi, M., et al. (2013). Challenges of project management in Global Software Development: Initial results. Science and Information Conference (SAI): 202-206.
Niazi, M. K., et al. (2013). Motivators of Adopting Social Computing in Global Software Development: Initial Results. Proceedings of the World Congress on Engineering 2013. London, U.K. 1: 1-5.
Niederman, F. y Tan, F. (2011). "Managing global IT teams: considering cultural dynamics." Commun. ACM 54(4): 24-27.
Noll, J., et al. (2010). "Global software development and collaboration: barriers and solutions." Magazine ACM Inroads 1(3): 66-78.
Nour, A. (2010). Architectural Knowledge Management in Global Software Development: A Review. 5th IEEE International Conference Global Software Engineering (ICGSE). Princeton, NJ, USA. 0: 347-352.
Nunamaker, J., et al. (2009). "Principles for effective virtual teamwork." Commun. ACM 52(4): 113-117.
Nurdiani, I., et al. (2011). Risk Identification and Risk Mitigation Instruments for Global Software Development: Systematic Review and Survey Results. Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW).
Ocker, R., et al. (2009). "Training Students to Work Effectively in Partially Distributed Teams." ACM Transactions on Computing Education (TOCE) 9(1): 1-24.
Palacio, R., et al. (2009). "Providing Support for Starting Collaboration in Distributed Software Development: A Multi-agent Approach." WRI World Congress on Computer Science and Information Engineering 7: 397-401.
Parvathanathan, K., et al. (2007). Global Development and Delivery in Practice: Experiences of the IBM Rational India Lab, IBM Press.

Peixoto, C. E. L., et al. (2010). Effort Estimation in Global Software Development Projects: Preliminary Results from a Survey. 5th IEEE International Conference on Global Software Engineering (ICGSE 2010): 123-127.
Persson, J. S. (2010). A Process for Managing Risks in Distributed Teams. L. Mathiassen. 27(1): 20-29.
Portillo-Rodríguez, J., et al. (2012). "Tools used in Global Software Engineering: A systematic mapping review." Journal Information and Software Technology 54(7): 663-685.
Portillo-Rodríguez, J., et al. (2014). "Using agents to manage Socio-Technical Congruence in a Global Software Engineering project." Inf. Sci. 264: 230-259.
Prikladnicki, R. y Audy, J. L. N. (2010). "Process models in the practice of distributed software development: A systematic review of the literature." Inf. Softw. Technol. 52(8): 779-791.
Prikladnicki, R., et al. (2008). Patterns of Evolution in the Practice of Distributed Software Development: Quantitative Results from a Systematic Review. 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) University of Bari, Italy.
Raza, B., et al. (2013). Topics and treatments in global software engineering research - A systematic snapshot. International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013): 85-96.
Richardson, I., et al. (2005). Global Software Development – the Challenges. SERC Technical Report 278, University of Limerick, Ball State University: 10.
Richardson, I., et al. (2007). Globalizing Software Development in the Local Classroom. the 20th Conference on Software Engineering Education & Training, IEEE Computer Society: 64-71.
Rocha, R., et al. (2011). "Collaboration Models in Distributed Software Development: a Systematic Review." CLEI Electronic Journal.
Sarma, A., et al. (2008). Challenges in Measuring, Understanding, and Achieving Social-Technical Congruence. CMU-ISR-08-106.
Schneider, S., et al. (2013). "Solutions in Global Software Engineering: A Systematic Literature Review." International Journal of Information Management 33(1): 119-132.
Šmite, D., et al. (2008). Reporting Empirical Research in Global Software Engineering: A Classification Scheme. IEEE International Conference Global Software Engineering (ICGSE), Bangalore, India.
Šmite, D. y Wohlin, C. (2011). "A Whisper of Evidence in Global Software Engineering." IEEE Software 28(4): 15-18.
Sobiech, F., et al. (2014). On iteration optimization for non-cross-functional teams in Scrum. Proceedings of the Conference on Research in Adaptive and Convergent Systems (RACS 2014): 266-271.
Soller, A. (2001). "Supporting Social Interaction in an Intelligent Collaborative Learning System." International Journal of Artificial Intelligence in Education (IJAIED) 12: 40-62.
Šteinberga, L. y Šmite, D. (2011). Towards Understanding of Software Engineer Motivation in Globally Distributed Projects. Proceedings of the 2011 IEEE Sixth International Conference on Global Software Engineering Workshop (ICGSE 2011): 117-119.
Steinmacher, I., et al. (2010). Awareness Support in Global Software Development: A Systematic Review Based on the 3C Collaboration Model. International Conference on Collaboration and Technology (CRIWG 2010). Maastricht, The Netherlands: 185-201.
Treude , C., et al. (2009). Empirical Studies on Collaboration in Software Development: A Systematic Literature Review.
Verner, J. M., et al. (2012). Systematic literature reviews in global software development: A tertiary study. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012), Ciudad Real (Spain).
Verner, J. M., et al. (2014). "Risks and risk mitigation in global software development: A tertiary study." Information & Software Technology 56(1): 54-78.
Vivian, R. L. M. H., Elisa Hatsue, et al. (2011). Context-awareness on software artifacts in distributed software development: a systematic review. International Conference on Collaboration and Technology (CRIWG 2011): 30-44.
Vizcaíno, A., et al. (2014). Desarrollo Global de Software, Ra-Ma.
Vizcaíno, A., et al. (2013). "Applying Q-methodology to analyse the success factors in GSD." Information and Software Technology 55(7): 1200-1211.
Vlietland, J. y Van Vliet, H. (2014). "Towards a governance framework for chains of Scrum teams." Information & Software Technology 57: 52-65.
Wainfan, L. (2005). Challenges in Virtual Collaboration: Videoconferencing Audioconferencing and Computer-Mediated Communications, RAND Corporation.

© ISSN: 2387-0184

International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)

IJISEBC, 01, I, 2014

Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software

Indra's technological proposal to tackle the immediate challenges of Software Engineering

Gabriel Sánchez Belmonte[1], Carlos García Moreno[1], Yolanda Hernández González[1]

[1] INDRA

gsanchez@indra.es, cgarciamo@indra.es, yhernandezg@indra.es

RESUMEN. Este artículo: muestra cómo las nuevas tecnologías que se están aplicando en diversos ámbitos lúdicos y productivos se pueden adaptar y aplicar para dar respuesta a los principales retos existentes en el sector de la Ingeniería del Software; expone las propuestas tecnológicas en las que Indra está trabajando para proporcionar el soporte tecnológico necesario para cubrir los distintos retos y oportunidades que se plantean en la Ingeniería de Software; y presenta la Suite MInd y sus principales características, la cual supone el elemento vertebrador de las propuestas tecnológicas planteadas.
Destacando que la Ingeniería del Software es un sector que necesita incorporar el dinamismo y agilidad tanto en la adopción efectiva de nuevas metodologías, como tecnologías y herramientas, para dar respuesta a los retos planteados en la actualidad y en el futuro inmediato, que pueden ser vistos como importantes oportunidades competitivas.

ABSTRACT. This paper: shows how new technologies that are being applied in various ludic and productive areas can be adapted and applied to respond to major challenges in the field of Software Engineering; exposes the technological proposals that Indra is working to provide necessary technical support to cover the various challenges and opportunities that arise in Software Engineering; and presents the Suite MInd and its main characteristics, which assumes the backbone of the raised technological proposals.
Stressing that Software Engineering is a sector that needs to incorporate the dynamism and agility in both the effective adoption of new methodologies and technologies and tools, to respond to the challenges today and in the near future, which can be seen as important competitive opportunities.

PALABRAS CLAVE: Ingeniería de Software, Retos de la Ingeniería de Software, Propuestas tecnológicas, Suite MInd, Big Data, Gamificación, Tecnologías colaborativas, Serious Games, DevOps.

KEYWORDS: Software Engineering, Challenges of Software Engineering, Technological proposals, Suite MInd, Big Data, Gamification, Collaborative Technologies, Serious Games, DevOps.

Sánchez Belmonte, G., García Moreno, C., y Hernández González, Y. (2014). Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 24-36. Consultado el [dd/mm/aaaa] en www.ijisebc.com

1. Introducción

El término Ingeniería del Software tiene más de sesenta años, una antigüedad más que suficiente para que esta disciplina hubiera alcanzado una madurez y grado de fiabilidad muy alto. Sin embargo, no parece ser así, siendo frecuentes todavía las discusiones sobre las estimaciones o la calidad del software entregado, muchos proyectos siguen teniendo retrasos, sobrecostes, etc.

La tecnología ha evolucionado de forma muy rápida en los últimos años. En la actualidad el cambio es todavía más rápido y en el futuro todo parece indicar que la tecnología seguirá evolucionando de forma cada vez más rápida. Se trata de una evolución exponencial, por lo que reputados expertos predicen que en unos años muchos de nosotros no podremos ni siquiera asumir o entender los cambios que se produzcan. La Ingeniería del Software continúa avanzando de forma más o menos continua, aparecen nuevas metodologías, técnicas y herramientas pero, ¿avanza al mismo ritmo que el entorno tecnológico global? Si comparamos estos avances (tecnológicos y metodológicos) en los últimos veinte años con un caso paradigmático de evolución tecnológica, como es la telefonía móvil, la respuesta parece clara. Se podría decir que, a diferencia del entorno tecnológico global, la Ingeniería del Software evoluciona de forma aritmética. Partiendo de esta situación nos preguntamos si esta evolución es lo suficientemente rápida, y más aún, si las organizaciones están demostrando ser capaces de incorporar los cambios producidos. La respuesta es obvia, no.

Un análisis de esta situación nos induce a hacernos otras preguntas: ¿Por qué el sector de la Ingeniería del Software no está a la cabeza en la adopción de las novedades tecnológicas que sí se incorporan en otros campos? ¿Qué ventajas competitivas supondrían para las empresas incorporar estas tecnologías para dar respuesta a los nuevos (y en algunos casos no tanto) retos en el sector? ¿Qué futuro tienen las grandes organizaciones que no son capaces de adoptar estos cambios?

En este artículo vamos a dar nuestra visión sobre las oportunidades que ofrecen las tecnologías que consideramos deberían guiar el futuro de la Ingeniería del Software en base a los retos a los que se enfrenta, y cómo estamos afrontándolos en Indra a través de la innovación tecnológica. Queremos destacar que el objetivo del artículo no es abordar todas las fases del ciclo de vida de Ingeniería del Software, sino hacer hincapié y profundizar en los aspectos detectados que suponen los principales retos y oportunidades, que son en los que se basa nuestra propuesta tecnológica actual para la evolución de la Ingeniería del Software.

2. Retos inmediatos en Ingeniería del Software

Los principales retos a los que se enfrenta la Ingeniería del Software no son nuevos, de hecho podríamos decir que algunos parecen asimilados con resignación por parte de las empresas, e incluso a veces de los clientes, al no encontrarse una respuesta realmente eficaz a los mismos.

Entre los retos "clásicos" encontramos los relacionados con la calidad y la productividad: sobrecostes, errores, funcionalidad no (o incorrectamente) implementada, retrasos, etc. Estos retos se pueden resumir en "insatisfacción del cliente". Los retos "modernos" a los que se enfrenta el sector desde hace más de una década vienen de la mano del fenómeno de la globalización, desde dos puntos de vista distintos. En primer lugar en lo que respecta a la competencia que suponen los países emergentes y, en segundo, respecto a lo que la globalización tecnológica supone.

Desde nuestro punto de vista todos estos retos suponen en realidad oportunidades. Las empresas que puedan por fin dar respuesta a los retos de siempre y saquen partido de las oportunidades que ofrece el panorama cambiante tecnológico y socioeconómico tendrán una clara ventaja competitiva y, nos atrevemos a aventurar que, serán las únicas que sobrevivan, en un sector que debería ser cada vez más sostenible.

En relación a esto, a continuación analizamos un conjunto de retos concretos que consideramos más destacables, si bien somos conscientes que no son los únicos, y que no reflejan el ciclo completo de la

Sánchez Belmonte, G., García Moreno, C., y Hernández González, Y. (2014). Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 24-36.
Consultado el [dd/mm/aaaa] en www.ijisebc.com

Ingeniería del Software.

2.1. Satisfacción del cliente

En ocasiones parecemos olvidar cuál es el fin del proceso de construcción del software: satisfacer las necesidades del cliente. El software construido debe dar respuesta a los requisitos y a las expectativas depositadas en él. Y aquí es imprescindible poner el foco en la toma de requisitos y la ejecución de las pruebas, sin olvidar el resto de fases del proceso. Parece obvio, pero hoy en día es un aspecto que sigue sin estar resuelto.

Vamos a suponer que realizamos una correcta toma de requisitos y que somos capaces de entender lo que nuestro cliente nos demanda. Uno de los aspectos a tener en cuenta, y olvidado en muchas ocasiones, es que los requisitos se deben trazar directamente con el resto de fases, hasta llegar a las pruebas, debiendo los casos de prueba verificar que el software obtenido cubre el alcance descrito en la toma de requisitos. Un error bastante común es probar el software directamente contra el diseño funcional o contra el diseño técnico, o lo que es peor, teniendo en cuenta únicamente cómo se ha codificado. Además, una insuficiente trazabilidad impacta también en esas fases, produciendo errores que propagan a las siguientes. Otro error grave es reducir el periodo de pruebas ya que, al tratarse de una de las fases que se encuentran al final del ciclo de vida del software, con más frecuencia de lo deseable se trata de reducir tiempos en ella para cubrir posibles desvíos en fases anteriores.

También es habitual olvidarse de que, además de funcionar correctamente, el tiempo de respuesta del sistema debe ser adecuado. Para ello hay que realizar pruebas de rendimiento (concurrencia de usuarios, volumen de datos, etc.), cuyo diseño y ejecución no deben dejarse para el final del proyecto, sino que deben abordarse concurrentemente al desarrollo. También debemos tener en cuenta las pruebas de regresión en un entorno de entrega continua.

¿Qué soluciones deberíamos dar a estos retos? Desde nuestro punto de vista sería necesario un trazado automático de los requisitos de usuario con las pruebas, mecanismos que permitan dimensionar de forma lo más exacta posible el resto de fases del proyecto a partir de estos requisitos, además de mecanismos que permitan una monitorización a alto nivel de un proyecto concreto y de la cartera de proyectos, previendo con la suficiente antelación (y analizando las causas de) desviaciones respecto a los objetivos (funcionales y técnicos) marcados, duración, costes, etc.

Por otro lado, serían especialmente útiles, mecanismos que proporcionen al usuario de las herramientas monitorización sobre la idoneidad del uso que esté haciendo de las mismas, y sobre todo que le motive a la hora de seguir de forma óptima la metodología propuesta al inicio del proyecto, incentivándoles a conseguir la excelencia en todo el proceso. Además se considera necesario proporcionar mecanismos de aprendizaje que eliminen y prevengan posibles malas prácticas, y un incremento de la usabilidad de las herramientas utilizadas, que facilite la correcta aplicación del proceso de Ingeniería del Software. Estos puntos entroncan con el concepto de productividad que tratamos a continuación.

Consideramos que la agilidad en todo el proceso, involucrando al usuario e integrando producción y desarrollo, supone otra respuesta a estos retos, como veremos más adelante.

2.2. Productividad / motivación

En cualquier actividad productiva es fundamental la motivación de las personas involucradas, tanto en la calidad como en la eficiencia de los proyectos que dirigen y ejecutan. Más aún en el entorno globalizado, en el que la Ingeniería del Software está cada vez más inmerso, y donde es necesario ser cada vez más competitivos. Además, el objetivo primordial del software que se desarrolla es ofrecer, no sólo respuestas a las necesidades más obvias (las explicitadas), sino también valor añadido a los clientes (y a su vez a los usuarios finales). Para ello es imprescindible el compromiso de los profesionales involucrados con la productividad y con la excelencia, de cara a conseguir dar respuesta a los objetivos reales del cliente de forma proactiva. Se trata, desde

nuestro punto de vista, de una actividad que no debe basarse únicamente en la mecanización, sino que, en cada fase del proceso, cada profesional debe aportar sus capacidades únicas para poder obtener los mejores resultados.

La necesidad de concienciar a los profesionales de la necesidad de su compromiso con la productividad y la excelencia plantea retos aún no resueltos. Para ello, en primer lugar, sería necesario poder contar con mecanismos que permitan monitorizar y valorar de forma objetiva las aportaciones de los profesionales en base a estos criterios, lo cual no es sencillo, teniendo en cuenta que muchos de ellos tradicionalmente se han venido valorando de forma subjetiva (proactividad, creatividad, colaboración, etc.). Además, los criterios supuestamente objetivos, en la práctica hoy en día son prácticamente imposibles de medir. Por ejemplo, los mecanismos utilizados para "medir" la productividad (número de líneas de código, de requisitos implementados, etc.) están afectados por factores que impiden (o dificultan mucho) su objetivización (complejidad y calidad del código, dificultad real de los requisitos, código heredado, etc.). A partir de esta monitorización se podrían seleccionar y aplicar los mecanismos de motivación más adecuados para cada persona. La usabilidad de las herramientas de ingeniería es otro aspecto fundamental para la productividad, facilitando (y optimizando) su correcto uso.

Para dar respuesta a los retos planteados también sería necesario contar con mecanismos para la asignación óptima de profesionales a proyectos, teniendo en cuenta el perfil de los mismos en base a los criterios expuestos. De esta forma se conseguiría optimizar no solo la productividad, sino también la obtención de los resultados que satisfagan a clientes y usuarios finales. Las necesidades de los proyectos en cuanto al dimensionamiento y capacidades del equipo varían con el tiempo, por lo que estos mecanismos deberían ser flexibles, en base a las características cambiantes de los proyectos, y anticipándose a estas variaciones.

2.3. Metodologías ágiles / nuevas formas de hacer las cosas

Como hablábamos al principio del artículo el mundo está cambiando a una velocidad que en ocasiones nos impide aceptar (o cuanto menos adaptarnos a) los cambios. En la Ingeniería del Software sucede algo parecido. Desde hace algunos años están apareciendo términos asociados a nuevas formas de trabajar (metodologías) como Extreme Programming, SCRUM, Agile, etc. que, si bien son conocidos, la realidad nos demuestra que en función de la compañía o del proyecto, no han sido adoptados, y cuando se han adoptado muchas veces ha sido de forma parcial, o incorrecta.

Estas metodologías presentan características/objetivos comunes: involucrar más al usuario en el proceso de desarrollo y entrega del producto final, compartir más información entre el equipo (trabajar entre pares, reuniones diarias, etc.) y plazos de entrega más cortos. Estas características ponen el foco en aspectos que las metodologías tradicionales no acababan de resolver, al tener proyectos demasiado largos donde el cliente daba sus requisitos al principio y al cabo de unos meses veía una primera versión del producto, que en muchos casos no tenía mucho que ver con lo que realmente necesitaba.

Si estas metodologías cubren aspectos a mejorar lo normal debería ser que se adoptaran de forma natural en los proyectos, entonces ¿por qué no se ha producido una adopción efectiva de los mismos? En nuestra opinión esto es debido, por un lado, al esfuerzo que supone adaptarse a ese cambio, y por otro a la resistencia al mismo. A muchas personas, en todos los niveles de las organizaciones, les cuesta adaptarse al uso de nuevas herramientas. Incluso actividades realizadas por la mayoría de las personas en la vida cotidiana encuentran oposición a la hora de ser realizadas en el entorno productivo, como puede ser la interacción a través de redes sociales, aspecto que analizaremos más adelante.

Aparte de las metodologías ágiles para las fases de desarrollo, también han surgido en los últimos años intentos metodológicos de trasladar el concepto de agilidad a las operaciones. Este es el caso de DevOps, que además propone la integración de ambas actividades (desarrollo y operación). Si las metodologías ágiles no se encuentran en la práctica implantadas efectivamente en la mayoría de las empresas, DevOps está aún más lejos de serlo, ya que implica la involucración y colaboración continua entre departamentos distintos, hasta el punto

de fusionarse en uno solo.

Estas dificultades a la hora de incorporar nuevas metodologías requieren de la incorporación de mecanismos que motiven a las personas a la hora de la adaptación a los cambios requeridos. Además, para paliar el esfuerzo que supone el cambio, serían necesarios medios que disminuyan la curva de aprendizaje de las nuevas metodologías y de las herramientas correspondientes para su soporte, además de una mejora de su usabilidad.

También consideramos necesarios mecanismos para analizar el comportamiento de las personas dentro de los equipos de Ingeniería del Software, a través de los patrones de uso de las herramientas. De esta forma se podría monitorizar el grado de adaptación a las nuevas herramientas y metodologías, y detectar mejoras a incorporar en las mismas.

2.4. Dispersión geográfica / diferencias culturales / idioma

Uno de los mayores retos a los que nos podemos enfrentar cuando queremos implementar un ALM (Application lifecycle management) en una compañía global es la dispersión geográfica. Disponer de equipos ubicados en diferentes geografías trae implícitos una serie de problemas: diferencias horarias, culturales e idioma. En este sentido, si cambiamos el punto de vista, podremos convertir lo que aparentemente parecen dificultades en la oportunidad que supone contar con una fuente capaz de generar continuamente conocimiento e ideas, y de enriquecer el generado por compañeros en otras partes del mundo, aportando puntos de vista distintos. Nuestra visión respecto a este fenómeno, por lo tanto, traduce los aparentes problemas en oportunidades de generación de conocimientos e innovaciones continuas.

Para poder aprovechar estas oportunidades es necesario evitar que los equipos trabajen como elementos estancos, sin compartir información, lo cual, obviamente, impacta en la ejecución y en el éxito de los proyectos. En este sentido es imprescindible que las personas sean capaces de trabajar en equipo y que se produzca una colaboración efectiva, principalmente con los miembros de su mismo equipo, pero también con el resto de la organización. Además, hay que tener en cuenta la dificultad añadida que supone la existencia de diferencias idiomáticas.

¿Cómo podemos articular la cultura de la colaboración y la innovación en un entorno de dispersión geográfica y cultural?, ¿cómo podemos hacer que el conocimiento compartido dentro de un equipo esté disponible para toda la compañía global? Una vez más, pensamos que la respuesta está en proporcionar mecanismos de motivación y de adaptación a nuevas formas de trabajo. En el siguiente punto vamos a tratar nuevos mecanismos para dar respuestas a este reto.

2.5. Cultura de la colaboración y la innovación

A parte de las implicaciones socioeconómicas, hay que tener en cuenta que la tecnología también ha contribuido a que vivamos y trabajemos en un entorno globalizado. Actualmente todo el mundo puede llegar a estar conectado en tiempo real gracias a las nuevas tecnologías, lo que supone que personas de distintos países, culturas, idiomas, etc. interactúan a diario y de forma continua en redes sociales, espacios abiertos, mensajería síncrona o asíncrona, etc.

Una comunidad de ingenieros de software de una compañía debe disponer de una metodología, procesos y herramientas que les permitan formar esa red profesional y que reduzca al máximo los retos asociados a la dispersión geográfica: distintos horarios, distintos idiomas, culturas, etc. Actualmente existen este tipo de espacios en muchas empresas, pero ¿se utilizan bien? Y, en los casos en los que el uso que se da a estas redes es el adecuado ¿se aprovecha realmente el conocimiento generado?

Uno de los casos más conocidos de éxito del aprovechamiento y puesta en valor del conocimiento generado en las redes sociales es el de las elecciones de Estados Unidos en 2012, donde a partir del estudio de la interacción social en estas redes se detectaron a los grupos clave, su ubicación, gustos, etc. para diseñar cuándo,

cómo y a través de qué medios hacerles llegar los mensajes de la campaña. Esta información se extrajo a partir de un análisis en gran parte manual, para el cual se precisó una gran inversión económica.

En el sector de la Ingeniería del Software (como en la mayoría) esta información no está siendo explotada de forma que se extraiga conocimiento clave a partir de ella. Es más, en los casos en los que se disponen de mecanismos "sociales" muchas veces se produce el fenómeno conocido como "infoxicación". Del contenido vertido en las interacciones en redes sociales, debidamente analizado, se podría obtener información relevante sobre el conocimiento, capacidades e intereses reales de los profesionales, detectando a nivel global los expertos en cada campo (lo cual actualmente parece inabarcable en una empresa con miles de profesionales en todo el mundo), y pudiendo evitar que se realicen una y otra vez las mismas consultas a estos expertos. Se podrían emplear estas redes para trabajar colaborativamente, y que todos los equipos de la empresa tuvieran acceso al conocimiento generado en esas interacciones.

En Ingeniería del Software el conocimiento debería extraerse de forma continua, por lo que sería necesario contar con mecanismos automáticos de análisis de información específica vertida en las redes sociales profesionales, que permitiesen tener acceso al conocimiento requerido en cada momento. Estamos hablando, al fin y al cabo, de facilitar a los profesionales su adaptación a la cultura de la colaboración, incrementando al mismo tiempo su productividad.

Por otro lado, vemos especialmente necesario incorporar la cultura de la innovación (y la colaboración en esa innovación) al campo de la Ingeniería del Software. Estamos en un sector que, por la forma tradicional de trabajar (que no ha cambiado mucho en las últimas décadas), no contempla la innovación como herramienta de trabajo. Una empresa de Ingeniería del Software que cuente con cultura de la innovación estará capacitada para afrontar los retos y aprovechar las oportunidades de forma distinta a la tradicional, pudiendo idear dentro de cada proyecto nuevas formas de satisfacer a clientes y usuarios finales, de resolver los problemas que se presenten, e incluso idear cambios en la forma de trabajar que mejoren la productividad. Las redes sociales también nos parecen un mecanismo idóneo para articular esta cultura de la innovación colaborativa. En este caso nos enfrentamos al reto de detectar conexiones y evitar duplicar líneas de innovación, ideas, etc.

Hasta este punto hemos analizado los retos que, desde nuestro punto de vista, se plantean en el sector de la Ingeniería del Software de cara a posibilitar la sostenibilidad del mismo en los próximos años, y a incrementar la competitividad de las empresas. También hemos visto que el sector no está respondiendo de forma ágil a los avances tecnológicos y metodológicos que están produciéndose cada vez a mayor rapidez, fallando en la adopción de los mismos y perdiendo la oportunidad de aprovechar las posibilidades que ofrecen. A continuación vamos a exponer la visión tecnológica de Indra en relación a la aplicación de estas tecnologías, como soporte fundamental para dar respuesta a los retos planteados.

3. Propuesta tecnológica para los retos actuales y futuros de la Ingeniería del Software

Indra apuesta fuertemente por la innovación en todos los sectores. En el caso concreto del sector de la Ingeniería del Software, en los últimos años, la actividad innovadora de la compañía se ha traducido en una propuesta tecnológica, cuyos resultados se centran en proporcionar mejoras a la Suite MInd. Esta Suite supone el soporte tecnológico principal para los objetivos de industrialización de la producción de la compañía y proporciona toda la funcionalidad necesaria para soportar el ciclo de vida de aplicaciones de software tanto en el ámbito de Proyectos como de Servicios. Está construida sobre un conjunto de herramientas integradas entre sí (véase figura 1) y soporta distintas metodologías con la posibilidad de disponer de distintos flujos de trabajo y transiciones de estado en función de cada necesidad. MInd aporta a estos procesos un soporte estratégico, táctico y operativo.

Adicionalmente, MInd soporta y facilita la gestión del conocimiento, la automatización de tareas, el control

de parámetros de calidad y la comunicación eficiente y sistematizada entre los miembros de los equipos de trabajo. También proporciona las herramientas y los procedimientos para adaptarse a los estándares internacionales de calidad en procesos de software.

La Suite MInd integra herramientas comerciales, open source así como desarrollos propios. Indra ha ampliado las capacidades de todas estas herramientas para proporcionar mejoras y una estandarización de su uso dentro del ciclo de vida de los proyectos y servicios permitiendo la integración entre ellas para proporcionar un valor añadido a lo que esas herramientas proporcionan por separado. De este modo se logra mantener la trazabilidad entre los diferentes procesos de una organización proveedora o consumidora de servicios de consultoría o desarrollo informático.

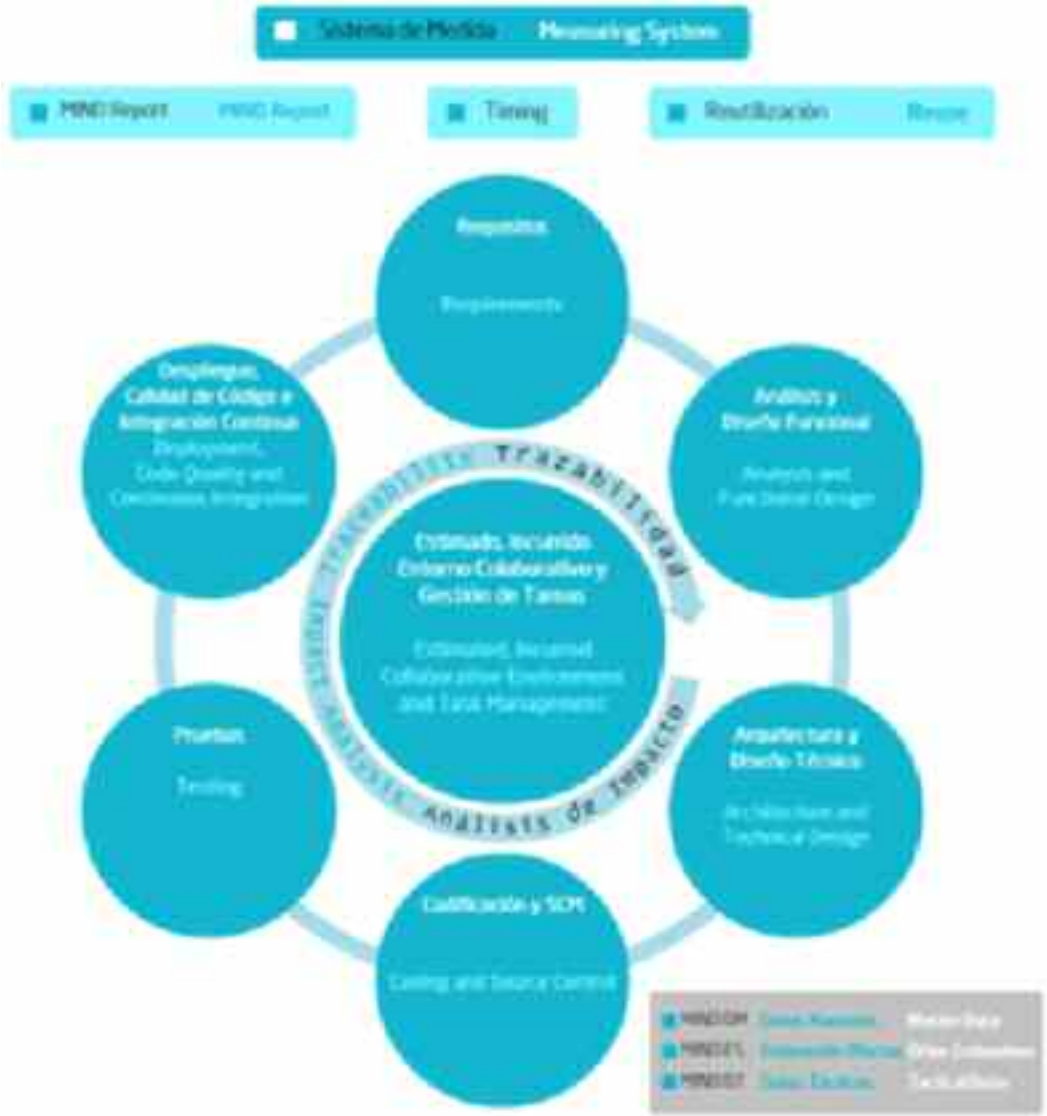


Figura 1. Suite MInd de Indra.

Entre las principales ventajas de MInd se encuentra la incorporación de un cuadro de mando que ofrece una visión global de todos los proyectos y servicios, permitiendo conocer el tamaño de las operaciones en curso y su evolución. De esta forma se facilita un seguimiento y control centralizado con independencia de que la

ubicación de la producción esté distribuida en diferentes países.

Por otro lado MInd proporciona mecanismos de de automatización en la recogida de métricas y KPIs, y un modelo de estimación competitivo. Aprovechamos nuestra experiencia en los niveles 4 y 5 de CMMI (Gestión Cuantitativa y Alta Madurez) utilizando técnicas estadísticas para el control de los errores software mediante gráficos de control.

Como puede verse, la Suite MInd incorpora ya capacidades relacionadas con los retos y oportunidades analizados. A continuación vamos exponer una serie de propuestas tecnológicas concretas, con las que trataremos de profundizar en la construcción de mecanismos para dar respuesta a dichos retos y oportunidades.

3.1. Big Data

Las incorporación en el sector de la Ingeniería del Software de las posibilidades que ofrecen las tecnologías relacionadas con el Big Data permitirían analizar la información heterogénea que se genera de forma continua en las distintas herramientas del ciclo de vida de Ingeniería del Software en entornos de desarrollo global multifactoría (con ubicaciones distribuidas geográficamente por todo el mundo) con el fin de poder realizar previsiones que permitan anticiparse a distintas situaciones para optimizar la toma de decisiones relacionadas con todas las fases de los proyectos, mejorando ostensiblemente la productividad y calidad de los productos y servicios desarrollados de acuerdo con el estado real actual y futuro de los proyectos, y con las necesidades de los clientes y usuarios.

Para ello, nuestra aproximación se centra en los siguientes aspectos:

- Extracción de "dark data" en factorías de software, con el propósito de obtener información importante sobre los productos, procesos y proyectos (y su calidad) a partir de la gran cantidad que existe hoy en día de datos no identificados ni aprovechados en el análisis de datos en las factorías de software.
- Construcción de mecanismos de control de la calidad de datos de Ingeniería del Software que permita asegurar la calidad de las colecciones de datos de los proyectos que entran en la fase de preprocesamiento de los algoritmos de análisis de Big Data.
- Adaptación de técnicas y herramientas analíticas, de forma que se posibilite la realización de predicciones y la optimización de la toma de decisiones en entornos de desarrollo global de software. Estas técnicas supondrían nuevas capacidades para la toma de decisiones en la gestión de proyectos software, realizando en base a la información del proyecto (y de proyectos precedentes) predicciones de costes, esfuerzos y calidad, posibles errores (con la correspondiente generación de casos de prueba asociados), tiempo de mantenimiento, desviaciones respecto a los objetivos, etc. También asistiría en la mejora de los procesos (ej. CMMI) mediante la predictibilidad del rendimiento de los mismos, en el enfoque de gestión y en la mejora del rendimiento de la organización. De esta forma también se permitiría dimensionar dinámicamente los proyectos a partir del trazado de los requisitos con la situación real de cada fase del proceso
- Construcción de herramientas analíticas para el análisis de la congruencia sociotécnica en desarrollo global de software, lo cual permitirá optimizar la organización y gestión de proyectos y factorías de software.
- Construcción de cuadros de mando que presenten información analítica en tiempo real.
- Extracción de patrones de comportamiento en el uso de las herramientas y monitorización continua de este uso. De esta forma se permitiría, por un lado monitorizar el grado de adecuación del uso que se hace de las herramientas, detectando desviaciones en el adecuado cumplimiento del proceso de Ingeniería del Software por parte de los equipos de los distintos proyectos. Por otro lado, se podrían detectar mejoras a implantar en las herramientas actuales y el grado de adaptación de los profesionales a las nuevas herramientas y metodologías.

Además, en relación a estas tecnologías, pensamos que la Ingeniería del Software debería dar respuesta a las dificultades que existen hoy en día a la hora de desarrollar aplicaciones Big Data, en las que no se disponen de mecanismos de automatización y los desarrollos son costosos en cuanto a tiempo y nivel de especialización requerida, realizándose de forma ad-hoc para cada caso concreto. Por ello pensamos que dentro de la adopción del Big Data en el sector de la Ingeniería del Software se debería incluir el poder proporcionar entornos de desarrollo para aplicaciones Big Data que automaticen la creación de las mismas. Para ello, nuestra propuesta tecnológica se basa en las Líneas de Producto Software (LPS), facilitando la incorporación de funcionalidades a las mismas mediante la integración automatizada de componentes reutilizables. Dichos componentes, serían soportados por distintas tecnologías y empaquetados como cajas negras parametrizables e integrables en las nuevas aplicaciones Big Data mediante API's bien definidas, formando un auténtico toolkit o librería de componentes que podrían ser usados de modo automatizado mediante el entorno LPS (que los parametrizaría siguiendo las especificaciones del usuario), o de forma independiente por programadores convencionales.

3.2. Tecnologías colaborativas

Si bien, la incorporación de herramientas sociales en todos los sectores es un hecho asumido, la Ingeniería del Software en la actualidad no explota de forma automatizada estas nuevas vías de comunicación social para la generación, compartición y sobre todo reúso de conocimiento, desaprovechando numerosas oportunidades presentes en un campo en el que las relaciones entre personas tienen un alto impacto en la generación de conocimiento y en la obtención de resultados. De ahí la necesidad de socializar y potenciar las relaciones entre miembros de un equipo de trabajo, automatizando la utilización del conocimiento generado en las interacciones entre los mismos. Las redes sociales, las plataformas de comunicación y, en general, los nuevos canales de comunicación pueden servir para mejorar la integración de equipos humanos y, por extensión, la calidad y eficacia de los productos software generados. Sería necesario centrarse en la extracción y gestión del conocimiento, ya que este tipo de redes implican un volcado de gran cantidad de conocimiento que no es explotado a día de hoy, necesitando nuevas tecnologías capacitadoras en este campo.

Nuestro enfoque tecnológico se centra, por lo tanto, en el desarrollo de técnicas de Listening Platform para la extracción de conocimiento en el espacio social, identificando temáticas (tecnológicas, metodológicas, etc.), necesidades y opiniones, relaciones entre usuarios, emisión de sugerencias automáticas, creación de vínculos entre usuarios, detección y búsqueda de usuarios expertos en cada tema, etc. Además de tener en cuenta las particularidades de los mensajes en redes profesionales (generalmente largos, con mucha semántica, y muy entrelazados entre sí, y con documentación externa tanto dentro como fuera de la empresa).

Por otro lado, vemos un gran potencial en la aplicación estas técnicas avanzadas de análisis de redes sociales en la innovación aplicada a la Ingeniería del Software, sirviendo como catalizador para la implantación de una cultura innovadora. Nuestra propuesta se centra en proporcionar mecanismos que permitan gestionar el conocimiento generado a través de la interacción social en la generación de ideas. De esta forma ese conocimiento podrá estar disponible, para facilitar la generación y evaluación de las mismas, relacionándolas con contenidos externos (tecnológicos, de mercado, etc.) e internos (productos, proyectos, estrategia). En este caso se haría necesario también incorporar análisis de terminología, polaridad (valoración sobre las ideas o la opinión ante la participación de los usuarios: positiva o negativa), eliminación de ruido, detección de usuarios influyentes, etc.

3.3. Gestión flexible de equipos de Ingeniería del Software

En este ámbito, vemos necesario el desarrollo de tecnologías que permitan una asignación óptima de recursos a proyectos, incorporando además una estimación inteligente de los tiempos de desarrollo previstos y una medición precisa del conjunto de dimensiones que afectan a un proyecto, con capacidad predictiva y de adaptación ante evoluciones imprevistas, permitiendo la anticipación a las mismas a la hora de determinar cuáles son los profesionales idóneos entre los disponibles en la organización para llevar a cabo el proyecto, tanto a nivel cualitativo como cuantitativo.

Para ello nuestra propuesta tecnológica se focaliza en el desarrollo de mecanismos que posibiliten cubrir los siguientes aspectos:

- Incorporación e interpretación la información procedente de distintos sistemas corporativos y de las herramientas de Ingeniería del Software para su análisis.
- Representación y medición automatizada del talento de forma integral, usando como base un modelo objetivo de representación del talento, que además tenga en cuenta otros factores, como los rasgos de la personalidad o las relaciones personales entre profesionales.
- Clasificación de profesionales en base a sus competencias adquiridas en la organización, medición de su evolución en el tiempo y predicción de estados futuros mediante modelos de autoaprendizaje. Para llevar a cabo esta categorización se deben tener en cuenta de forma objetiva las aportaciones de cada profesional en base a criterios difícilmente objetivizables, como son la proactividad, creatividad, colaboración, etc.
- Poner especial atención en la reputación como factor determinante, garantizando la calidad de su medición y minimizando los riesgos de fraude.
- Categorización de proyectos, de forma integral e inteligente mediante consultas automáticas sobre datos internos de la compañía, de forma que se obtenga una representación formal de la misma, susceptible de ser procesada por máquinas, permitiendo visualizar los recursos disponibles en la organización, teniendo en cuenta restricciones de cada proyecto: número de personas necesarias para su ejecución, con qué perfiles, durante cuánto tiempo serían necesarias, etc.

Las tecnologías que estamos desarrollando para hacer posible estos objetivos se basan en:

- Extracción, almacenamiento y procesamiento sin supervisión (y en lenguaje natural) de grandes volúmenes de información heterogénea procedente de fuentes dispersas.
- Algoritmos de PLN (procesado de lenguaje natural) que supongan herramientas lingüísticas robustas, orientadas tanto al análisis del texto en dependencias sintácticas como al reconocimiento de entidades, y adaptables tanto a varias lenguas como a fuentes textuales heterogéneas.
- Técnicas de categorización y segmentación, de caracterización dinámica que permitan cuantificaciones evolutivas, de matching, de estimación y recomendación automáticas, técnicas predictivas; etc.

3.4. Gamificación

Para dar respuesta a los retos planteados en relación a la motivación de las personas en los equipos de Ingeniería del Software, tanto para su máxima implicación y aportación a los proyectos, como en su adaptación al cambio (tecnológico y metodológico), la aproximación de Indra se centra en el desarrollo de tecnologías que permitan el uso de la mecánica de jugabilidad en Ingeniería del Software, más concretamente en las herramientas del ciclo de vida del desarrollo. Estas tecnologías estarán enfocadas a permitir el máximo aprovechamiento de las ventajas que la gamificación puede aportar al proceso de Ingeniería del Software, y que posibiliten hacerlo desde un punto de vista integral, en lugar de gamificando funcionalidades aisladas.

De esta forma se podrán potenciar distintos aspectos como la estimación de requerimientos, la adquisición de buenas prácticas, el intercambio de conocimientos dentro del equipo, las pruebas, el mantenimiento, la gestión del proyecto, el uso de nuevas herramientas, la aplicación de nuevas metodologías, etc. mediante la motivación y el compromiso de los profesionales, la innovación, la colaboración y la interacción social, etc. Por lo tanto, a través de la gamificación tratamos de ayudar a dar respuesta a la mayoría de los restos planteados (satisfacción del cliente, productividad, agilidad, colaboración, etc.).

Para dar soporte al desarrollo y aplicación del concepto de gamificación en Ingeniería del Software vemos necesario el desarrollo de una serie de tecnologías habilitadoras. En concreto hemos detectado la necesidad de llevar a cabo los siguientes desarrollos tecnológicos, para proporcionar características avanzadas a la gamificación: técnicas multidispositivo; técnicas de Inteligencia artificial y estadística para el análisis del compor-

tamiento de los usuarios y el establecimiento de perfiles (profiling), el análisis de emociones y la personalización; técnicas para la construcción de diálogos flexibles y naturales; técnicas de GIS (Geographic Information Systems); técnicas de colaboración y socialización; técnicas para metodologías ALM informales, ágiles y sociales; etc.

A partir de estos aspectos estamos centrando nuestros esfuerzos en la gamifiación de las distintas herramientas del ciclo de vida del desarrollo del software, principalmente la toma de requisitos, la gestión de proyectos y la gestión (definición, ejecución, etc.) de pruebas.

A parte de estos aspectos, consideramos imprescindible desarrollar mecanismos que posibiliten la automatización de la gamificación de forma unificada y coherente de las distintas herramientas y entornos del ciclo de vida, a través de un framework que implemente las funcionalidades comunes y generales en las mecánicas de gamificación, además de la incorporación de características avanzadas, y automatización de las funcionalidades y contenidos incluir para cada usuario.

3.5. Serious Games

El concepto de Serious Games se basa en capitalizar el poder de atracción de los juegos para ayudar a las personas a aprender y entrenar, haciendo que se aprenda y disfrute del aprendizaje. En primer lugar, en general, las personas entendemos mejor a través de experiencias prácticas en lugar de teorías, y los juegos ofrecen esta experiencia. Además de eso, los juegos involucran y motivan a los jugadores. Los juegos presentan problemas bien ordenados, en los que la información se presenta al jugador bajo demanda y en el momento requerido, manteniendo un equilibrio correcto entre desafío, el aburrimiento y la frustración. Por otra parte, además de la motivación y la diversión, los juegos permiten repetitividad de las tareas, niveles de dificultad adaptativos, así como la posibilidad de informar de forma automática sobre el comportamiento del alumno, lo que permite extraer de estos resultados datos más amplios sobre el proceso de aprendizaje / formación.

No se debe confundir el concepto de juego serio con el de gamificación. La gamificación consiste en incorporar mecanismos de jugabilidad (competitividad, recompensas, desbloqueo de ítems, etc.) para motivar aspectos concretos a través de incentivos. Los juegos serios permiten el aprendizaje de procesos y tareas de forma transparente al estar en realidad usando un juego, es decir, jugando. Basándonos en esta diferencia de base pensamos que la aplicación de los dos conceptos simultáneamente multiplicaría la efectividad cada uno de ellos por separado para la consecución de los objetivos planteados.

En concreto, desde el punto de vista de los retos planteados, la propuesta tecnológica de Indra se centra en la aplicación del concepto de juegos serios para disminuir la curva de aprendizaje en nuevas metodologías y herramientas, no sólo venciendo la oposición al cambio en las empresas, sino proporcionando además la motivación necesaria para su adopción. Además se podría aplicar a mejorar el uso de las herramientas y procesos actuales, en base al refuerzo del aprendizaje adquirido, eliminando posibles malas prácticas arrastradas en el tiempo.

3.6. Interfaces de usuario avanzadas

En los últimos años hemos vivido la irrupción de nuevas formas de interacción con la tecnología, de la mano del campo de los videojuegos. Son muchas las simulaciones y pruebas de concepto que hemos podido ver, sin embargo, la adopción de nuevas formas de interacción con las tecnologías para uso "productivo" parece aún lejana, no sólo en el sector de la Ingeniería del Software. Supone un reto, por lo tanto, trasladar las ventajas que estos nuevos dispositivos proporcionan al desarrollo de proyectos software.

La propuesta tecnológica de Indra en este sentido se centra en incorporar las posibilidades que ofrecen estas nuevas formas de interacción a las aplicaciones de uso ordinario y, especialmente, empresarial. Esto implicaría una nueva forma de concebir la interacción de usuario y una evolución tecnológica en el proceso de producción de software.

Para ello pensamos que supondría ventajas competitivas (permitiendo adelantarse a una necesidad futura evidente), el poder proporcionar un entorno de desarrollo para aplicaciones de interacción natural de forma aislada o combinadas con los mecanismos tradicionales de interacción. Sería un entorno gráfico que permitiría incluir acciones de interacción natural en aplicaciones de ámbito generalista (ej. aplicaciones web, aplicaciones de escritorio, etc.) que no existen en la actualidad de forma generalizada.

Por otro lado, la incorporación de estos desarrollos tecnológicos en las propias herramientas de Ingeniería del Software proporcionaría incrementos en la productividad al facilitar el uso de las mismas, además de la motivación de los profesionales a la hora de desarrollar su trabajo y de seguir correctamente el proceso de Ingeniería del Software, a la vez que se facilitaría la adaptación al uso de nuevas herramientas debido a la reducción del impacto que supondría una interacción natural en esta adaptación, proporcionando así una motivación adicional.

3.7. DevOps

Como hemos visto anteriormente, la incorporación efectiva de la agilidad en todo el proceso de Ingeniería del Software supondría una oportunidad para las empresas que consigan llevarlo a cabo de forma efectiva. Nuestra respuesta tecnológica a esta oportunidad se centra en el uso de las tecnologías descritas en los puntos anteriores.

A los retos planteados en este sentido anteriormente se une la dificultad que supone que las metodologías ágiles en realidad no son metodologías propiamente dichas, sino guías, "frameworks" o conjuntos de buenas prácticas que tratan de complementar los principios del "manifiesto ágil". Esto provoca que cada empresa (incluso cada equipo) implemente la agilidad de forma distinta. Esta dificultad se hace más pronunciada en el caso de DevOps, donde la agilidad va más allá del proceso de desarrollo, añadiendo también la producción, formando una única actividad que se realiza de forma continua.

En este sentido, en el desarrollo de la Suite MInd se ha aplicado la metodología de DevOps, a partir de la incorporación de herramientas habilitadoras para la integración y entrega continuas, y realizando el desarrollo y producción de forma conjunta y continua por parte un único equipo.

Actualmente los pasos que estamos siguiendo para la asimilación completa de la "metodología" DevOps en el proceso de Ingeniería del Software se centran en la mejora de las herramientas habilitadoras, y en trasladar las lecciones aprendidas y las mejores prácticas detectadas en el desarrollo de la Suite MInd a los proyectos (de tipo producto y servicio) que se desarrollen utilizando dicha suite.

4. Conclusiones

En este artículo hemos visto cómo nuevas tecnologías que se están aplicando en diversos ámbitos lúdicos y productivos se pueden adaptar y aplicar para dar respuesta a los principales retos existentes en el sector de la Ingeniería del Software.

Hemos visto cómo los distintos retos y oportunidades planteados están estrechamente relacionados entre sí, y cómo, de la misma forma, la incorporación de cada una de estas tecnologías pueden dar respuesta a uno o varios de forma simultánea. En este sentido hemos expuesto las propuestas tecnológicas en las que Indra está trabajando para proporcionar el soporte tecnológico necesario para cubrir estos aspectos. También hemos presentado la Suite MInd y sus principales características, la cual supone el elemento vertebrador de las propuestas tecnológicas planteadas.

Como conclusión principal queremos destacar que la Ingeniería del Software es un sector que necesita incorporar el dinamismo y agilidad tanto en la adopción efectiva de nuevas metodologías, como tecnologías y herramientas, para dar respuesta a los retos planteados en la actualidad y en el futuro inmediato, que pueden ser vistos como importantes oportunidades competitivas.

Sánchez Belmonte, G., García Moreno, C., y Hernández González, Y. (2014). Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 24-36. Consultado el [dd/mm/aaaa] en www.ijisebc.com

Cómo citar este artículo / How to cite this paper

Sánchez Belmonte, G., García Moreno, C., y Hernández González, Y. (2014). Propuesta tecnológica de Indra para afrontar los retos inmediatos de la Ingeniería del Software. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 24-36. Consultado el [dd/mm/aaaa] en www.ijisebc.com

Software Development by Steelmood

El Desarrollo de Software por Steelmood

Fernando Ruiz-Falcó[1]

[1] Vicepresidente Steelmood

fernando.ruizfalco@steelmood.com

RESUMEN. Este artículo analiza el Desarrollo de Software en las empresas a través de la experiencia y los casos vividos por la empresa consultora, 'Steelmood'. Teniendo como guía los siguientes puntos: Lecciones aprendidas de otras industrias, La industria de fabricación de software a medida hoy, Las bases del Modelo Software Development by Steelmood, La dinámica de trabajo de un equipo en SDS, Beneficios del Modelo SDS, Barreras para implantar el Modelo SDS y El proceso de transformación y escalado.
Llegando a la conclusión de que estamos cerca de un importante cambio de paradigma en la industria del desarrollo de software a medida y presentando su propia propuesta para ayudar a las empresas a dar el salto a este nuevo paradigma cuanto antes.

ABSTRACT. This paper analyzes Software Development in companies through experience and cases experienced by the consulting firm, 'SteelMood'. Taking as a guide the following: Lessons learned from other industries, Manufacturing industry custom software today, Bases of the Model Software Development by SteelMood, Work dynamic of a team in SDS, SDS Model Benefits, Barriers to implement SDS Model and Process of transformation and scaling.
Concluding that companies are near a major paradigm shift in software development industry as and presenting their proposal to help companies make the leap to this new paradigm as soon as possible.

PALABRAS CLAVE: Desarrollo de Software, Tendencia del Desarrollo de Software, SDS, Proceso de transformación y escalado, Consultoras, Ingeniería de Software.

KEYWORDS: Software Development, Software Development Trends, SDS, Process transformation and scaling, Consulting, Software Engineering.

1. Introducción

"Every business is a software business, and every business can profit from improved software process". Watts Humphrey

Con la apertura del CAIS (Centro Avanzado de Ingeniería de Software) de Steelmood, estos últimos años, hemos tenido ocasión de hablar en profundidad con los responsables de desarrollo de software de las principales compañías españolas. Esto nos ha permitido tener una idea de los problemas reales desde el punto de vista del "gran consumidor", es decir, compañías que invierten varios cientos de millones de euros al año en desarrollar software a medida de sus necesidades. Este mercado tiene un tamaño de unos tres mil millones de euros en España y unos tres billones de dólares en el mundo.

Como síntesis de estas entrevistas, los tres principales retos a los que se enfrenta esta industria son:

- Requerimientos: cómo realizar unas especificaciones funcionales que recojan las necesidades reales del negocio y los usuarios del Sistema.
- Construcción: cómo construir el producto software en el tiempo y coste previsto y con el nivel de calidad esperado.
- Rendimiento de la máquina: cómo mejorar la gestión y el rendimiento de la máquina productiva, esto es, el retorno de la inversión realizada (ROI).

Hace treinta años, estos problemas eran los mismos, sobre todo los dos primeros. Cuando teníamos un problema de este tipo en algún proyecto, razonábamos con el cliente que ello era debido a la juventud de la industria de desarrollo y a la falta de experiencia realizando trabajos similares. Desde luego, ninguno de estos argumentos sigue siendo válido hoy: ha pasado mucho tiempo (la industria ya no es tan joven) y, además, la gran mayoría de las funciones que se están codificando hoy, ya han sido desarrolladas. Aún así, nos enfrentamos al mismo tipo de problemas porque no hemos sabido madurar suficientemente el proceso productivo como sí han hecho, de forma espectacular otras industrias, durante este periodo de tiempo.

Es increíble lo poco evolucionado que está el paradigma actual de gestión de la producción de desarrollo de software. La única explicación sencilla que podría explicar tan llamativo fenómeno es que mentes brillantes, tanto en los propios clientes como en sus proveedores, han estado tan absorbidas por el poder (y evolución) de la tecnología, que, a diferencia de lo que ha ocurrido en otras industrias, no han tenido tiempo suficiente para detenerse a reflexionar sobre el proceso productivo en sí mismo.

Repasemos brevemente lo que ha ocurrido en estas últimas décadas en las industrias que fabrican en el mundo físico (automoción, electrónica de consumo, aeroespacial, maquinaria pesada, etc.). En primer lugar, en todas ellas se ha producido una muy significativa evolución en mejora de la calidad, costes y plazos de entrega de los productos que fabrican. Pensemos, por ejemplo, en un automóvil; si comparamos su coste y calidad hace treinta años con los actuales, llegamos a la conclusión de que hoy son mucho mejores (consumo, seguridad, fiabilidad, prestaciones, etc.) y a un coste muy inferior. Este mismo razonamiento es válido para otras muchas industrias manufactureras: ordenadores, electrodomésticos, aviones, maquinaria pesada, etc.

En segundo lugar, todas estas industrias, por puro efecto darwinista, han expulsado del mercado a quienes no han sabido evolucionar (o revolucionar) sus procesos productivos para encontrar mejoras tan significativas. Por seguir con el ejemplo del automóvil, si un fabricante como Toyota rompe las reglas de la industria ofreciendo ventajas en calidad y costes a sus clientes, o los demás reaccionan, o simplemente desaparecen, ya que ¿Quién quiere comprar un coche peor o más caro?

En tercer lugar, el proceso de mejora pasa por saltos disruptivos que no son sencillos de afrontar pues implican verdaderos cambios de paradigma. Por supuesto que ni a Toyota ni a Motorola, por citar sólo dos ejemplos, les ha sido sencillo mejorar tan significativamente sus métodos productivos. Es imprescindible la visión y el com-

promiso de la alta dirección, la disciplina e insistencia en el tiempo y la involucración de todos los actores afectados (operarios, supervisores y gestores). La tarea no es fácil, pero la motivación, clara. Quien no proceda de ese modo, desaparecerá.

2. Lecciones aprendidas de otras industrias

En todos los casos que estamos citando de otras industrias, hay un hecho fundamental que ha sido condición necesaria para realizar saltos significativos de mejora: dedicar una parte importante del esfuerzo intelectual y operativo a reflexionar y mejorar el proceso productivo en sí mismo, más allá de que nos dediquemos a producir automóviles o lavadoras.

Evidentemente, otra parte del esfuerzo seguirá estando en el diseño y fabricación del propio producto, pero esto no es el todo, sino solo una parte y, en algunos casos, desde luego que no es la más significativa.

Dicho de otra forma: en una fábrica de coches trabajan ingenieros expertos en automóviles que conocen perfectamente sus componentes (motores, carrocerías, electrónica, materiales, etc.) pero también hay ingenieros que apenas saben nada de lo anterior y son expertos en los propios procesos productivos (compras, ensamblaje, LEAN, KANBAN, Six-Sigma, etc.). De hecho, cuantitativamente, en la industria del automóvil de hoy, trabajan muchos más de los segundos (ingenieros de procesos) que de los primeros (ingenieros de producto).

En realidad, la lección aprendida que obtenemos es que encontrar el balance adecuado entre ingenieros de proceso e ingenieros de producto para cada industria en cada momento de mercado, es clave para el éxito. Es decir, es esencial acertar con las dosis necesarias de esfuerzo en producto y esfuerzo en proceso necesarias para satisfacer las expectativas del cliente.

La segunda lección es que es imprescindible la visión, determinación y compromiso de la alta dirección en conseguir mejoras importantes y, además, disponer de un modelo estructurado para implantarlas. Sin la concurrencia de ambas, únicamente es posible tener éxito por casualidad. Y cuanto menos cosas relativas a nuestra propia subsistencia dejemos en manos de la diosa Fortuna, mejor.

La visión, determinación y compromiso es imprescindible como en cualquier caso que implique un cambio de paradigma, esto es, que se proponga una nueva utopía más allá de lo razonable. Si John Fitzgerald Kennedy no hubiera anunciado en mayo de 1.961 su visión de que un americano pisaría la Luna antes de terminar la década y no hubiera puesto toda su determinación para hacerlo, Neil Amstrong probablemente no podría haber dicho su famosa frase "un pequeño paso para el hombre pero un gran salto para la humanidad" al pisar el suelo de nuestro satélite el 21 de julio de 1.969. ¡Únicamente ocho años después!

Pero es también necesario que esta energía sea canalizada por un modelo que facilite su puesta en práctica de forma estructurada, sistemática, eficaz y eficiente.

La tercera y última lección es que el gran salto se inicia poniendo unas pocas personas a pensar en profundidad sobre la ingeniería del proceso productivo, pero el impulso fuerte realmente se produce cuando la mentalidad y actividades concretas relativas a la mejora de la calidad de los procesos se extienden por todas las personas que participan en el proceso productivo.

Veamos como resumen esta evolución los profesores Jay Rao y Fran Chuán:

"En la posguerra de la Segunda Guerra Mundial, los profesores Demin y Juran, enseñaron técnicas estadísticas y de gestión de la calidad a empresas japonesas. El centro de atención en ese momento era la inspección, la reacción a los defectos y el control de calidad. En los años 70 la responsabilidad por la calidad era mucho mas funcional y estaba limitada a unos pocos trabajadores de la empresa (…) En 1980 la norteameri-

cana Naval Air Systems Command introdujo el término TQM (Total Quality Management). Este nuevo concepto incluía la prevención de defectos y hacía a todos los trabajadores responsables de la calidad. El hecho de que fueran muchos los proveedores que trabajaban con Naval Air Systems Command hizo que muchas compañías empezaran a conocer y adoptar la gestión de la calidad total, y que esta se propagara como el fuego (…) El SIx-Sigma ha sido la última evolución en el movimiento de la calidad. Aunque sigue manteniendo algunos principios estadísticos y procesos fundamentales de los métodos anteriores, se trata de un método más deliverado, sistemático y exhaustivo que aquellos, lo cual representa un gran paso hacia delante. Desarrollado a mediados de los años ochenta por Motorola, Six-Sigma salta a la fama cuando Motorola gana en 1988 el premio nacional a la calidad Malcolm Baldrige, siendo la primera vez que se presentaba." (2012: 16)

3. La industria de fabricación de software a medida hoy

Es evidente que la industria de desarrollo de software a medida está muy atrasada en el proceso evolutivo que acabamos de describir. Todavía no ha terminado de dar el primer paso para incorporar el control estadístico de los procesos y filtros de calidad que propuso Deming y, mucho menos, para involucrar a todas las personas en el compromiso de mejorar en costes y calidad.

Hoy, el estado del arte de la industria, se limita a conformarse con adoptar etiquetas formales (ISO, CMMI, etc.) más centradas en el qué hay que hacer para obtener una certificación que cómo se mejora realmente, sin abordar los problemas reales desde los propios equipos de trabajo.

Es decir, se trata, en el mejor de los casos, de pequeños grupos dedicados a la mejora de procesos luchando contra todos los elementos (clientes, equipos de trabajo, etc.), donde es escaso el compromiso de la alta dirección y el de los propios equipos de trabajo. Los modelos de trabajo aportan mejoras parciales pero no son en absoluto una aproximación holística que involucre a toda la organización de modo sistemático.

De esta manera, más del 90% del esfuerzo de los equipos de trabajo se centran en la ingeniería de producto, cubriendo su dimensión tecnológica y funcional, pero infraponderando estrepitosamente la dimensión de mejorar el proceso productivo en sí mismo.

Una imagen sencilla de la situación general es que los equipos tienen tanta leña que cortar, que no creen tener tiempo para afilar el hacha, cuando, visto desde fuera, es evidente lo rentable de la inversión en el afilado.

Este escaso avance en lo esencial, no significa que durante estos años, tanto clientes como proveedores, no hayan realizado ninguna evolución. Es conveniente señalar tres significativos:

Tecnología. Es evidente el avance que se ha producido en este campo. Evolución en lenguajes de programación, hardware, sistemas operativos, redes de comunicación, dispositivos, gestores de bases de datos, etc.

De hecho, asumir el esfuerzo de evolución en este campo, es la razón fundamental de la poca evolución en el afilado del hacha: las nuevas tecnologías eran tan espectaculares que se convertían en el objetivo mismo, perdiendo la perspectiva de que, en general, la tecnología es un medio al servicio del negocio más que un objetivo en sí misma.

Modelos. Aunque les falte por alcanzar un uso masivo y no sean completos para abordar todas las componentes de forma integrada, sí se han producido avances en algunos campos, generalmente por el lado académico. A continuación, se citan los más significativos que hemos introducido como componentes en nuestro modelo de desarrollo SDS: TOGAF para Arquitectura Empresarial, BABOK (del International Institute for Business Analyst) para gestión de requerimientos, Personal Software Process (PSP) y Team Software Process (TSP), del Software Engineering Institute y Carnagie Mellon University, para la construcción, conceptos TSP

y AGILE para la organización de equipos y LEAN (Lean Advancement Initiative, Massachusetts Institute of Technology) para el análisis y diseño de procesos.

Maduración del proceso de compra-venta. La presión competitiva del mercado, al carecer de un modelo que permita gestionar el coste total del desarrollo de software, se ha centrado en la disminución del coste unitario, es decir, el precio por hora de la mano de obra interviniente. De esta manera se han firmado estos años muchos acuerdos marco o contratos de externalización de desarrollo y mantenimiento con fuerte presión en el coste horario de la mano de obra como principal parámetro de negociación (tanto con proveedores locales como globales). Esta situación fue bien aprovechada por compañías indias, ofreciendo mano de obra bien cualificada a coste horario sensiblemente inferior.

Consecuentemente, este recorrido ha llegado al extremo del péndulo el cual está a punto de dar la vuelta y empezar a moverse en dirección contraria. Los clientes se han dado cuenta de que han conseguido disminuir sensiblemente el coste por día trabajado, pero, al mismo tiempo, ha aumentado el coste total, que es su problema real. Y, además, en los casos de negociación más radical en precio, la calidad ha empeorado. Más adelante, analizaremos en mayor profundidad las razones y vías de solución de esta paradoja, pues entendemos que va a ser (o está siendo ya) una de las fuerzas principales que impulsarán el cambio de paradigma en la industria de desarrollo de software a medida.

Por todo ello, es un hecho ya en el mercado que grandes compradores de Off-Shore en países como Reino Unido y Holanda, por ejemplo, y que firmaron grandes contratos de externalización con compañías basadas en India, no los están renovando y están buscando otras soluciones (In-site o Near Shore). En España, aunque la proporción de mercado con India es menor, el fenómeno es el mismo, con proveedores locales o globales, con fuerte presión en coste unitario.

En resumen, por su propia experiencia y a base de disgustos, los clientes hoy ya han aprendido que el coste unitario de la mano de obra es uno de los parámetros a tener en cuenta a la hora de contratar con un proveedor, pero no el único si queremos mejorar efectivamente el problema real para el cliente que no es otro que mejorar la rentabilidad de su inversión en software.

Para poder abordar el problema real es necesario introducir otras dos dimensiones nuevas a la ecuación: la productividad y la calidad del producto entregado. Ilustremos esto con un ejemplo.

En efecto, es más barato pagar, digamos, 200 € por día por una persona (o proveedor) (A) con una productividad de 30 líneas de código por hora que 100 € por otra (B) con una productividad de 10 líneas de código por hora. En el primer caso la línea de código cuesta 0,83 € y en el segundo 1,25 €. El coste de producir una línea de código (o un punto función) no refleja completamente el coste total, pero ya es una aproximación mucho más cercana a la realidad que considerar únicamente el coste horario.

En este ejemplo tan sencillo (y tan frecuente en estos últimos años) vemos como una disminución del 50% en coste unitario puede producir un resultado de un aumento del 50% del coste total.

Para incorporar al ejemplo la calidad debemos comenzar por realizar una breve reflexión sobre el asunto, comenzando desde la raíz.

Dado que en el proceso intervenimos humanos, vamos a cometer errores e insertarlos al producto. Aquí el software se comporta como cualquier otro proceso productivo y nos sirve todo lo aprendido en otras industrias (TQM, Six-Sigma, etc): cuanto antes se corrija el error en la cadena productiva, más barato será hacerlo.

En el caso del software, en palabras de Manies y Nikual:

"Es una realidad que entre el 40% y el 60% de los errores y defectos del software son el resultado de una

pobre gestión y definición de requisitos. En palabras simples, esto significa que aproximadamente la mitad de los problemas encontrados se podrían haber evitado, simplemente, con dejar claro desde el principio, lo que el cliente espera del proyecto."(2011: 26)

La mayoría de las veces los errores en requerimientos no se detectan hasta la fase de pruebas, lo que implica enormes decepciones, retrasos y sobre coste por retrabajo.

En este punto conviene introducir el "Failure Appraisal Ratio" pues es el indicador que nos ayuda a balancear el esfuerzo al detectar errores en etapas tempranas optimizando el resultado final. Se define como la división entre el tiempo total dedicado a la prevención de errores (filtros de calidad) y el tiempo total dedicado a pruebas. De acuerdo con la literatura actual, el valor óptimo para software de gestión está alrededor de 2, es decir, cuando dedicamos el doble del tiempo en filtros intermedios que en pruebas.

¿Cuánto es este ratio en su organización? Si es algún valor entre cero y uno, como es habitual hoy, tiene usted ante sí una enorme oportunidad de mejora.

Ahora estamos en disposición de incorporar el asunto de la calidad a nuestro pequeño ejemplo. La pregunta fundamental es ¿Cuánto cuesta que se produzca un error después de la liberación del producto? Este coste tiene dos componentes:

- El coste económico y en disminución de la satisfacción del cliente, derivado de la suspensión del servicio interrumpido por el fallo de la aplicación. Puede ser enorme pero no es sencillo de estimar consensuadamente.
- El coste de los técnicos que analizan y reparan el error. Aunque puede ser mucho menos significativo que el anterior, es fácil de calcular y es el que suele incorporarse a los modelos como beneficio cuantificable (y dejando el anterior como no cuantificable).

Evidentemente, el tiempo que se tarda en arreglar un error, una vez liberado el producto, es un parámetro sensible a la aplicación e instalación en concreto. Si usted dispone de información de su propio caso, utilícela; pero mientras dispone de su propia serie histórica, puede considerar las 20 horas que calcula Standish Group como el tiempo medio necesario para corregir un defecto después de la liberación del producto.

Por lo tanto, para completar en el ejemplo el coste total de cada caso para el cliente, necesitamos incorporar la tasa de defectos que incorpora cada uno. Supongamos que el primero tiene un coste de 200 € por día, una productividad de 30 líneas de código por hora e incorpora un defecto por cada mil líneas de código y que el segundo cuesta 100 € por jornada, produce 10 líneas de código por hora e incorpora una tasa de 10 defectos por cada mil líneas de código.

Según lo visto hasta ahora, en el primer caso el coste de producir mil líneas de código es 0,83*1000 = 830 € y en el segundo 1,25*1000 = 1.250 €. Ahora podemos saber además, lo que nos costará mantener el producto en cada caso.

En el primer caso tendremos un defecto que nos costará 20 horas (2,5 jornadas) arreglarlo, es decir 2,5 jornadas * 200 € = 500 €. El coste total del sistema en este caso es 830 para su desarrollo y 500 para su mantenimiento, es decir 1.350 €.

En el segundo caso, hemos de esperar 10 defectos que nos costará arreglar 20 horas cada uno, es decir 10* 2,5 jornadas * 100 € = 2.500 €. El coste total del sistema en este caso es 1.250 € para su desarrollo y 2.500 € para su mantenimiento, es decir 3.750 €.

En resumen, en el ejemplo vemos un caso en el que se disminuye el coste de la mano de obra a la mitad y aumenta un 177% el coste total y aunque parezca disparatado, no lo es tanto como algunos ejemplos reales

de grandes compañías estos últimos años.

De este análisis de la situación actual de la industria del desarrollo de software a medida, conviene extraer dos conclusiones importantes. En primer lugar, para empezar a gestionar la totalidad del asunto, los clientes necesitan incorporar parámetros de productividad y calidad en sus procesos de compra y de gestión. Y la segunda es que este hecho lo han descubierto por sí mismos, o están cerca de hacerlo, basados en su propia experiencia de estos últimos años. Los proveedores, en general, estábamos demasiado ocupados incorporando nuevas tecnologías y disminuyendo nuestros costes unitarios de la mano de obra.

Esta insatisfacción con el modelo actual acumula considerables dosis de energía para impulsar el cambio, cuando aparezca la palanca adecuada para canalizarla.

4. Las bases del Modelo Software Development by Steelmood

Hemos llamado al modelo Software Development by Steelmood para señalar que, aunque el modelo es característicamente nuestro, lo hemos construido utilizando como componentes otros modelos y estándares internacionales en diversos campos, como los citados anteriormente. Hemos trasladado al desarrollo de software de gestión a medida, alguna de las ideas básicas que se han producido en la ingeniería de procesos de las industrias manufactureras anteriormente analizadas.

Veamos primero sus principales características y después sus bases conceptuales.

La primera característica de SDS es que es un modelo disruptivo, esto es, su puesta en marcha implica un cambio de paradigma en la organización del cliente. Este cambio será mayor o menor según el nivel de madurez del cliente, pero es siempre significativo pues implica un cambio profundo en la forma de realizar las actividades por los propios equipos de trabajo de desarrollo.

La segunda característica de SDS es que es un modelo holístico. Su objetivo es abordar todos los asuntos relacionados con el desarrollo de software a medida, con aportaciones de diversas disciplinas para poder abarcar de forma integrada todos los aspectos del asunto.

La tercera característica de SDS es que es un modelo pragmático. Su finalidad es la mejora de los resultados, es decir, del retorno de la inversión de nuestros clientes en desarrollo de software a medida.

La primera base conceptual del modelo es introducir procesos sistemáticos y cuantitativos de mejora en tres capas:

- La persona individual. Se trata de ayudar a cada ingeniero de software a que mejore su propio desempeño personal. Para ello, nos basamos en las ideas extraídas de PSP (Personal Software Process es Service Mark de la Universidad de Carnegie Mellon), del coaching ejecutivo y de la gestión del cambio personal de Goullart y Kelly.
- El equipo de trabajo. El propósito es mejorar el desempeño de cada conjunto de ingenieros de software trabajando en un proyecto común. Para ello nos basamos en las ideas extraídas de TSP (Team Software Process Process es Service Mark de la Universidad de Carnegie Mellon), del coaching ejecutivo y de la gestión del cambio en un equipo natural de trabajo de Goullart y Kelly. Y la gestión de la calidad percibida de Noriaki Kano.
- Procesos organizacionales. Para promover la esbeltez del flujo de procesos corporativos del cliente relacionados con el desarrollo de software, con los que, de una u otra forma, han de interactuar (y respetar) los equipos de trabajo de desarrollo. Nos basamos en ideas extraídas de LEAN (promovida por el LAI, Lean Advancement Initiative, MIT) y las 4R´s del cambio y Arquitectura de la Movilización de Goullart y Kelly.

El eje central de la mejora del desempeño individual y colectivo es enseñar a registrar con granularidad y rigor la información sobre el tiempo invertido en construir cada artefacto, el tamaño del mismo y los defectos que insertamos. Esto significa:

Tiempo: Se trata de registrar el tiempo realmente invertido en desarrollar el artefacto. La suma de estos tiempos no es ocho horas al día, ni cuarenta horas a la semana. No se trata de repartir el tiempo que trabajamos entre distintos criterios de actividad o imputación, que es lo que habitualmente se hace en la actualidad. La lógica es distinta y el objetivo también y es clave explicar esto bien, en los cursos de inmersión, a los ingenieros neófitos en el modelo, para que puedan aplicarlo correctamente. No se trata de justificar o repartir; se trata, simplemente, de registrar el tiempo verdaderamente invertido en cada artefacto, cualquiera que éste sea. La recogida de esta información se puede realizar de forma automática en la propia plataforma de trabajo.

Tamaño: Se registra el tamaño de los artefactos producidos. Evidentemente, es necesario definir una métrica de tamaño para cada tipo de producto. Nuestro modelo es agnóstico respecto a la elección de la métrica concreta, no vemos una ventaja significativa por utilizar una en vez de otra, sino que, lo único verdaderamente importante, es ser coherente con la métrica elegida y utilizar siempre la misma para poder comparar. Así, por ejemplo, para medir el tamaño de un programa, nos parece válido utilizar número de líneas de código, puntos función o casos de uso; para análisis funcional, número de páginas, líneas o palabras, etc.

Defectos: Para los defectos, es necesario definir una taxonomía de errores para cada tipo de artefacto: análisis funcional, diseño técnico, codificación, pruebas, etc. Para ello es importante adaptar adecuadamente el nivel de sofisticación de la clasificación propuesta en los manuales del modelo con el nivel de madurez de los equipos involucrados. Se definen en el grado de detalle adecuado para la mejora de los resultados, aconsejando huir de definiciones excesivamente complejas en un principio, para ir evolucionando en la lista con su uso.

Granularidad y rigor estadístico: Los registros tienen una granularidad de horas, minutos y segundos. Algunos productos los desarrollamos en algunos segundos y, para otros, invertimos muchas horas. En ambos casos, lo mediremos con exactitud para que los datos históricos correlacionen estadísticamente. Así nos aseguramos que no estamos introduciendo ningún sesgo con el procedimiento de medida. Esto es fundamental por varias razones:

La primera es que si comenzamos a tener datos estadísticos sobre nuestro comportamiento, empieza a ser posible predecir el mismo con márgenes de errores estadísticos. Esto supone un enorme avance en términos de predictibilidad de las entregas y, en consecuencia, de time-to-market. También es clave para la estimación de carga de nuevos proyectos, SLA´s de productividad y calidad, a los que realmente nos podemos comprometer.

Y en segundo lugar, porque tenemos información de partida para construir un conjunto completo de métricas que facilitan la mejora de los individuos, la mejora de los equipos y las decisiones de los supervisores y directivos.

En la siguiente ilustración, proporcionamos la lista completa de métricas:

El segundo punto conceptual básico es establecer filtros de calidad inmediatamente después de cada paso del proceso productivo hasta equilibrar el Appaissal Failure Ratio. Con estos filtros de proceso evitamos arrastrar errores a lo largo del ciclo de vida. Cuanto antes removamos el error insertado, más barato será hacerlo, evitaremos retrabajos y mejoraremos la calidad del producto resultante.

En el diagrama siguiente, mostramos el mapa general de los pasos y artefactos en el ciclo de vida de un desarrollo de software. Obsérvese que, en cada paso del proceso, hay un filtro con los códigos R, I y V. La R significa realizar una revisión (el propio autor del producto revisa el mismo para corregir errores), I simboliza una inspección (otra persona distinta al autor, con similar cualificación profesional, inspecciona el producto para corregir errores). Y la V significa validación, donde es algún representante del cliente el que valida el producto, señalando sus posibles errores.

En este punto, conviene ver algún ejemplo. Por su serie histórica, sabemos que el Analista A escribe análisis funcionales con una productividad de 0,77 páginas por hora, insertando un defecto mayor y 20 menores cada 100 páginas. Además, el 80% de los defectos mayores que ha insertado históricamente el Sr. A son de completitud del requerimiento, el 10% son de consistencia y el otro 10% se reparte entre factibilidad y agregación; y el 60% de los defectos menores son de ortografía, el 20% de gramática y el otro 20% de contenido menor. Además, identifica la mitad de los defectos que inserta realizando inspecciones de su propio trabajo, con una productividad de revisión de 10 páginas por hora.

Naturalmente, el primero que dispone de toda esta información es el propio Sr. A, y con un poco de ayuda de un coach, le será muy útil como instrumento básico para la mejora de su propio trabajo. No olvidemos el principio básico de que podemos mejorar lo que somos capaces de medir.

Pero la información también está disponible para los demás miembros del equipo, de manera que si el Sr A reporta que ha escrito un funcional de 100 páginas en 20 jornadas de trabajo y que lo ha revisado durante 4 horas y ha corregido cero defectos mayores y cinco menores, el supervisor sabe inmediatamente que:

La productividad del Sr A en este trabajo (0,65 páginas por hora) ha sido algo inferior a su productividad media (O,77 páginas por hora). O este funcional es un poco más complejo de lo normal, o la motivación del Sr A está más baja de lo normal, o ha habido algún problema o complicación especial.

Sin embargo, la velocidad de revisión en este caso (25 páginas por hora), es 2,5 veces superior a su media (10 páginas por hora). Esto significa que tengo que esperar que aún queden defectos sin identificar en el funcional y puedo decidir que se realice otra inspección o revisión.

El tercer punto conceptual básico es disponer de una herramienta que facilite la imputación de datos y calcule y ponga accesibles, en sus diversos grados de agregación, todas las métricas.

Con la información de estas métricas y un poco de adiestramiento (un curso de una semana intensiva y un periodo de un año de coaching en el equipo de trabajo), se comienzan a tomar decisiones basadas en datos objetivos en los tres niveles mencionados anteriormente: individual, equipo de trabajo y organización en su conjunto.

Y el cuarto punto básico conceptual es medir también la calidad subjetiva, como parámetro básico de la relación contractual. Para ello, como hemos señalado anteriormente, utilizamos las ideas de calidad subjetiva de Noriaki Kano, incorporadas al procedimiento EAP (Expectations Aligment Process) definido por Steelmood:

Al comienzo del proyecto o servicio pedimos al cliente que nos explique sus expectativas o parámetros de satisfacción subjetiva para este proyecto o servicio. Así comprendemos los criterios de la calidad percibida definidos por el cliente para cada caso, pudiendo ponderar cada uno según importancia de cada criterio (de 0 a 10).

Al final del mismo, o en los periodos de revisión pactados, el cliente evalúa de 0 a 10 su satisfacción obtenida en cada uno de los criterios, explicando las razones de su puntuación, proporcionando feedback.

Si el resultado es inferior a 7, nos aplicamos una penalización y devolvemos al cliente el 20% del importe facturado.

El objetivo fundamental de este procedimiento es de alineamiento con el cliente. De esta manera, nos aseguramos que estamos haciendo lo que realmente es importante para él, aunque sea difícil de medir.

5. La dinámica de trabajo de un equipo en SDS

Acabamos de resumir las principales características y bases conceptuales del Modelo SDS. A continuación, describimos la dinámica de trabajo de un equipo SDS, que es la célula fundamental del Modelo.

Son equipos autodirigidos, que reparten entre sus miembros todos los roles administrativos y de control de procesos, y realizan lanzamientos, ciclos y postmortem, típicamente en iteraciones de unas seis. Obsérvese que todos estos conceptos son análogos y muy compatibles con metodologías AGILE tipo SCRUM, pero siempre con la consideración de que, en el Modelo SDS, las decisiones se toman basadas en métricas cuantitativas.

En resumen, SDS asegura que, de forma sistemática, los principales actores del aparato productivo (ingenieros de software y equipos de desarrollo) dediquen parte de su esfuerzo a reflexionar y mejorar su propio proceso productivo, facilitando a supervisores y directivos la toma de decisión más adecuada, basada en datos objetivos y medibles.

6. Beneficios del Modelo SDS

Si clasificamos los beneficios por su carácter financiero o no financiero (operativo), y, , a su vez, cuantificables o no cuantificables, el Modelo proporciona beneficios en los cuatro cuadrantes.

A continuación se facilita una lista de los principales beneficios generales que se pueden particularizar en un "Business Case" para cada caso, de acuerdo con sus propios objetivos y circunstancias:

- Ahorro significativo en el coste total del desarrollo (entre el 25% y el 50%, según el caso concreto). Este ahorro se basa en:
  - o Disminución drástica (alrededor del 80%) del coste del mantenimiento correctivo derivado de una mayor calidad del producto, en el momento de su liberación (alrededor de 0,4 defectos por cada mil líneas de código).
  - o Reducción a la mitad del tiempo dedicado a pruebas debido a la supresión de errores en los filtros previos.
  - o Reducción del coste del desarrollo propiamente dicho cercana al 20% por disminución de retrabajos y vueltas atrás en el proceso.
- Ahorro en suspensiones de servicios de negocio por disminución de errores en producción, deri-

vados de la mayor calidad del producto liberado.

- Mejor time-to-market por mejora de predictibilidad de las fecha de entrega.
- Mayor satisfacción de los usuarios por la mejor calidad de los productos producidos y mayor cumplimiento de compromisos de fechas de entrega.
- Mayor motivación y compromiso de los equipos de trabajo.

7. Barreras para implantar el Modelo SDS

Las principales barreras que nos hemos encontrado para implantar el Modelo son tres:

1. Decisión y compromiso del cliente con la mejora. Es la barrera más fuerte y más difícil de tratar y, como hemos señalado anteriormente, es un ingrediente absolutamente imprescindible

2. Resistencia al cambio de las personas y los equipos de trabajo. Al final, el Modelo altera la esencia misma de su trabajo al introducir rutinas y disciplinas nuevas. El tratamiento de esta barrera es mucho más sencillo de lo que aparenta y se basa en la transparencia, la información y la gestión de un cambio de paradigma. En los cursos de capacitación en el modelo, se enseña a los participantes que los primeros beneficiarios de la aplicación de modelo son ellos mismos, pues mejorará su calidad de vida y su desempeño profesional.

3. Aparente dificultad para escalar rápidamente el Modelo a grandes volúmenes (miles de personas). Al igual que la anterior, es más su apariencia que la realidad. Como desarrollamos en el epígrafe siguiente, es posible industrializar el proceso de transformación y escalar rápidamente su implantación en varias fases con alcances y riesgos acotados.

8. El proceso de transformación y escalado

La esencia del proceso de transformación consiste en que, al menos una célula de desarrollo (equipo de 3 a 20 personas), comience a trabajar con el método simultáneamente. Esto se puede conseguir de dos maneras distintas: capacitando en el Modelo SDS a un equipo de trabajo (propio o de otro proveedor) ó externalizando el trabajo al CAIS.

Ambos procedimientos tienen ventajas e inconvenientes que aconsejarán uno u otro camino para cada caso. Lo verdaderamente importante es producir la personalización del Modelo SDS para el caso propio (protocolo de actuación, taxonomía de defectos, etc.) y comenzar a medir nuestra propia serie histórica.

Esto es, personalizar y arrancar el modelo con una simple célula de trabajo que pueda servir como grupo de referencia (benchmark de métricas y procesos).

A partir de este punto, se pueden añadir células de trabajo (o grupos de células de trabajo) a gran velocidad por uno u otro camino (externalización o capacitación de equipos existentes). En todo caso, es fundamental industrializar el proceso de capacitación debidamente personalizado por segmento (ingeniero, supervisor y directivo), mantener un equipo de referencia custodio de la evolución del modelo y punta de lanza del mismo y diseñar una adecuada arquitectura de movilización para todas las partes afectadas de la organización.

Por tanto, es perfectamente factible escalar el uso del modelo de forma rápida y controlada hasta alcanzar grandes volúmenes.

Los resultados finales mejoran con el tiempo en un doble sentido. En primer lugar, porque una parte importante del beneficio se produce en la fase de mantenimiento y el ahorro será más apreciable en el resultado final según aumente la proporción de producto desarrollado con SDS frente al total del producto en explotación.

Y en segundo lugar, por el efecto de la curva de aprendizaje, los profesionales no alcanzan todo su potencial hasta dos o tres años de experiencia real en el uso del modelo.

Sin embargo, se apreciarán claramente sus primeros beneficios desde el primer proyecto ejecutado de esta manera. Normalmente a los 2 o 3 meses de uso se empiezan a apreciar mejoras en el desempeño de los equipos.

9. Conclusiones

Estamos cerca de un importante cambio de paradigma en la industria de desarrollo de software a medida. Este cambio será similar en magnitud y resultados a los que han ido sufriendo diversas industrias manufactureras durante estas últimas décadas, de las que se pueden extraer lecciones aprendidas válidas.

Con esta visión en la mente, en Steelmood hemos desarrollado un modelo propio (SDS) para ayudar a nuestros clientes a dar el salto al nuevo paradigma cuanto antes. Desde luego que no será fácil conseguirlo, pero es imprescindible intentarlo si uno quiere subsistir. Aunque, como le gustaba decir al propio Deming, "al fin y al cabo, subsistir no es obligatorio".

Cómo citar este artículo / How to cite this paper

Apellido, A., Apellido, F., y Apellido, M. (2014). Software Development by Steelmood. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 38-50. Consultado el [dd/mm/aaaa] en www.ijisebc.com

References

Gouillart, Francis J. y Kelly, James N. 1.996. Transforming the Organization. McGraw-Hill Inc., 1.996.

Humphrey , Watts S. 2000. The Personal Software ProcessSM (PSPSM). TECHNICAL REPORT. TECHNICAL REPORT CMU/SEI-2000-TR-022 ESC-TR-2000-022. Team Software Process Initiative. Carnegie Mellon Software Engineering Institute November 2000.

Humphrey , Watts S. 2000. The Team Software ProcessSM (TSPSM). TECHNICAL REPORT. CMU/SEI-2000-TR-023 ESC-TR-2000-023. Team Software Process Initiative. Carnegie Mellon Software Engineering Institute November 2000.

Manies, M. & U. Nikual, U. 2011. "La Elicitación de Requisitos en el contexto de un proyecto software". Ing. USB Med, Vol. 2, No. 2, pp. 25-29. ISSN: 2027-5846. Jul-Dic, 2011. Rao, Jay y Chuán, Fran. 2012. Innovación 2.0 ¿Por qué cuando hablamos de innovación nos olvidamos de las personas? Editorial Profit.

Lean Advancement Initiative http://ssrc.mit.edu/programs/lean-advancement-initiative-lai

El modelo de Kano y gestión de expectativas http://steelmood.blogspot.com.es/2014/03/como-gestionar-las-expectativas-de-los.html y http://www.isixsigma.com/tools-templates/kano-analysis/

© ISSN: 2387-0184

On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE?

Sobre la adopción industrial del Model Driven Engineering (MDE) ¿Está su empresa preparada para MDE?

Antonio Vallecillo[1]

[1] Universidad de Málaga, Spain

av@lcc.uma.es

ABSTRACT. Model Driven Engineering (MDE) is an approach to software development where models play a central role in all software engineering processes. Conceived to provide significant gains in productivity, portability, maintainability and interoperability, MDE is now starting to be effectively used in industry. Thus, companies are beginning to evaluate their possibilities for adopting it. This paper examines the current state of MDE in industry, summarizes the current obstacles for adoption, and discusses the advantages that it should bring to businesses and its limitations. Finally, some ideas for a smoother transition towards a wider adoption of MDE are outlined.

KEYWORDS: Model Driven Engineering, MDE, Software development, Software engineering processes, Adopting MDE, Software Engineering.

1. Introduction

1.1. Context

The ever-increasing complexity of software applications and the rapid changes in the technologies are posing serious challenges to our traditional software development methods and tools, which are having problems to cope with the new requirements. At the same time, most companies are now facing serious difficulties with their IT systems, in particular in the way in which they are developed, acquired and perceived by users:

– Customers are increasingly demanding better functionality, improved performance, more usable interfaces and enhanced integration facilities with their heterogeneous systems. These needs are normally very hard to incorporate into current applications and software production environments.

– Investments in software applications and IT systems are seldom recovered because of rapidly changing and volatile technologies – many enterprise applications become obsolete even before their costs are amortized.

– Migration and evolution of systems are costly and painful processes. This is why many companies decide to build sets of new layers atop their existing applications, fearing that a change in their core systems will quite simply ruin their business.

– Developing large applications by manual coding is still a very expensive and rather unpredictable process: very often software projects are, for the most part, overdue, error-prone, and blast all budget previsions.

– When software development is outsourced, the contracting company ends up becoming over-reliant on the application developer, losing most of its independence and control. In-house software development has several drawbacks too, because maintaining an IT department up-to-date with the latest tools and able to keep up with the ever-evolving technologies and experience represents a high cost – and does not necessarily solve the aforementioned problems, either (e.g., late, inefficient and over-budget projects).

– Once a system has been developed, most companies are only left with the code of the application, where their business knowledge and rules remain embedded or hard-wired. Thus, migration to a new technology normally means starting from scratch, because companies have no reusable and technology-independent blueprints of their business models.

Software Engineering is the discipline whose goal is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [IEEE90]. Thus, Software Engineering is currently trying to tame the inherent complexity of such processes and of the new requirements in different ways, which range from the creation of new programming languages (e.g. multi-paradigm ones) and development methods (e.g. agile) to new abstraction mechanisms and tools.

One of these new approaches is called Model-Driven Engineering (MDE) [Sch06, BCW12]. MDE is an approach to software development where the central focus of attention shifts from writing code by hand to dealing with higher levels of abstractions (models). Moreover, MDE broadens the scope of models beyond code generation, lifting them to first-class citizens of all software engineering processes: analysis, design, development, maintenance, modernization, etc.

MDE is based on three sound and time-proven principles: higher levels of abstraction, higher levels of automation, and standardization [B+04, BCW12, Sel03]. MDE raises the level of abstraction by enabling specifications that use different models to focus on different concerns and by automating the production of such specifications and the software that meets them; the use of standards aims to ensure the required interoperability between all MDE artefacts and tools.

MDE represents a big leap in Software Engineering, especially for developing and maintaining information systems, and implies some radical changes to the way software is constructed using traditional programming

methods. MDE has already proved to bring along significant gains to companies implementing it. But there are of course many companies still reluctant to adopt these changes to their proven internal processes and to their businesses in general: for many, MDE only represents yet another fad in software engineering. However, this time there is sufficient evidence that MDE is mature enough to provide significant benefits and value to companies, and to consolidate as an established approach for the development of industrial software. This paper discusses such evidences, and analyses the state of MDE adoption by the industry, the current difficulties for implementing it, as well as the advantages that it should bring to companies and its limitations.

1.2. A bit of history

It all started in the 90s with the advent of the Unified Modeling Language (UML), which provided designers with a standard set of graphical notations for representing software systems at a higher level of abstraction than code permitted, hence removing part of the accidental complexity of the problems being addressed. Models proved to be very useful for understanding the interesting characteristics of a system (either existing or to be built), predicting some of its properties, communicating with stakeholders, or guiding the implementation of the system (i.e., models as blueprints). Bran Selic perfectly defined the characteristics that software models have to satisfy to be useful [Sel03]: they have to be purposeful (i.e., address a specific set of concerns); abstract (emphasize important aspects while removing irrelevant ones); understandable (expressed in a form that is readily understood by observers); accurate (faithfully represent the system modeled); predictive (can be used to answer questions about the system modeled) and cost-effective (much cheaper and faster to develop and maintain than the actual system).

But it was soon clear that having models was not enough. They had to be manipulated and maintained in an automated way, and be an integral part of the software development processes. To address this issue, the Model-Driven Architecture (MDA) initiative was launched by the OMG in late 2000 to propose a new way to consider the development and maintenance of information systems [Wat08]. In the MDA approach, models are the key elements used to direct the course of understanding, design, analysis, construction, testing, deployment, operation, administration, maintenance, evolution, integration, acquisition and modernization of systems. MDA differentiates between platform-independent models (PIM) and platform-specific models (PSM). Thus, the basic functionality of the system can be separated from its final implementation and the business logic can be separated from the underlying platform technology. Models are described using metamodels that define the concepts of the languages in which models are written, as well as the well-formed rules that correct models should fulfil. MDA also introduced model transformations, as the key elements that enable the automated implementation of a system from the different models defined for it or, alternatively, enable reconstructing abstract models from legacy code to realize software modernization or migration.

We can look at MDA as one way of realizing MDE. More precisely, MDA is the approach proposed by the Object Management Group (OMG) to achieve MDE, using OMG standards (e.g., UML for modeling, MOF for metamodeling, QVT for model transformation, XMI for model interchange, etc.). The Eclipse ecosystem of programming and modelling tools (Eclipse Modeling Framework) is another MDE approach, where EMF provides metamodeling facilities; ATL is used as transformation language, etc.

MDA also permits defining further models of the system, each one focusing on a specific concern, and at the right level of abstraction. These specific models are described using Domain Specific Languages (DSLs) [FP10, Tol11, Vol13, KLRT13] and are related by model transformations that drive the automated code generation from the models.

Since the emergence of MDA much has happened in the field of system and software model engineering. A variety of new acronyms (MDD, MDE, MBE, MIC, ADM, SBA, MDI, etc.) are appearing to delimit the constantly extending scope of application of the core modeling techniques. In addition, the evolution towards modeling practices has joined with the Open Source Software movement in environments like Eclipse to give more strength to this paradigm shift.

For example, Model-Driven Development (MDD) principally focuses on the generation of implementations from models [Sel03, BVGR08], and can be considered as part of MDE, which is a wider discipline that covers all software engineering activities, beyond automated code generation, using models, metamodels and model transformations. In MBE, in turn, software models play an important role although they are not necessarily the key artifacts of the development – i.e. they do not "drive" the process [Cab14].

1.3. Some misconceptions about modeling and MDE

Before going any further, it would be useful to identify some typical misconceptions that many software engineers and developers still have about models and MDE.

a) Programs are not models, models are not programs. Many people think that programs are text-based sets of instructions that can be executed by a computer, and models are graphical, non-executable representations of a system. But programs can also be written in a visual form, and many models are in fact executable. There are also models described using textual notations (such as HUTN, for instance). Probably the only difference between programs and models lies on the level of abstraction, and the fact that a program must always contain enough information to be executed in a given platform (models may not). Interestingly, this has led to the term "prodel" (or "mogram" [Kle08]) to refer to either a program or a model, in cases in which they can be used interchangeably.

b) MDE = Modeling (or, similarly, MDA = Using UML). As mentioned above, MDA not only entails higher levels of abstraction using models as representations of a system, but also implies automation and standardization. The UML only provides notations for representing abstractions of a system, being just one part of MDA. In fact, models are not sufficient; they only leverage all their potential when used inside MDE processes. Similarly, MDE not only involves modeling the system (or some aspects of it), but also considers these models as software artefacts that need to be manipulated using model transformations or other model operations (difference, merge, etc.) for various purposes: generating implementations (i.e., code and configuration files), deriving analysis models to analyze the system properties, inferring models from existing code, or maintaining all related artefacts in synch, e.g., to realize round-trip engineering. Depending on the technological spaces involved in a model transformation, it can be classified as model-to-model (M2M), model-to-text (M2T) or text-to-model (T2M) transformation.

c) Modeling = Using UML. The UML is just one modeling language, but it is not the only one. In fact, Domain Specific Languages (DSLs) are now becoming commonplace for modeling the structure and behavior of systems [FP10, Tol11, BCW12, Vol13, KLRT13, BF14]. In addition, other modeling notations are commonly used in practice, including Flowcharts, the Entity-Relationship (E/R) notation, IDEF, Simulink, Modelica, the Goal Structuring Notation (GSN), BPMN, i*, System Context Diagrams (SCD), etc. Normally, software engineers use more than one modeling notation in their projects. A recent study [Tow14] showed that UML, Flowcharts, SCD and SysML were those more heavily used, and that, on average, software engineers use 5.5 different modeling languages at work. The complete list of modeling notations used goes up now beyond 35, which clearly confirms that UML is not the only modeling language in town.

d) MDE has nothing to do with business processes. MDE is not only about modeling the structure of systems or their behavioral and extra-functional aspects. Models can also be successfully used to represent business processes (using any existing notation apt for this purpose, such as BPMN, SPEM or UML), which can then be designed, analyzed, simulated, implemented, deployed, managed, re-engineered and maintained using existing MDE practices and tools. This discipline (called Business Process Modeling) is gaining acceptance in the industry as more usable and powerful supporting tools are becoming commonplace and readily available – and not only to specify and reason about business processes, but to generate code (see, e.g., [W14]).

e) MDE implies an "all-or-nothing" approach. Adopting the use of models and MDE practices needs not

be a binary decision across the company, or even within a project. Every company should decide "the right amount of modeling" for every project [Cab13], depending on its particular characteristics and on the project requirements.

2. Current state of MDE adoption

2.1. Empirical studies about the adoption and use of MDE in the industry

MDE was originally conceived to provide significant gains in the development of software in various aspects – chiefly in productivity, portability, maintainability and interoperability [KW03, CK08]. However, the industrial adoption and use of MDE still seems to be an exception rather than the norm [Sel12].

There are numerous verifiable examples of success stories with MDE, which are clear proofs of its feasibility and value. There are of course cases of failure, too. Thus, companies deciding whether to adopt MDE or not are often faced with confusion: successful cases tend to paint things too brightly, whereas cases of failure do not seem to have applied MDE properly [HWRK11]. In general, there has always been a lack of precise information about the level of adoption of MDE in industry, how it is used, and how (and when) it has been adopted. Besides, it is difficult to provide absolute measures of the real benefits of MDE and of its impact in current development projects.

To respond to the lack of accurate and contrasted information, the MDE community has been very actively working over the past few years on the compilation of empirical data and evidences about the use of design models in industry [A+6, BBB11, DP06, FBL10, G+11, GTA14, LCM06, Pet13, Pet14] and about the adoption of MDE practices [A+07, Avi14, AVT06, B+14, BB14, BC10, BF14, BLW05, CGR14, DGHC14, DGWC14, DMBD14, DPP14, DV10, FL08, HRW11, HWRK11, HWR14, KR06, KR08, KRK10, KTM12, LRTR13, MCMT14, MD08, MGS13, M+06, N+14, OMG09, Pez14, PPL13, R+06, SCG14, Sel12, Tow14, T+11, T+12, T+13, T+13b, WW06]. The goals of these studies have been to measure the penetration of MDE in industry, to analyze the benefits and problems found, the hurdles for adoption, and the actual value that MDE is able to deliver to companies. The following subsections summarize the findings of these papers, focusing on the aspects of interest to this study.

2.2. Models are not quite here yet

Regarding the use of design models in industry, all systematic studies seem to agree that the knowledge about modeling and its use are not widespread yet. In the main, developers do not make the best use of models and tend to perceive little or no value added in modeling. More precisely:

- Design models are not used very extensively, and where they are used, the use is informal and mostly without tool support; the notation is often not UML but many others.
- Models are used primarily as a communication and collaboration mechanism where there is a need to solve problems and/or get a joint understanding of the overall design in a group.
- Many software practitioners are still completely unaware of modeling notations and of MDE.
- Many software designers, who use UML, are using models only to illustrate and document the architecture of a software solution.
- The use of models decreases with an increase in developers' experience, and increases with higher levels of qualifications.
- Many software engineers currently use diagrams and simulations in their work but do not consider they are modeling.
- Models are seldom updated after initial creation, and are usually drawn on a white board or on paper.
- Complexity, coordination, and cost are ongoing issues.
- OCL [WK03] is rarely used in the modeling community. Only a few software engineers agree

with the importance of constraints, and the majority of modelers never define pre and post-conditions or invariants as part of their models. Constraints are either completely ignored or incorporated later at the code level, using programming languages.

– Tool support is still insufficient, in particular for model validation, simulation and interchange; for specifying constraints and correspondences between models; for compiling OCL or ALF statements to robust code libraries that can be used in production environments, etc.

– As long as there is no common agreement about a usable action language for specifying the behavior of models and executing them (e.g., ALF), developers are not really interested in another language which is restricted to constraint expressions. The benefits of models are not seen in comparison to programming languages by many current software developers.

– Modeling also requires a mindset change. The use of (the right) abstraction techniques is more difficult and less common than expected [SZ13], and modeling requires different skills other than programming.

As summarized by M. Petre [Pet14], "One of the major reasons professional practitioners give for declining to use UML is that, after due consideration, they have concluded that it doesn't add sufficient value to warrant the cost of transition. Many practitioners already have a repertoire of tools and representations that has been thoughtfully developed and evolved over time to fit their effective practices."

2.3. The value of MDE

The previous statement by Petre is a real and common criticism, but it normally applies to the inadequate introduction of models in the development processes of companies. As discussed before, models leverage all their potential benefits when used as part of MDE practices, and not in isolation. This is reinforced by most conducted studies, which show that models, when integrated within MDE processes and used in the right domains, can significantly improve both the quality of software and the productivity of the teams that develop it [A+07, Avi14, AVT06, B+14, BB14, BC10, BF14, BLW05, DGHC14, DGWC14, DPP14, DV10, MCMT14, MGS13, SCG14, WW06]. In particular, experiences of adoption of MDE in industrial settings report about significant gains:

– On average, projects achieved between 60% and 100% code-generation, contributing to significant productivity and quality improvement. In particular, productivity improvements ranged between 2X and 8X, when measured in terms of equivalent source lines of code or in function points.

– Models proved to be successful in obtaining an approximate 2.3X reduction in effort through the use of co-simulation, automatic code generation, and model testing.

– The use of scenario-based test generation tools yielded approximately 33% reduction in the effort required to develop test cases.

– Overall reduction in defects ranged between 1.2X and 4X, and detection happens much earlier in the development process, when they are less expensive to fix.

– The overall cost of quality decreased due to a decrease in inspection and testing times.

– Up to 80% of maintenance costs can be saved by working directly with models.

For example, in one case study at Motorola [BLW05], a 30X to 70X reduction was reported in the time needed to correctly fix a defect detected during system integration testing. This reduction was due to the ability to add a model test that illustrates the problem, fix the problem at the model level, test the fix by running a full regression test suite on the model itself, regenerate the code, and run the same regression test suite on the generated code – the time to do this was 24 hours or less, while achieving the same quality with several hundred thousand lines of hand code could have easily taken them one to two months.

Web engineering is another domain where MDE has demonstrated many of its potential benefits for building complex Web-based information systems (leading what is now called "Model-Driven Web Engineering", MDWE). MDWE has proven to be economically profitable and sustainable in many real cases, with peak pro-

ductivity rates reaching five times the number of delivered function points per staff/month of traditional programming languages like Java [A+07]. Empirical studies in large companies (e.g., ACER) also show that between 2 and 3 months are required to train developers and make them fully productive with the MDWE tools; but once they master them, gains in productivity can reach 60% of development time, and up to 80% in maintenance and evolution costs [BB14] – due to the automatic generation of code from models (it can be fully automated in many MDWE applications), and the possibility of working at the model level for maintaining the systems [BF14].

In general, MDE has been successfully applied in a broad range of companies, and in a number of different domains including telecommunications, business and finance, defense and aerospace, manufacturing, health, and web information systems. The size of companies that adopt MDE for their own developments is normally large. Very small firms, in turn, are successful in using MDE for conducting consultancy and software development for others: there is a growing market for companies providing Domain-Specific Modeling environments, tools, consultancy and support. Example of such successful companies include MetaCase (https://www.metacase.com/), Obeo (http://www.obeo.fr/), Atego (http://www.atego.com), Sodious (http://sodius.com/), WebRatio (http://www.webratio.com/), Icinetic (http://www.icinetic.com/), Itemis (http://www.itemis.com/) and Mia-Software (http://www.mia-software.com/), among many others.[1]

Apart from gains in quality and productivity, significant improvements in reuse and maintainability have also been reported. The degree of modeling maturity has evolved from informal whiteboard or napkin-modeling to formal modeling with simulation, code generation, test-case reuse, automated marshaling code generation, etc., which are the processes where the real value and benefits of models and MDE can be perceived.

Moreover, large companies that have successfully adopted MDE clearly recognize that models, in their own right, have become very valuable assets. Models permit capturing the company business rules, processes and knowledge, independently of the technologies used to implement them, and at the right level of abstraction. These high-level models now allow companies to analyze and reason about their processes and business rules using specialized tools, much more efficiently than before.

3. Adopting MDE

3.1. Hurdles to MDE adoption

In light of these results, one wonders why companies are not adopting MDE yet. Far from this, MDE is not a dominant software development paradigm and is still seen by many practitioners as "either unproven or, even worse, as yet another waning fad in a discipline that already suffers from an excess of silver bullets" [Sel12].

The fact is that although the benefits of MDE are frequently cited and are often considered to be obvious, there are reasons why MDE may have detrimental effects. One of them is that MDE involves dependent activities that have both positive and negative effects. For example, automatic code generation can have a positive effect on productivity. But the extra effort required to develop the models, along with the possible need to make manual modifications, may have a negative effect. The balance between these two effects is related to context, and needs to be carefully considered [HRW11, HWR14].

Furthermore, there are significant hurdles that may hinder the adoption of MDE in practice. Although many of these are technical in nature, the main ones come about for organizational, cultural and economic reasons [Sel08, HRW11, Sel12, HWR14, MCMT14].

[1] The list shows just a few companies as examples; it is by no means intended to be complete or exhaustive.

Examples of Technical hurdles include:

– Lack of mature tools (scalable, robust, usable, efficient and interoperable). As occurs with programs, if no good compilers/debuggers/IDEs are available for a programming language, nobody beside academics will ever use it (especially in production environments).
– Lack of well-defined semantics (of models, metamodels, and transformations) and of a sound theory of MDE.
– Lack of documented and proven MDE processes.
– No "MDE Good Practices" manuals available. Only success stories of various companies (mostly large companies) have been documented. However, these reports normally give no indication on whether the changes implemented in one company will work for others, how to apply them, or when.
– Lack of model validation tools.
– It is by no means guaranteed that higher abstraction levels lead to better software [Kra07].
– Different flavors of MDE exist and choosing the right variant for a company's business is critical (and not obvious) to a project's success [HRW11].

Examples or Cultural hurdles include:

– Lack of education, team experience and skills sets in most developers and software practitioners. MDE is not only a change in technology, but a complete paradigm shift.
– Too much emphasis on technology and not enough on technology users and their needs.
– Inadequate or flawed information about MDE concepts, goals, tools and real achievements – for many companies it is not clear whether MDE is just an academic theory, the tool vendor's sales pitch or if there are indeed many organizations successfully using MDE to realize measurable benefits on real software engineering projects.
– Lack of "systems" perspective and lack of abstraction skills.
– Conservative mindset of many software practitioners; resistance to technological change, even if the new technology can lead to better results.
– Conservative mindset of managers, normally reluctant to drastic changes in development processes, methods and tools.

Economic hurdles include:

– Fear of excessive over-costs.
– Current business climate heavily focused on short-term gain discourages investment in new methods and tools [Sel08].
– Upfront investments difficult to quantify in the mid-term.
– Development teams' re-education and training can indeed be expensive (since it may imply changing their mindsets, not only their methods and tools).

3.2. Pros and Cons: Technical Issues

Something critical for a company before deciding whether or not to adopt MDE (either in full, in a project, or in a set of related projects), is to understand the pros and cons that such an adoption would imply. We have previously mentioned (Section 3.1) that MDE may also have detrimental effects, and that each decision can entail advantages and disadvantages, as well as savings and associated costs, which need to be carefully evaluated.

These tradeoffs can have a significant impact on different aspects of the development process and on the final product quality (see, e.g. [BLW05, DV10, HWRK11]):

– Requirements elicitation time is reduced by the use of models and the right domain specific lan-

guages, which use notations closer to domain experts and to the system stakeholders; but these efforts can be completely wasted without the right tools that make effective use of them in the whole process – not to mention the time needed to master these new languages.

– Design time is reduced by the use of the right languages and tools, which work at the appropriate level of abstraction; but the learning curve needed to master these notations and tools is not negligible and it may require several projects before it starts to pay off (see, e.g., [Cab11, DV10]).

– Development time is reduced by automatic code generation; but it is also increased by the need of developing models and implementing model transformations. Here we also need to mention that higher level artefacts are easier to reuse in subsequent projects (for instance, the models or even the model transformations).

– Protection of investment. Depending on the domain, developing models could be a more difficult, expensive and time consuming task than programming; however for a company, the value of models can be much higher than programs when used within MDE environments: models become platform-independent; they can be used to generate code into different platforms; have a longer lifespan than code; are more reusable, and can facilitate the implementation of novel applications (such as Simulation-based Acquisition [Nav13] or Model-driven Interoperability, for example). In addition, high-level models allow companies to become more independent from software vendors and technology provides, hence protecting their investment in modeling.

– Testing time is reduced by reducing defects in code and by the use of model-based techniques; but it can be significantly increased by the effort needed to test model transformations and validate models (mature debugging and validation tools for models and model transformations are not commonplace). Besides, testing times can also be significantly reduced by the use of models and the application of Model-based Testing (MBT) techniques. Again, the introduction of MBT requires the implementation of changes in the testing processes of the company, which cannot be neglected.

– Maintenance time is reduced since it is achieved at the model level; but new efforts are required to automatically keep models and code in synch.

– Integration time is reduced because it is done at model level, and all gluing and mediation artefacts can be automatically generated; but increased by the need to define correspondences between the models, and implementing the transformations that generate and deploy all artefacts.

By looking at these tradeoffs, we can synthesize from them three major factors that do have a significant impact on the success of an MDE project:

– First, the level of knowledge and expertise of the company in the problem domain (this only depends on the particular project), and the value that the company sees in solving the problem.

– Second, the level of training (and education) of the development team on the solution domain (Modeling and MDE, in this case), and their familiarity with it.

– Third, the availability of the right languages and tools for solving the problem and implementing the solution. It may be the case that the tools are not yet ready for industrial use, being just mere prototypes or toy-ish applications that do not scale, are not robust enough, or are unmaintainable.

Interestingly, these three aspects are related to the determinant technical factors that Mohagheghi et al. identify in [MGS13] for the adoption of MDE in industrial settings: perceived usefulness, ease of use and maturity of the tools.

Therefore, each company, depending on the particular project, on its perceived value and ROI, and on its capability to be resolved using MDE, should decide whether to adopt MDE or not, for which projects, and with which development team. The criteria to use, and the measures needed to assess the success of the project should be clearly established since the beginning, and continuously evaluated throughout the overall process.

Note that outsourcing part of these projects could be an option in some early projects, in the case of a lack

of in-house resources or expertise. For example, external MDE experts can be contracted to introduce the company development team to this new paradigm, helping to train them in MDE; or simply subcontracted for developing part of one project using MDE, so that our company could evaluate its use and its results in our particular application domain.

3.3. Pros and Cons: Beyond Technical Issues

Apart from the technical issues mentioned above, companies should take into account other factors that also affect the adoption of MDE practices. In particular, these factors include a number of cultural, social, economic and organizational issues, which also influence the relative success or failure of MDE practices. Most of these issues have already been mentioned in Section 3.1, when listing the general hurdles to MDE adoption. In contrast to the technical issues, they are normally the more critical ones to overcome [Sel12, HWR13].

Some of them could be solved by counting on better educated software engineers: not only in technical questions but also in cultural, human and business matters – allowing them to view technology more as a means rather than as an end in itself.

In the long term, education should be facilitated through changes in academic curricula and investment in foundational research: the competencies needed for dealing with the new concepts should be part of the education provided by universities and other teaching institutions. This is similar to the introduction 25 years ago of object-oriented programming (OOP) in the curricula of computer scientists and software engineers. Current managers no longer question the value of OOP, because they know it well and it forms part of their culture. More importantly, in MDE this knowledge accounts not only for modeling theories, concepts and tools, but also for the business aspects and the workings of the marketplace. At the same time, more research on a theory of MDE is required, to ensure that the corresponding technology and methods are well understood, useful, and dependable [Sel12].

In the meantime, companies have two choices: they can either invest in educating their software engineering teams, or subcontract external experts that introduce these concepts and expertise into the company know-how and practices (by, e.g., jointly working with local teams on MDE projects). Alternatively, they can outsource the MDE development to third parties, while they continue to use their previous development methods and tools. It is therefore up the company to estimate the cost of training and re-educating their development teams, versus outsourcing the services they need.

Although better education can tackle most of the social and cultural issues mentioned in Section 3.1, economic issues also need to be addressed. Normally, they can be mitigated by a gradual introduction of MDE practices in the company. In this way, pilot projects can serve to assess the actual costs of the projects in a controlled way, hence alleviating possible risks and unwelcome surprises. The pilot projects could also allow managers to estimate the costs of MDE adoption in other environments, the upfront investments needed, and their expected ROI.

Finally, we also have to consider the managerial and organizational aspects that also influence the successful adoption and deployment of any new paradigm, in this case MDE. Several authors [Sel08, HRW11, Sel12, HWR14, MCMT14] have systematically analyzed various industrial case studies, from different domains, and they have all identified the same essential organizational aspects of any successful MDE deployment.

– Motivation: The company should have a clear reason (not only technical) that drives the change, and such a reason should be shared by all employees.

– Commitment: the organization must be fully committed to making the project a success; otherwise the doubts can ruin the adoption process when initial problems appear.

– Innovation: the organization must be willing to adopt new development processes, integrate them with their own processes, and adapt the existing ones if required.

– Gradual adoption: MDE should be adopted in a progressive and iterative approach; pilots play a key role here, versus all-or-nothing strategies.

– Conscious adoption: Adoption of MDE requires a real understanding of the necessary process and technology changes, the consequences of the adoption, and the associated costs and benefits.

– Involvement: The decision to adopt MDE cannot be made by a single individual, group or department. All principal players in the company should feel part of the change.

– Business focus: ultimately, successful MDE has to have a clear business focus. MDE should not only be adopted for technical reasons, but as a solution to new commercial and organizational challenges.

3.4. And now?

So far we have discussed the current state of the adoption of MDE in industry, the current hurdles for that adoption, and the pros and cons that companies considering MDE should weigh up. It is now up to each individual company to decide whether or not to adopt MDE, how, and when.

Although there are no universal answers to these issues, the following list of questions could be helpful to companies in order to make decisions on the adoption of MDE in their organizations.

1. Does my company understand the implications of the necessary process change to adopt MDE in the project? [What are those changes? What are the implications?]
2. Is the project I'm considering able to be carried out using MDE? [Why? How?]
3. Does my company have development teams trained in MDE concepts, tools and processes that can carry out the project? [Who are they?]
4. Do I have the MDE tools that I need for carrying out the project? [Which ones?]
5. Are the MDE tools that I need mature enough for the job? [Why?]
6. Is the MDE approach that I am considering easy to use by my development teams, and to integrate with the rest of my company's notations, processes and tools? [Why?]
7. Is the company management seriously committed to the adoption of MDE? [How much?]
8. Do I know how to estimate the costs and duration of the project, and the expected ROI? [How?]
9. Have I established the criteria to use and the metrics needed to measure the success of the project? [What are the criteria and the metrics?]
10. Is the project technically feasible and economically viable? [Why?]
11. Have I considered outsourcing the project to a company with expertise in MDE? [Why?]
12. Do I have any real need to adopt MDE for the project? [Which one? Is it urgent?]

4. A new paradigm shift

By looking from a high-level point of view at all the issues and tradeoffs outlined above, they represent something that we have seen before. They resemble the situation we went through in the late 80s when Object-Oriented Programming (OOP) emerged and started replacing the existing and well-established Structured Programming: the strong acceptance by Academia, the initial doubts by large companies, the slow emergence of useful tools, and then the wide adoption by industry. Basically, we are just facing a new paradigm shift, in which models are the new first-class citizens of our theories, methods, tools and techniques – the same role objects took when OOP was adopted. We should learn from that experience.

OOP was originally created in the late 1960s at the Norwegian Computing Center in Oslo, when Ole-Johan Dahl and Kristen Nygaard developed the Simula I and Simula 67 programming languages. It took Objects almost 30 years to be start to become mainstream: almost 14 years to be known in the industry (in 1981, with the publication in the Byte magazine of a special issue dedicated to Smalltalk [Byt81]), and then another 14 years to become the dominant programming methodology in the mid-90s, as acknowledged by the Gartner Group in 1995 [Gar05].

To analyze the level of technology adoption of MDE, let's examine Figure 1, which plots together the Gartner "Hype Cycle" (HC) model [Gar14] and the "Technology Adoption Lifecycle" (TALC) model popularized by Everett Rogers and Geoffrey Moore [Moo14].

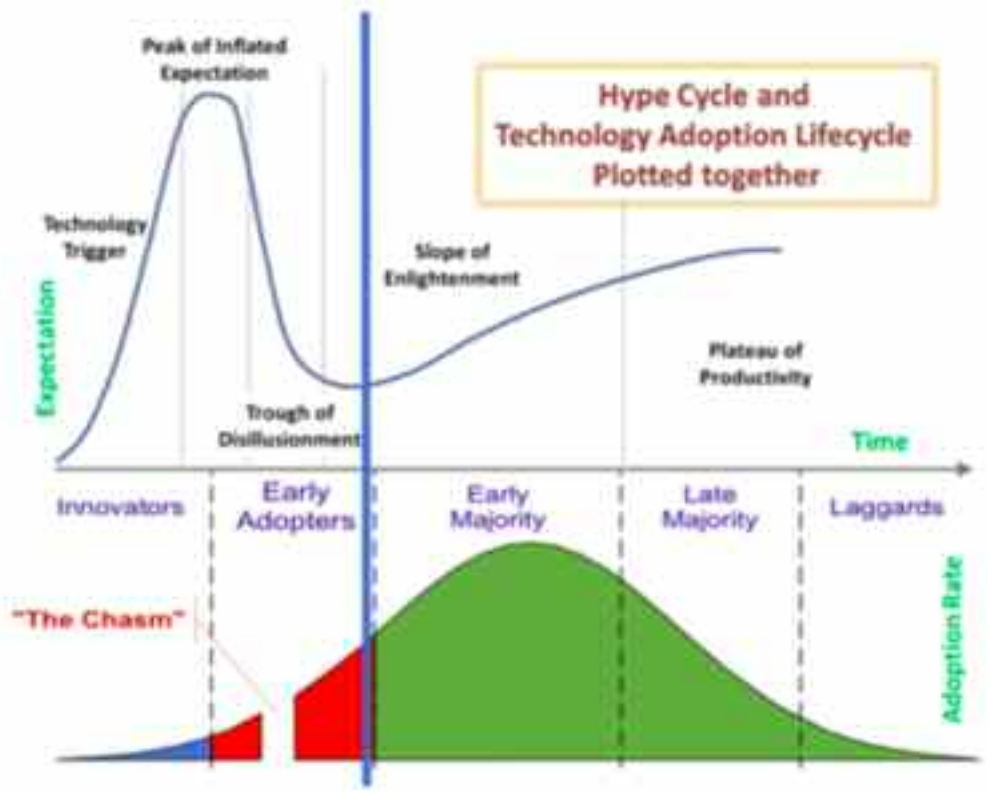


Figure 1. The "Hype Cycle" (HC) and the "Technology Adoption Lifecycle" (TALC) models plotted together (from: http://setandbma.wordpress.com/2012/05/28/technology-adoption-shift/), with the current position of Model Driven Engineering indicated by a thick blue line.

Most studies (e.g., [CGR14, Tow14]) tend to coincide that MDE has just passed the Trough of Disillusionment of the HC model (where it was in 2012 [Gar13]), and is now starting the second period of growth. This is corroborated by several facts, which are precisely the ones that characterize the start of the Slope of Enlightenment of the Hype Cycle model [Gar14]:

– the market has already started to mature, and it has become more realistic about how the new technology can be useful to organizations;

– many companies have already commenced to use the technology, and more organizations fund pilot projects;

– there are more instances and real case studies of how the new technology can benefit the industry starting to materialize and becoming more widely understood (see, e.g., the list of references in the bibliography at the end of the paper);

– second and third-generation products are appearing from technology providers: tools are starting to become more usable, scalable and robust;

– conservative companies still remain cautious.

Considering the TALC model, presently MDE has already passed the Chasm, surviving the most critical

phase, and is now at the beginning of what is called "The Bowling Alley". At this stage, where we are now, market momentum picks up as early pragmatists overcome their reluctance and decide to adopt the new technology to solve niche-specific problems. These companies decide to leave their comfort zone to find solutions for particular applications in which they want to stand out from the competition, or for which they find no other solution.

Knowing where we are, we can benefit from previous experiences of technology adoptions. In particular, the key to success at this stage is to use a gradual and iterative approach to adoption. First, to provide a complete solution for one application (or domain) while identifying closely aligned applications that could benefit from a similar solution [Moo14]. When the momentum from successfully delivering solutions for the first application (the lead bowling pin) is felt, this momentum is leveraged into adjacent applications. By dominating several applications, the company is now able and ready to decide when to use MDE and when not, estimate the benefits and costs for the adoption in each case, and make an optimal use of this new paradigm.

To estimate the adoption of MDE in the coming years, we can easily draw a parallel between OOP and MDE, assuming that there is a difference of 25 years between them (although OOP was used in academia since the 70s, it was not known by the industry until 1981, and started to be adopted in the 90s; in the case of MDE, OMG launched MDA in 2001 but MDE was not fully recognized by the industry until 2005 [Gar05, Sch2006]) and that the pace of adoption by industry will be similar, both being comparable technologies. In this respect, the situation of OOP in 1991 was very similar to the situation of MDE now, ending the period of Early Adoption and about to start climbing the second slope of the Hype Cycle. Thus, we estimate that the situation of MDE in 5 years' time (around 2020) will coincide with the position of OOP in the chart in 1995. That is, if MDE technologies keep maturing and developing at the same pace, we can expect MDE to reach the peak of adoption by the early majority of the industry in 2020.

As it happened with OOP, we will know that MDE has been successfully adopted when it becomes transparent and we no longer discuss about its application: it becomes an integral part of our curricula, and fully integrated into our software engineering processes and practices. This does not mean that we apply it in all projects, but that we know how, and when, to apply it.

In this sense, all indicators show that MDE has already passed the turning point, and its adoption by industry is progressively expanding. Thus, companies should now start to evaluate their options and opportunities for adopting MDE, and the strategies to make this happen.

5. How can we help?

Regardless of what individual companies decide, there are also collective actions to be carried out by different groups to advance MDE and to exploit its full potential.

In the first place, the competencies needed for dealing with MDE and with its new concepts and mechanisms, should be part of the education provided by Universities and other teaching institutions. Most universities already incorporate UML and modeling courses in their curricula. MDD and MDE are becoming popular subjects in many master courses too, with many masters entirely devoted to MDE, and several universities already offering subjects on model-driven matters at graduate level. Moreover, the use of Models and Model-driven techniques has been incorporated in the new edition of the Software Engineering Body of Knowledge (SWEBOK) [PF14]. The presence of MDE topics in most Software Engineering curricula today is one of the definitive signs of the relevance of MDE and the best guarantee that the adoption of MDE by a company is worth the investment in the medium and long term.

But universities should incorporate not only MDE theories, concepts and tools into their Software Engineering curricula, but also business aspects and the workings of the software marketplace [Sel12, Pai14].

As we have seen, technology should not be an end in itself, but just a means, and should always be considered within the business context it is developed for and applied in.

At the same time, more research is required on MDE. New methods and tools to cope with the growing demands imposed by new technologies and increasingly complex systems are needed. More importantly, there is an urgent need for research on a sound theory on MDE to ensure that the corresponding technologies and methods are well understood and reliable [Sel12].

Finally, companies should take the initiative now and become the drivers of the change. Of course, MDE technologies are still not perfect; they need to be significantly improved to be more useful in industry. New problems will also appear as soon as MDE is more widely used in new projects and in different domains. New challenges will need to be addressed too, to meet new market demands. In this sense, companies must definitely work together with universities and research centers to help improve the usability, usefulness and applicability of MDE in industry.

6. Conclusions

As occurs with any major technology, the evolution of MDE has reached the critical point where industry needs to decide whether to adopt it or not. On the one hand, it seems an unavoidable process because the benefits and advantages seem to outweigh the costs and limitations, it is now part of our software engineers' education, and the industry is starting to successfully embrace it. On the other hand, many large companies are still reluctant to consider these changes of technology paradigms because they represent a revolution to their proven internal processes and to their businesses in general.

In this paper we have summarized the state of MDE adoption by the industry, the current difficulties for implementing it, as well as the advantages that it should bring to companies and its limitations.

Whether or not the industry eventually accepts MDE as a major technology solution for the engineering of its software systems is always difficult to predict, despite the recognizable signs currently provided by all indicators. Experience is showing that MDE is able to successfully address many of the traditional shortcomings in software development and maintenance of software systems, and to provide significant benefits and value to businesses. The key to progress now lies in all parties working together to make these new technologies available so that software applications can be more systematically, quantifiably, and predictably developed, resulting in more robust, reliable and useful software systems.

Acknowledgements

Thanks to the reviewers of this paper, and especially to David Ameller, Loli Burgueño, Jordi Cabot, Jesús García-Molina and Manuel Wimmer for their valuable comments and suggestions on earlier drafts of this work.

Cómo citar este artículo / How to cite this paper

Vallecillo, A. (2014). On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE?. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 52-68. Consultado el [dd/mm/aaaa] en www.ijisebc.com

References

a) On the use of MDE in industrial settings

[A+07] Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., Tosetti, E. Developing eBusiness Solutions with a Model Driven Approach: The Case of Acer EMEA. In Proc. of ICWE 2007, LNCS 4607, pp. 539–544, Springer, 2007.

[Avi14] Ávila-García, O. Optimización del rendimiento de aplicaciones ABAP. Novática, 228: 29-35, 2014.

[AVT06] Afonso, M., Vogel, R., and Teixeira, J. From code-centric to model-centric software engineering: practical case study of MDD infusion in a systems integration company, in Workshop on MBD/MOMPES, 2006.

[B+14] Büttner, F., Bartels, U., Hamann, L., Hofrichter, O., Kuhlmann, M., Gogolla, M., Rabe, L., Steimke, F., Rabenstein, Y., Stosiek, A. Model-driven standardization of public authority data interchange. Sci. Comput. Program. 89:162-175, 2014.

[BB14] Brambilla, M., Butti, S. Quince años de Desarrollo Industrial Dirigido por Modelos de aplicaciones Front-End: desde WebML hasta WebRatio e IFML Novática, 228: 26-43, 2014

[BC10] Bone, M., Cloutier, R. The current state of model based system engineering: results from the OMG SysML request for information 2009. In: Proc. of the 8th Conference on Systems Engineering Research (CSER), March 2010.

[BF14] Brambilla, M., Fraternali, P. Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. Sci. Comput. Program. 89:71-87, 2014.

[BLW05] Baker, P., Loh, S., and Weil,F. Model-Driven Engineering in a Large Industrial Context — Motorola Case Study. In Proc. of MoDELS 2005, LNCS 3713, pp. 476–491, Springer, 2005.

[Cab11] Cabot, J. MDD pays of in the mid-term: an industrial experiment. Post in the Modeling Languages blog, http://modeling-languages.com/mdd-pays-mid-term-industrial-experiment/, 2011

[Cab13] Cabot, J. UML adoption in practice: has anything changed in the last decade? Post in the Modeling Languages blog, http://modeling-languages.com/uml-adoption-in-practice-has-anything-changed-in-the-last-decade/, 2013.

[CGR14] Cabot, J., García-Molina, J., Rossi, G. Adopción industrial de la ingeniería del software dirigida por modelos. Novática, Núm. 228. Abril-junio 2014.

[DGHC14] Davies, J., Gibbons,J., Harris, S., Crichton, C. The CancerGrid experience: Metadata-based model-driven engineering for clinical trials. Sci. Comput. Program. 89:126-143, 2014.

[DGWC14] Davies, J., Gibbons, J., Welch, J., Crichton, E. Model-driven engineering of information systems: 10 years and 1000 versions. Sci. Comput. Program. 89:88-104, 2014.

[DMBD14] Drapeau, S., Madiot, F., Brazeau, JF., Dugré, PL. SmartEA: Una herramienta de Arquitectura Empresarial basada en las técnicas MDE. Novática, 228:21-28, 2014.

[DPP14] Di Ruscio, D., Paige, R.F., Pierantonio, A. Guest editorial to the special issue on Success Stories in Model Driven Engineering. Sci. Comput. Program. 89:69-70, 2014.

[DV10] Diaz, O., Villoria, F.M. Generating blogs out of product catalogues: An MDE approach. Journal of Systems and Software, 83(10):1970-1982, 2010.

[FL08] Forward, A., Lethbridge, T.C., 2008. Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. In: International Workshop on Models in Software Engineering (MiSE 2008), pp.27–32, ACM Press, 2008.

[HRW11] Hutchinson, J., Rouncefield, M., Whittle, J. Model-driven engineering practices in industry. In Proc. of ICSE 2011, pp. 633-642, ACM, 2011.

[HWR14] Hutchinson, J., Whittle, J., Rouncefield, M. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. 89:144-161, 2014.

[HWRK11] Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S. Empirical assessment of MDE in industry. In Proc. of ICSE 2011, pp. 471-480, ACM, 2011

[KR06] Kulkarni, V., Reddy, S. Introducing MDA in a large IT consultancy organization. In Proc. of ASPEC 2006, pp. 419-426, Dec. 2006.

[KR08] Kulkarni, V., Reddy, S. A model-driven approach for developing business applications – experience, lessons learnt and a way forward. In Proc. of 1st India Software Engineering Conference, pp. 21-28, Feb. 2008.

[KRR10] Kulkarni, V., Reddy, S., Rajbhoj, A. Scaling up model-driven engineering – experience and lessons learnt. In Proc. of MoDELS 2010, LNCS 6395, pp. 331-345, 2010.

[KTM12] Kuhn, A., Thompson, A. and Murphy, G. An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development. LNCS 7590, p. 352-367, Springer, 2012.

[LRTR13] Leotta, M., Ricca, F., Torchiano, M., Reggio, G. Empirical evaluation of UML-based model-driven techniques: Poster paper. RCIS 2013, pp.1-2, 2013

[R+06] Rios, E., Bozheva, T., Bediaga, A., Guilloreau, N. MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. In Proc. of ECMDA-FA 2006, pp. 78-89, 2006.

[M+6] Mansell, J.X., Bediaga, A., Vogel, R., Mantel, K. A Process Framework for the Successful Adoption of Model Driven Development. In Proc. of ECMDA-FA 2006, pp. 90-100, 2006.

[MCMT14] Murguzur, A., De Carlos, X., Mendialdua, X., Trujillo, S. Ingeniería del Software Dirigida por Modelos: Adopción industrial para software empotrado. Novática, 228:44-50, 2014.

[MD08] Mohagheghi, P., Dehlen, V. Where is the proof? A review of experiences from applying MDE in industry. In Proc. of MDA-FA 2008, LNCS 5095, pp. 432-443, Springer, 2008.

[MGS13] Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M., An empirical study of the state of the practice and acceptance of

model-driven engineering in four industrial cases. Empirical Software Engineering 18(1):89-116, 2013.
[N+14] András Nádas, Tihamer Levendovszky, Ethan K. Jackson, István Madari, Janos Sztipanovits: A model-integrated authoring environment for privacy policies. Sci. Comput. Program. 89:105-125, 2014.
[OMG09] The Object Management Group. Compilation of SysML RFI - Final Report. OMG Document syseng/2009-06-01 (2009)
[Pai14] Paige, R. Ingeniería de Software con modelos: Panorama actual y futuros retos. Novática, 228:11-15, 2014
[Pez14] Pezuela, C. ARTIST: Una solución global para la modernización de software hacia el cloud. Novática, 228:16-20, 2014
[PPL13] Papotti, P.E., do Prado, A.F., Lopes de Souza, W., Cirilo, C.E. and Pires, L.F. A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation. In Proc. of CAiSE 2013, LNCS 7908, pp. 321-337, Springer, 2013.
[SCG14] Sánchez Cuadrado, J., Cánovas Izquierdo, J.L., García Molina, J. Applying model-driven engineering in small software enterprises. Sci. Comput. Program. 89: 176-198 (2014)
[Sel12] Bran Selic. What will it take? A view on adoption of model-based methods in practice. Software and System Modeling 11(4):513-526, 2012.
[T+11] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G. Preliminary findings from a survey on the MD state of the practice. In Proc. of ESEM 2011, pp. 372-375, 2011.
[T+12] Tomassetti, F., Torchiano, M., Tiso, A., Ricca, F., Reggio, G. Maturity of software modelling and model driven engineering: A survey in the Italian industry. In Proc. of EASE 2012, pp. 91-100, 2012
[T+13] Telinski, L, Agner, W., Soares, I.W., Stadzisz, P.C., Simão, J.M. A Brazilian survey on UML and model-driven practices for embedded software development, Journal of Systems and Software, 86(4): 997-1005, 2013.
[T+13b] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G. Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the Italian industry. Journal of Systems and Software 86(8): 2110-2126, 2013.
[Tow14] Towers, J. Model Based Systems Engineering 'The State of the Nation'. Presentation at the Atego seminar "Realising the Benefits of Model-Based Systems Engineering", 2014. http://www.atego.com/downloads/slides/1403/1JT_The-State-of-the-Nation.pdf
[WW06] Weigert, T., Weil, F. Practical experience in using model-driven engineering to develop trustworthy systems. In Proc. of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06), pp. 208–217, 2006.

b) On the use of UML and design models in industrial settings

[A+06] Anda, B., Hansen, K., Gullesen, I., and Thorsen, H. Experiences from Introducing UML-based Development in a Large Safety-Critical Project, Emiprical Software Engineering 11:555-581, 2006.
[BBB11] Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R. Empirical evidence about the UML: a systematic literature review. Software – Practice and Experience 41(4):363–392, 2011.
[DP06] Brian Dobing and Jeffrey Parsons. How UML is used. Commun. ACM 49(5):109-113, 2006.
[FBL10] Forward, A., Badreddin, O., Lethbridge, T.C. Perceptions of software modeling: a survey of software practitioners. In: 5th Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, pp. 12–24, 2010.
[G+11] Genero, M., Fernández-Sáez, A.M., Nelson, H.J., Poels,G., Piattini, M. Research Review: A Systematic Literature Review on the Quality of UML Models. J. Database Manag. 22(3): 46-70, 2011
[GTA14] Tony Gorschek, Ewan Tempero, Lefteris Angelis, On the use of software design models in software development practice: An empirical investigation, Journal of Systems and Software, 95:176-193, 2014
[LCM06] CFJ Lange, MRV Chaudron, J. Muskens. In practice: UML software architecture and design description IEEE Software 23(2):40-46, March-April 2006
[Pet13] Marian Petre. UML in practice. In Proc. of ICSE 2013, pp. 722–731, IEEE, 2013.
[Pet14] Marian Petre. "No shit" or "Oh, shit!": responses to observations on the use of UML in professional practice. Software and Systems Modelling 13:1225–1235, 2014

c) On MDA, MDD, MDE and DSLs

[B+04] Booch, G., et al. An MDA Manifesto, in Frankel, D. and Parodi, J. (eds.) The MDA Journal: Model Driven Architecture Straight from the Masters, pp. 133-143, Meghan-Kiffer Press, 2004.
[BCW12] Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice. Morgan & Claypool, 2012.
[BVGR08] Bezivin, J., Vallecillo, A., García-Molina, J., Rossi, G. Model Driven Software Development . Special issue Novática/UPGRADE Vol. IX, No.2, 2008.
[Cab14] Cabot, J. Clarifying concepts: MBE vs MDE vs MDD vs MDA. http://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/ Last accessed: Dec 2014.
[CK08] Cook, S., Kent, S. The Domain Specific IDE. Novática/Upgrade IX(2):17-21, 2088
[FP10] Fowler, M., Parsons, R. Domain-Specific Languages. Addison-Wesley, 2010
[FR07] France, R and Rumpe, B. Model driven development of complex software: A Research Roadmap. Future of Software Engineering, IEEE, 2007.
[GMR04] García-Molina, J., Moreira, A., Rossi, G. UML and Model Engineering. Special issue Novatica/UPGRADE Vol. 5, No.2, 2004.
[Kle08] Kleppe, A. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson, 2008.
[KLRT13] Kelly, S., Lyytinen, K., Rossi, M., Tolvanen, J.P: MetaEdit+ at the Age of 20. In: Seminal Contributions to Information Systems Engineering, pp. 131-137. Springer, 2013.

[KW03] Kleppe, A. G., Warmer, J. MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Longman Publishing Co., 2003.
[Nav13] Naval Air Warfare Centre, DoD. Simulation-based Acquisition (SBA). http://www.navair.navy.mil/nawctsd/Resources/Library/Acqguide/sba.htm, 2013. Last accessed, Dec 2014.
[Sch06] Schmidt, D.C. Model-Driven Engineering. IEEE Computer 39 (2):25-31, 2006.
[Sel03] Selic, B. The Pragmatics of Model-Driven Development. IEEE Softw. 20(5):19-25, 2003.
[Sel08] Selic, B. MDA Manifestations. Upgrade. IX(2):12-16, April 2008.
[Tho04] Thomas, D. MDA: Revenge of the modelers or UML utopia? IEEE Software 21(3):22–24, 2004.
[Tol11] Tolvanen, J.P. Creating Domain-Specific Modelling Languages That Work: Hands-On. In Proc. of ECMFA 2011, LNCS 6698, pp. 393-394, Springer, 2011.
[Vol11] Voelter, M. MD*/DSL Best Practices (Update March 2011) http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf Last accessed, Dec 2014.
[Vol13] Voelter, M. DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org, 2013.
[Wat08] Watson, A. A Brief History of MDA. Upgrade IX(2):7-11, April 2008.
[WK03] Warmer, J., Kleppe, A. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman, 2003.

d) General references

[BF14] Bourque, P., Fairley, R.E. (eds.) Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org.
[Byt81] Byte Magazine. Special Issue on SmallTalk. August 1981.
[Dav89] Davis, F.D. Perceived usefulness, perceived ease of use, and user acceptance of information technology, MIS Quarterly, 13(3): 319–340, 1989.
[Gar05] Gartner, Inc. Gartner Hype Cycle of Emerging Technologies. https://www.gartner.com/doc/484424/gartners-hype-cycle-special-report, 2005. Last accessed: Dec 2014.
[Gar13] Gartner, Inc. Gartner Hype Cycle of Application Architecture, 2013. https://www.gartner.com/doc/2569522/hype-cycle-application-architecture-, 2013. Last accessed: Dec 2014.
[Gar14] Gartner, Inc. Gartner Methodologies: The Gartner Hype Cycle. http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp. Last accessed: Dec 2014.
[IEEE90] IEEE Computer Society. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1990.
[Kra07] Kramer, J. Is Abstraction the Key to Computing? Comms. of the ACM, 50:37-42, 2007.
[Moo14] Moore, G.A. Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. Harper Business Essentials, 1991 (revised 1999 and 2014).
[SZ13] Saitta, L., Zucker, J.D. Abstraction in Artificial Intelligence and Complex Systems. Springer, 2013.
[W14] Wikipedia. Comparison of Business Process Modeling Notation tools. http://en.wikipedia.org/wiki/Comparison_of_Business_Process_Modeling_Notation_tools Last accessed: Dec 2014.

© ISSN: 2387-0184

Software Engineering: Reflections on an Evolving Discipline

Ingeniería de software: Reflexiones sobre una disciplina en evolución

David Garlan[1]

[1] Professor of Computer Science and Director of Software Engineering Professional Programs. Institute for Software Research, School of Computer Science, Carnegie Mellon University. Pittsburgh, USA.

garlan@cs.cmu.edu

ABSTRACT. This paper analyzes Software Architecture, defining it and describing the evolution of this field and its role in software engineering. In addition, it covers key concepts of a software architecture course, steps to pursue an architectural thinking, the elements of organizational architecture maturity and emerging trends and issues such as: Architecture evolution, Architecture conformance, Frameworks, platforms, and ecologies, and Self-Adaptive Systems.
Further we examine how software engineering has matured over the past two decades (and the role that software architecture has played in this process), the requirements of architectural thinking (at both technical and organizational levels), the importance for an organization to have mature architectural practices and the existence of important new trends that are reshaping the way software architecture is practiced.

KEYWORDS: Software Engineering, Challenges of Software Engineering, Software Architecture, Evolution of Software Engineering, Architectural thinking, Organizational architecture maturity, Emerging trends and issues.

1. Introduction

The big problem in the Software Engineering is 'How to bridge the gap between requirements and solutions?' Software Architecture attempts to address problem by providing a layer of abstraction that supports:

- A high level of system design
- System-level abstractions and qualities
- Design reuse design through common architectural idioms.

Over the past two decades the field of software engineering has made remarkable progress, but many challenges remain.

These significant innovations include:

- Processes for making software development predictable and repeatable
- Better understanding of how to balance technical and business concerns
- Tools to improve the quality of our systems
- Techniques to master the complexity of software systems

But, in this field 'The Challenge' is to:

- Turn Software Architecture into an engineering discipline
  - o from ad hoc definition to codified principles
- Develop systems "architecturally"
  - o build systems compositionally from parts
  - o assure that the system conforms to the architecture and has the desired properties
  - o use standard integration architectures
  - o reuse codified architectural design expertise
  - o reduce costs through product lines

2. Software Architecture: past and present

2.1. What is software architecture?

There are many definitions in the literature: indeed, CMU's Software Engineering Institute's web site on software architecture lists over 100 of them (Software Engineering Institute - Carnegie Mellon University, 2014). One definition that is often used is that the Software Architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them and properties of both.

From an operational point of view issues addressed by Software Architecture include:

- Gross decomposition of a system into parts
  - o often using rich abstractions for component interaction
  - o often using common patterns/styles
- Emergent system properties
  - o performance, throughput, latencies
  - o reliability, security, fault tolerance, evolvability
- Rationale
  - o justifying architectural decisions
- Allowed change
  - o "load-bearing walls"

2.2. Evolution of the field and its role in software engineering

Figure 1 shows a brief summary of the evolution of software engineering.

Figure 1. Software Engineering Evolution.

And in this overall evolution of software engineering, Software Architecture has likewise evolved considerably. Some features of this evolution, organized by decade, include:

1980's

- Informal use of box and line diagrams
- Ad hoc application of architectural expertise
- Diverse, uncodified use of architectural patterns and styles
- No identified "architect" on most projects

1990's

- Recognition of the value of architects in software development organizations
- Processes requiring architectural design reviews & explicit architectural documentation
- Use of product lines, commercial architectural standards, component integration frameworks
- Codification of vocabulary, notations & tools for architectural design
- Books/courses on software architecture

2000's

- Incorporation of architectural notions into mainstream design languages and tools (e.g., UML-2)
- Methods based on architectural design and refinement (e.g., Model-Driven Design)
- Some architecture analysis tools

- Architectural standards for Enterprise Systems (e.g., RM-ODP, TOGAF)
- Architectural frameworks (e.g., SOA)

3. What should software engineers know about software architecture?

3.1. Elements of a course on software architecture

Any course on software architecture must contain the following elements:

General Concepts (Definition of software architecture and Basic concepts: views, styles, patterns,…)

Principles of Architecting (Understanding architectural requirements, Architecture styles and tactics, Product lines and integration frameworks, and Techniques to go from architecture to code)

Architecture in Practice [Evaluating architectural designs, Handling architectural problems, Documenting a software architecture, Presenting an architecture to others and Architecture for X (security, usability, reliability, etc.)]

3.2. Architectural thinking

Key to being an effective software architect is a set of mental perspectives and concepts, sometimes referred to as "architectural thinking". These include:

An engineering mindset: Reasoning about qualities such as Scalability, Reliability, Performance, and Cost. In particular, being able to make principled trade-off analysis when it is not possible to achieve all of these simultaneously.

Different issues for architecture & programs

Architecture - (interactions among parts, structural properties, declarative, mostly static, system-level performance, outside module boundary)

Programs – (implementations of parts, computational properties, operational, mostly dynamic, algorithmic performance, inside module boundary)

Styles, Platforms, and Product Lines (Figure 2)

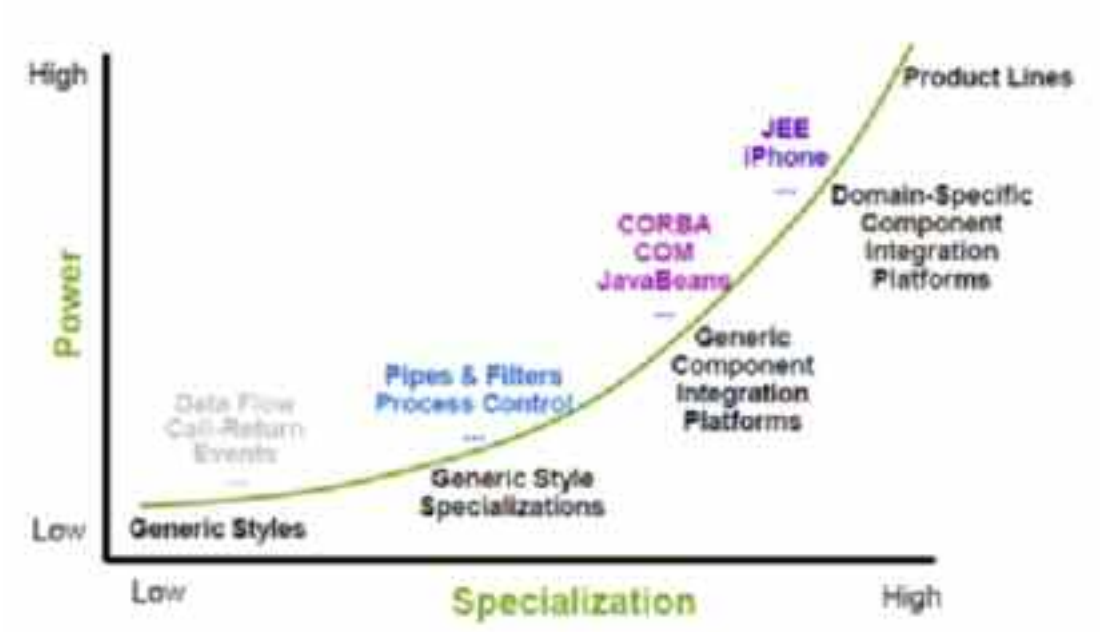


Figure 2. Styles, Platforms, and Product Lines in software architecture.

3.3. Organizational architecture maturity

Organizations differ considerably in their architectural practices. Key features that indicate architectural maturity include:

- Software architects are explicitly recognized and rewarded (Not everyone can become a software architect)
- The company has architecture training (Basic training for all entering software engineers, and advanced courses for designated architects)
- Architecture reviews are a requirement (No project can be approved without a review)
- The company maintains architecture case history, checklists, and guidance documents (Allows corporate knowledge to be maintained)
- The company uses and maintains product lines (Requires both technical and organizational changes)
- Architecture documentation standards are defined and followed (It is not enough to say "We use UML")

4. Emerging trends and issues

4.1. Architecture evolution

Businesses must evolve their architectures from A to C, through a series of incremental architectures. Examples include migrating batch-oriented systems to web-based interactive system; or migrating client-server systems to service-oriented architectures (SOA).

An important issue is how do we approach this problem: Can we leverage past evolution histories? How does this problem link to project planning, cost estimation, work assignments, etc?

4.2. Architecture conformance

We would like to make sure that the implementation conforms to architecture (and vice versa).But the issue

is what does it mean to "conform" and how would we evaluate its satisfaction.

4.3. Frameworks, platforms, and ecologies

We have been building on top of platforms and using software frameworks for most of the history of software engineering. The nature of such platforms has evolved.

But the issue is how to create ecosystems that allow a platform to be sustainable. This one requires a deep understanding of context: economic, legal, organizational, human motivation, user communities,…

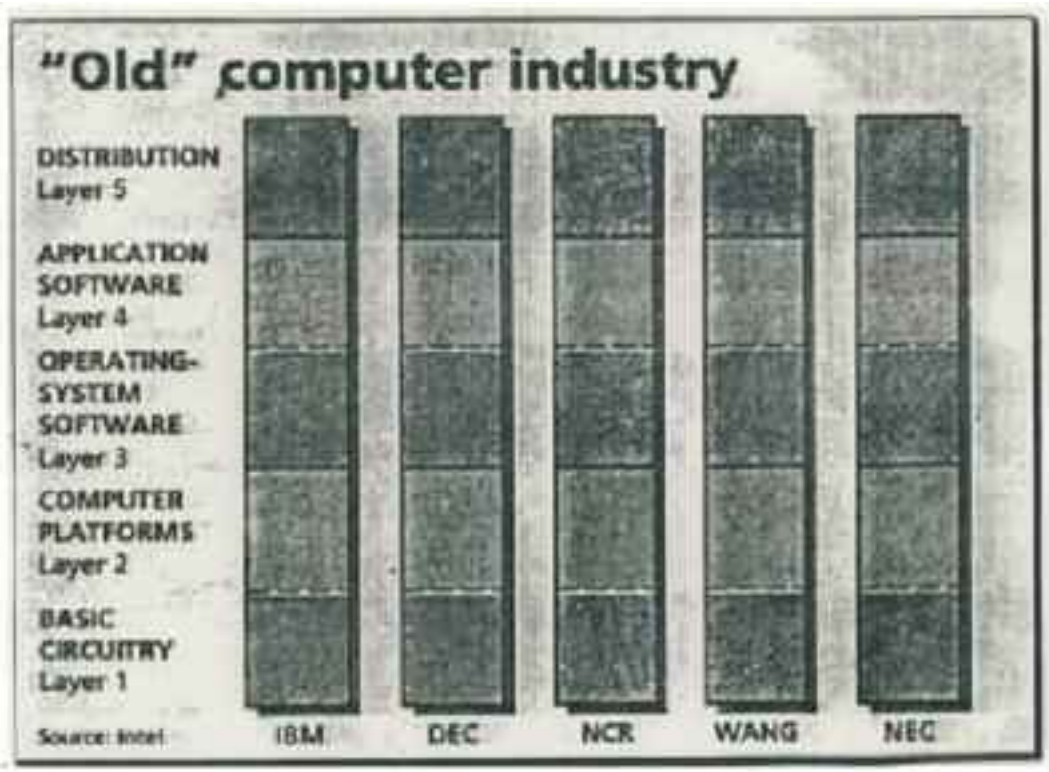


Figure 3. Old structure of the computer industry. Source: Reprinted from The Economist, Feb 27, 1993.

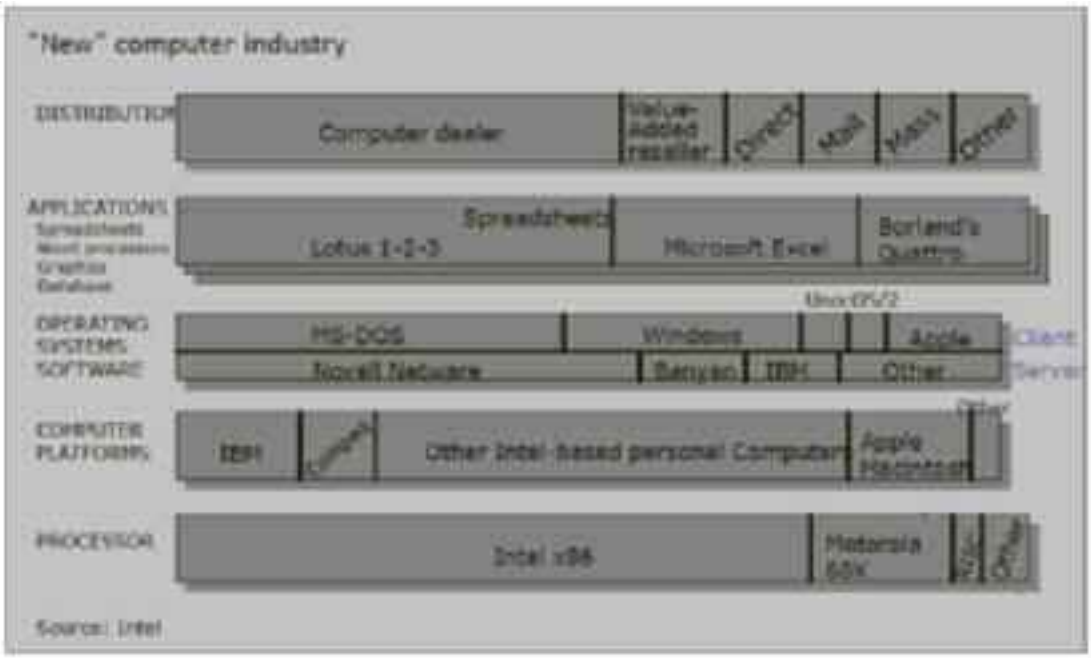


Figure 4. New structure of the computer industry. Source: Reprinted from The Economist, Feb 27, 1993.

Today, there are many new architectural ecosystems:

Architectures enable new ecosystems. Examples: Windows, iPhone, Java EE, Eclipse, Robotics OS, VISTa, SOA

Architectures introduce new roles into the ecosystem. Examples: governance bodies, platform developers and maintainers

Failure to understand how architectures and ecosystems interact can lead to failure. Example: JavaPhone

4.4. Self-Adaptive Systems

Systems must have high availability today and the use of human operators is expensive and error-prone. Hence, systems must automatically adapt to: failures, changing environmental conditions, changing requirements and threats. Examples of adaptation operations: server reboot, reduce fidelity to accommodate high load, and block an intruder. But the challenge is how to control this.

One approach to this problem is exemplified by the Rainbow System. Key features of that system are:

- Uses system models at run time as a basis for self-healing (monitoring; problem detection; repair)
- Supports new reasoning techniques for self-adaptive systems (determining "best" repair strategy and analyzing soundness of repairs)
- Can be used to achieve self-securing systems (both reactive and proactive).

5. Conclusions

In conclusion, the principal lessons to take away are:

- Software engineering has matured over the past two decades
- Software architecture has played a major role in this process
- We now understand what it means for an organization to have mature architectural practices
- This requires "architectural thinking" at both technical and organizational levels
- There are important new trends that are reshaping the way software architecture is practiced

Indeed, Software Architecture is a science that today that has substantial practical impact, but remains the focus of much research andcontinues to evolve as technology drives the field.

Cómo citar este artículo / How to cite this paper

Garlan, D. (2014). Software Engineering: Reflections on an Evolving Discipline. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 70-77. Consultado el [dd/mm/aaaa] en www.ijisebc.com

References

The Economist (1993, Feb 27). Structure of the computer industry. The Economist.
Software Engineering Institute - Carnegie Mellon University (2014). Retrieved November 15, 2014, from http://www.sei.cmu.edu/

25 Challenges of Semantic Process Modeling

25 Desafíos de la Modelación de Procesos Semánticos

Jan Mendling[1], Henrik Leopold[2], Fabian Pittke[1]

[1] Institute for Information Business, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria
[2] Department of Computer Sciences, Vrije Universiteit Amsterdam, Faculty of Sciences, De Boelelaan 1081, 1081HV Amsterdam, The Netherlands

jan.mendling@wu.ac.at, h.leopold@vu.nl, fabian.pittke@wu.ac.at

ABSTRACT. Process modeling has become an essential part of many organizations for documenting, analyzing and redesigning their business operations and to support them with suitable information systems. In order to serve this purpose, it is important for process models to be well grounded in formal and precise semantics. While behavioural semantics of process models are well understood, there is a considerable gap of research into the semantic aspects of their text labels and natural language descriptions. The aim of this paper is to make this research gap more transparent. To this end, we clarify the role of textual content in process models and the challenges that are associated with the interpretation, analysis, and improvement of their natural language parts. More specifically, we discuss particular use cases of semantic process modeling to identify 25 challenges. For each challenge, we identify prior research and discuss directions for addressing them.

KEYWORDS: Process modeling, Formal semantics, Natural language processing, System analysis and design.

1. Introduction

Process models play an important role in various application scenarios that relate to system analysis and design [Wes12, DRMR13]. They often serve as a specification to bridge between business requirements and workow implementation. Process models have been intensively studied in terms of their behavioural properties, for instance on the basis of formalisms such as Petri nets, automata, labeled transition systems or temporal logic, to name but a few [van00].

Compared to the extensive stream of research into behavioural semantics, it is surprising to observe that the textual content of process models has received by far less attention. This fact reects a painful gap in current research since the domain understanding of process models builds the more on its textual content the less the persons creating and reading the models have a formal education in computer science. On the one hand, it is often casual modelers from the line of business that work with models [Ros06], and these tend to pay little attention to behavioural semantics. On the other hand, their model understanding strongly depends on the appropriate formulation of text labels in the process model and their accurate interpretation [MRR10].

The aim of this paper is to make this identified research gap more transparent. To this end, we define a modeling language with an explicit reference to its textual content and describe the interpretation of text on the three levels of the single label, the model fragment and the whole model collection. We use these three levels to organize 25 challenges of semantic process modeling. These 25 challenges relate to the various tasks that involve the interpretation, analysis and improvement of text labels in a process model. In this way, it complements prior research on tasks and use cases as identified for business process modeling and process mining [vdA13], change patterns [WRR08] and refactorings [WRMR11].

The paper is structured as follows. Section 2 describes the setting in which process models are created and their different components. Section 3 identifies challenges of working with label. Section 4 describes challenges of working with textual labels on the level of a model fragment or a whole model. Section 5 describes challenges in the relation to the management of an overall model collection and its textual content. Section 6 discusses the challenges before Section 7 concludes the paper.

2. Background

Process modeling plays an important role in various areas of system analysis and design. Specifically business process modeling was identified as one of the most prominent applications of conceptual modeling altogether [DGR+06]. Modeling techniques are typically used for creating models of good quality. The different components of a modeling technique are illustrated in Figure 1.

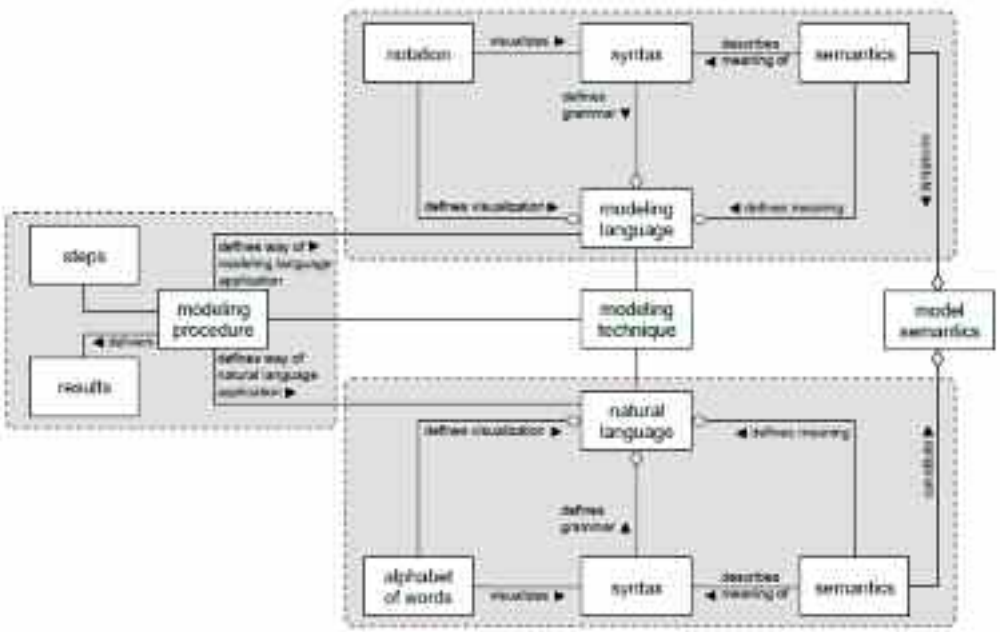


Figure 1. Syntax and Semantics of Process Models [Leo13].

Classically, a modeling technique has been considered to consist of two interrelated parts: a modeling language and a modeling procedure [Men08]. The modeling language consists of three parts: syntax, semantics and a notation. The syntax defines a set of elements and a set of rules how these elements can be combined. A synonym is modeling grammar [WW90, WW95, WW02]. Semantics bind these elements to a precise meaning. For process model, behavioural semantics are often defined using Petri net concepts [LVD09]. The notation defines a set of graphical symbols that are utilized for the visualization of models [Moo09]. The modeling procedure defines steps by which a modeling language can be used [WW02, DRMR13]. The result of applying the modeling procedure is a model that complies with a specific modeling language.

Recent research has extended this classical conceptualization with a more explicit specification of the textual parts of models. Therefore, Figure 1 shows the natural language part as a separate component. The terminology used in the models is defined by the alphabet of words while the syntax is defining the rules of building text fragments that are permissible for the specific type of model [Leo13]. For instance, the activity label of a process model is typically assumed to contain a verb and a business object [LESM+13]. The semantics in this context refer to the precise interpretation of the words used in the label.

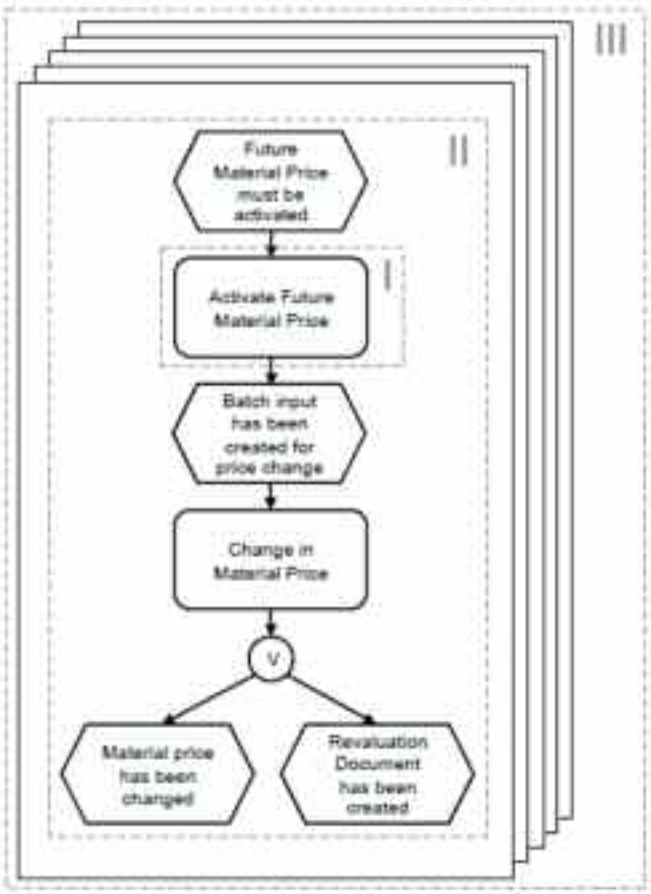


Figure 2. Three Levels of Semantic Process Modeling.

Extending the perspective of process modeling towards the explicit discussion of natural language components is promising specifically for applications that require to analyze both behavioural and textual semantics, such as process model matching [WDM10], process model reuse [KFSO14], service identification [LM12], or model translation [BESL+13]. On the other hand, this more integral perspective on conceptual modeling reveals various challenges.

In the following sections, we aim to describe tasks and corresponding challenges. We organize them into three categories that are based on the extent of their textual content (see Figure 2).

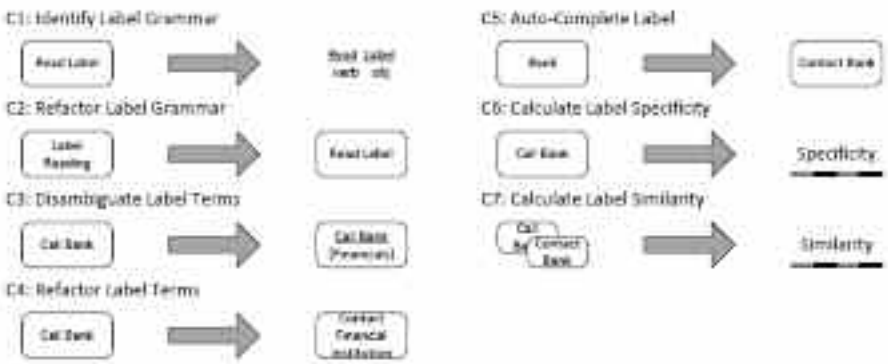


Figure 3. Challenges in Relation to Labels.

The first category relates to labels and their analysis. The second category describes analysis onvthe level of whole models or model fragments. Finally, the third category discusses challenges on the level of whole model collection. Each challenge is structured accordingly. We discuss each challenge by clarifying the goals and the necessary input information of the associated task. Based on that, we further specify the challenges linked to a particular task and illustrate them with the help of small examples. Finally, we conclude with a short summary of prior research and explain how the respective challenge has been addressed with conceptual or technical solutions.

3. Label Challenges

In this section, we describe various challenges on analyzing and reworking labels of elements that appear in a process model. Figure 3 gives an overview.

C1: Identify Label Grammar. The goal of this task is the automatic identification of the semantic components of a process model element label. The input for this task is an element label and, if applicable, the process model and the process model collection the label is part of.

The challenge of this task is the proper recognition of the various and potentially ambiguous grammatical label structures. It is further complicated by the shortness of element labels and the fact that they often do not represent proper sentences. As a result, it is dificult to always identify the correct part of speech of label terms. As an example, consider the label “plan data transfer", which may refer to the “planning" of a “data transfer" or the “transfer" of “plan data". Prior research has approached this challenge by describing grammatical styles of labels and defining corresponding parsers [LSM10]. Ambiguity can be resolved based on the inclusion of further contextual and external knowledge [LESM+13]. Besides the recognition of the label grammar, the resulting techniques can also be used for checking the compliance with a grammatical guideline [LESM+13, BDH+09, DHLS09].

C2: Refactor Label Grammar. The goal of this task is to refactor the existing grammar of a particular label to a more desirable grammatical style. The input for this task is the label and its previously identified semantic components.

The challenges in the context of this task include lemmatization, i.e. deriving the base form from an inected word, as well as the proper recognition of compound words. As an example, consider the label “new user registration". For refactoring this label into the widely requested verb-object style [Sil11, MRR10, MRvdA10], we first need to transform the nominalized action “registration" into to the verb “register". Second, we have to recognize that the adjective “new" refers to the “user" and not to the entire “user registration". As a result, we obtain the refactored verb-object label “register new user". Prior research has approached these challenge by building on WordNet and a number of structural heuristics [LSM12].

C3: Disambiguate Label Terms. The goal of this task is to recognize the meaning of a term from a process model element label. The input for this task is a label term including its context, i.e., the label it belongs to and, if applicable, the model and the process model collection the label is part of.

The challenge of this task is to identify the correct meaning of a word despite the limited context that is provided by process model element labels. As an example, consider the label “check application". Depending on the context, the word “application" could refer to a “job application" as well as a “computer application". Prior research has approached this challenge by selecting the most probable meaning from lexical databases such as WordNet [PLM13] or BabelNet [PLM15] based on the label context.

C4: Refactor Label Terms. The goal of this task is to replace syntactically identical words with different meanings (homonyms) and syntactically differing words with the same meaning (synonyms) with unambiguous alternatives. The input for this task is a label term including its context and the previously identified meaning

of that label term. The challenge of this task is to identify un-ambiguous and suitable alternatives for the considered homonymous or synonymous term. As an example, consider the homonym "application". Depending on the context, the word "application" may be, for instance, replaced with "job application". In case synonyms such as "invoice" and "bill", a choice for the most suitable word must be made. Prior research has approached this challenge by building on the meanings and the context information from the lexical database BabelNet [PLM15].

C5: Auto-Complete Label. The goal of this task is to automatically provide useful suggestions for completing an incomplete label. The input for this task is an incomplete label, for instance, only consisting of a business object combined with further context information, such as the process model or the process model collection.

The challenge of this task is to recognize the context of a label, to generate suitable completion candidates, and to rank them according to their relevance. As an example, consider the label "bank", which only consists of a business object. An automated technique would be required to analyze the context and to propose a suitable action such as "contact" or "call". Prior research has approached this problem by building on existing process knowledge [CHSB13].

C6: Calculate Label Similarity. The goal of this task is to obtain a (realistic) similarity value between 0 and 1 for two given process model element labels. The input for this task are two process model element labels. If required, additional information such as the previously derived semantic components may complement the labels.

The challenge of this task is to identify means that facilitate the realistic measurement of the semantic similarity of two labels. The task is complicated by the specificity of many terms that are used in process models as well as different levels of granularity. As an example, consider the two labels "check application documents" and "evaluate CV". Apparently, the second label is a sub task of the first. However, it represents already a challenge to properly quantify the similarity between "document" and "CV". Prior research has approached this challenge by computing and aggregating the Lin similarity among the words or the semantic components of the two labels [CDD+13]. Non-semantic approaches based on the Levenshtein distance have been, for example, proposed in [EKO07, DDvD+11].

C7: Calculate Label Specificity. The goal of this task is to quantify the specificity of a given process model element label. The input for this task is a process model element label and, if required, its semantic components.

The challenge of this task is to identify suitable means for measuring the specificity of the label terms as well as the label as a whole. Particularly challenging are labels which contain words that cannot be found in lexical databases such as WordNet. As an example, consider the label "call customer service hotline". The specificity of the term "hotline" can be, for instance, determined based on the position of the word in the WordNet taxonomy. However, this is not possible for the term "customer service hotline" as this term is not part of the WordNet database. Prior research has approached this challenge by using on WordNet [Fri09, KB07] and other heuristics such as label length and the number of semantic components [LPM13].

4. Model Challenges

In this section, we describe various challenges on analyzing and reworking semantic fragments of process models. Figure 4 gives an overview.

C8: Discover Label Mapping. The goal of this task is to map a phrase to a text label. The input for this task is a process model with its text labels and a piece of text containing several phrases.

The challenge of this task is to identify the activity in the process model, which is semantically the closest

to a text sentence. As an example, consider a natural language text containing the sentence "he ordered the book" and a process model containing activities with the labels "send invoice" and "order book". Prior research has approached this task as a classifier problem. The tool SemTalk helps to maintain consistency between labels and separate concepts [FWAW03].

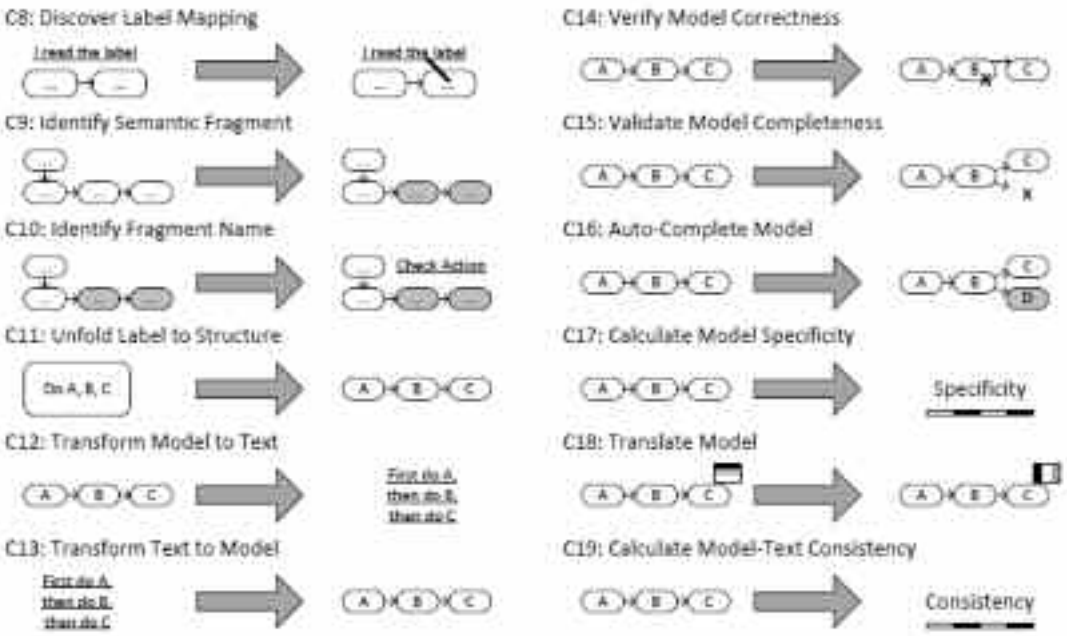


Figure 4. Challenges in Relation to Models.

C9: Identify Semantic Fragment. The goal of this task is to identify a fragment of a process model that is semantically closely related. The input for this task is the model and the labeled activities.

The challenge of this task is to determine those activities that are related and can be described as a whole on a more abstract level. As an example, consider the activities "receive order" and "check order". Together, these are activities that both relate to the handling of orders. Prior research has approached this challenge by different approaches on process model abstraction. One approach uses semantic relations such as meronymy [SDMW10] and different notions of distance [RMD11, SRW12]. Various abstraction scenarios are summarized in [SRWN12].

C10: Identify Fragment Name. The goal of this task is to identify the name of a set of activities that describe them at a more abstract level. The input for this task is a process fragment containing the set of activities.

The challenge of this task is to find a name for this fragment that captures its content in a semantically meaningful way. Also, the name of activities can be defined from different perspectives, e.g. what is being done or what is supposed to be achieved. As an example, consider again the "activities "receive order" and "check order". A technique for naming this fragment should propose a label like "handle order". Prior research has approached this challenge by describing different strategies for defining a name of a fragment or a whole process based on theories of meaning such that different proposals can be derived automatically [LMRR14].

C11: Unfold Label to Structure. The goal of this task is to decompose a label into different activities and to transform this into a corresponding fragment of a process model. The input for this task is an activity label that describes more than just a single activity.

The challenge of this task is to identify that several activities are described and which structure can best

capture their semantics. As an example, consider a single activity label "receive and check order". Apparently, the single label refers to two activities which might be executed in parallel or sequential order. Prior research has approached this challenge by identifying commonalities in process model collections and deducting regular anti patterns that incorporate several activities in one activity label [PLM14].

C12: Transform Model to Text. The goal of this task is to transform a process model into a natural language process description. The input for this task is a process model along with the semantic component annotations of its elements.

The challenge of this task is to present the non-sequential structure of a process model in a sequential fashion. In addition, the text should be as natural as possible. As an example, consider the sequence of the activities "receive order", "check order", and "send products" in a process model. A technique to transform the model fragment into text should create a text fragment like "The process begins with the receipt of an order. After the order is checked, the products are sent to the customer". Prior research has approached this challenge by proposing a technique that automatically generates a textual representation of a given process model based on the refined process structure tree and the meaning text theory [LMP14]. Template-based approaches have been proposed in [Cos10, MB13].

C13: Transform Text to Model. The goal of this task is to elicit a process model from a natural language process description. The input for this task is a piece of text.

The challenge of this task is to properly discover the activities as well as the order of activities including decisions, concurrency, and loops. As an example, consider the text "The kitchen prepares the meal. In the meantime, the waiter takes care of the beverages". An automated technique would have to recognize the roles "kitchen" and "waiter", the activities "Prepare meal" and "Take care of beverages" as well as the fact that the two activities are conducted in parallel. Prior research has approached this challenge by applying standard natural language processing techniques and a number of signal words and phrases [FMP11, dAGSB09, GKC07, SP10].

C14: Verify Model Correctness. The goal of this task is to check whether a process model is correct according to the semantics defined by its activity labels. The input for this task is the process model together with semantic annotations for the activity labels.

The challenge of this task is to identify those activities that significantly inuence the control-flow of the process model and validate if the control-flow matches the semantics of the label. As an example, consider the activity label "assess application" that requires an application has been checked for completeness. Therefore, there has to be a prior activity that guarantees this requirement to be fulfilled. Prior research has approached this challenge by propagating preconditions and effects over the process model for semantic verification [WHM10] or by building on linguistic knowledge [vdVGvdR97, GL11]. Correctness by design is provided by approaches using automatic planning of business processes [HLD+05].

C15: Validate Model Completeness. The goal of this task is to check whether a process model is correct according to the semantics defined by its activity labels. The input for this task is the process model together with semantic annotations for the activity labels.

The challenge of this task is to identify those activities that significantly inuence the control-flow of the process model and validate if the control-flow matches the semantics of the label. As an example, consider the activity label "assess application" that may either result in an approval or a rejection. For the sake of semantic model consistency, the application cannot be accepted and rejected at the same time and thus demands an exclusive decision after the activity. Prior research has approached this challenge by using semantic error patterns, for instance based on antonyms [GL11]. The approach in [TF07] discusses the opportunities of using semantic web technologies to reason about process models. Validation in the context of customization of

process variants is discussed in [DB13, GBPG13, AMGG14].

C16: Auto-Complete Model. The goal of this task is to provide user-assistance during process modeling and to avoid typographical and syntactical errors in process models. The input for this task is a set of process models from which suggestions to complete the process are created.

The challenge of this task is the definition of learning recommendation system that suggests a list of meaningful process fragments that may be entered at this current position within the process model. As an example, consider again the sequence of the activities "receive order", "check order". A recommendation system might suggest a XOR-split with the activity "send products" if the check is successful "inform customer" if the check fails. Prior research has approached this challenge by using business rules and structural constraints to propose appropriate process fragments [HKO07].

C17: Calculate Model Specificity. The goal of this task is to identify and adjust element labels according to their level of detail within a hierarchy of process models. The input for this task is a process model as well as its position in a process hierarchy or a process architecture.

The challenge of this task is to measure the concept of specificity and to recommend actions to adjust element labels that do not comply to the level of detail within the process hierarchy. As an example, consider the a sequence of activities "receive order", "check purchase order", and "send products" which describes the handling of an incoming order on a general level. Apparently, the second activity is too specific as it entails a particular type order that needs to be checked. Prior research has approached this challenge by providing a set of syntactical and semantic metrics that measure the granularity of element labels [LPM14].

C18: Translate Model. The goal of this task is to overcome the language barrier for re-using of process models in multi-national companies. The input for this task is a process model in a particular language.

The challenge of this task is dealing with the short texts in labels, recognizing the context of the process model, and appropriately translating the process model into the target language. As an example, consider the activity "receive order". If we consider a translation of this activity, the translation system should be capable to recognize that the word order is used in the sense of a commercial document and not in the sense of a military command and thus chose the appropriate translation. Prior research has approached this challenge by developing a technique for the automated translation of business process models that builds upon statistical machine translation and word sense disambiguation [BESL+13].

C19: Calculate Model-Text Consistency. The goal of this task is to measure the consistency between a process description as a process model and as a natural language text and to identify notable differences between these descriptions. The input for this task is the process model together with a textual process description.

The challenge of this task is again defining abstract representation to map the content of both text and model and identifying deviations of both types. As an example, consider a sequence of the activities "receive order", "check order", and "send products" as well as the text fragment "After the order is received, the respective products are send to the customer". Apparently, the textual description is not consistent to the activity sequence because one activity is missing in the textual description. Prior research has approached this challenge by translating a textual description into process models resolving arising inconsistencies either in an automated or mediated manner [GKC07].

5. Collection Challenges

In this section, we describe various challenges on analyzing and reworking semantic fragments of process models. Figure 5 gives an overview.

C20: Discover Model Mapping. The goal of this task is to discover a mapping between the sets of activities of two process models. The input for this task is a pair of process models and a similarity matrix over the pairs of activities.

The challenge of this task is that activities are potentially described on different levels of granularity such that not only 1:1, but also 1:n and n:m matches are possible. As an example, consider the coarse-granular activity "build car" in one model and the sequence of "purchase parts", "assemble parts", and "check car" in a second model. Prior research has approached this challenge by using concepts from ontology matching [WDM10]. These have been extended towards using constraints to reduce the search space [LNW+12] and including feedback [KLW+]. A comparison of different techniques is reported in [CDD+13].

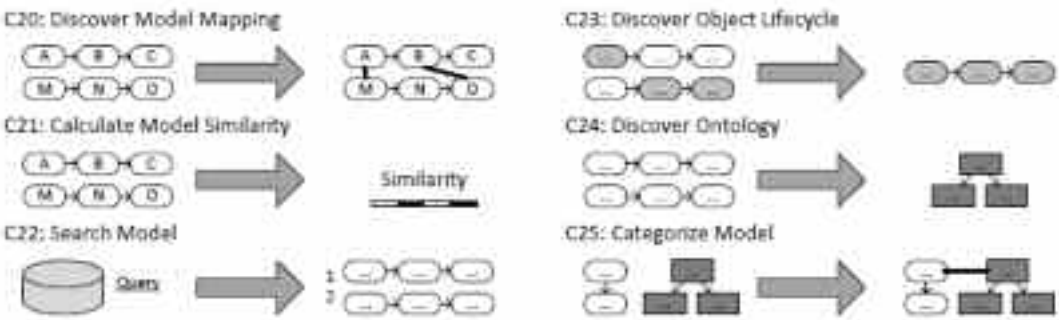


Figure 5. Challenges in Relation to Collections.

C21: Calculate Model Similarity. The goal of this task is to determine how similar process models are. The input for this task is a pair of process models and a mapping between their activities.

The challenge of this task is to consider adequately different aspects of representational heterogeneity including labels, structure and behaviour. For example, there are different ways to model the fact that both activities A and B are executed or just one of them. Models can be trace equivalent, but have different structure. Prior research has approached this challenge by defining behavioural abstractions. The behavioural profile [WMW11], transition adjacency [ZWW+10] and matrix relations [ABDG14] define behavioural relations over the cartesian product of activities. The matrices of two models can then be compared cell-wise [DvDD+13]. As an alternative, graph edit distance can be used [DGD09]. Similar approaches are defined in [EKO07, EG07, CGB06]. A comparison of approaches is reported in [DDvD+11].

C22: Search Model. The goal of this task is to rank process models of a collection according to how similar they are to a given search query. The input for this task is a search query and a collection of process models.

The challenge of this task is identify those features that are supposedly relevant for calculating the semantic distance between the query and each of the process models. As an example, consider a query containing the term "Human Resources". A suitable technique would be able to identify also models that do not contain this term, but also those that contain related terms such as "employee" or "contract". Prior research has approached this challenge by building on WordNet [APW08] and language modeling [QAR11]. Alternatively, query languages such as PQL [KB04] and BPMN-Q [ADW08], as well as indexing [YDG12, JWW+10] or clustering techniques [QAR11, RMKL12]. Also, behavioral profiles are used to search for models [KWW11].

C23: Discover Object Lifecycle. The goal of this task is to discover the lifecycle of objects from the activities described in a collection of process models. The input for this task is a collection of process models and the semantic annotation of the activity labels.

The challenge of this task is to integrate the parts of the lifecycle, which might be scattered over several models. For example, consider one model including the activities “receive order" and “check order" and a second model with the activities “check order" and “confirm order". Prior research has approached this challenge by identifying action patterns between activity pairs [SWMW12], which can be synthesized to lifecycle models of the respective business objects [SWM12]. Based on these lifecycle models, compliance between process models and object lifecycle can be discussed [KRG07].

C24: Discover Ontology. The goal of this task is to discover a formal ontology from a collection of process models. The input for this task is a collection of process models and the semantic annotation of the activity labels.

The challenge of this task is to extract pieces of information that can be used for identifying formal concepts and relationships. As an example, consider decomposition relationships between process models and semantic groupings that are not explicitly defined. Prior research has approached this challenge for building taxonomies [PW11].

C25: Categorize Model. The goal of this task is to identify a category in which a particular model fits best. The input for this task is a process model and a taxonomy, which is in the simplest case a set of categories.

The challenge of this task is that category descriptions might contain only a few terms and that process models might include tasks that relate to several categories. As an example, consider PCF Taxonomy, which contains 1131 hierarchically organized concepts. Prior research has not addressed this challenge in detail. Promising directions include the extension of existing approaches for semantic annotation of process models [FT09, LD05, BSPW08, BDW07?]. There is also work that identifies categories inductively from the models [MDM13].

6. Discussion

This section discusses the state of current research on semantic business process modeling based on the challenges identified above. At this stage, it has to be noted that the merits of these challenges should not seen in terms of a claim for completeness - indeed, it is unclear whether it is feasible to provide a complete list of challenges at all. The benefits of this compilation has to be seen much more in its capability of separating well-researched areas from topics that have received little attention so far. Therefore, we want to structure this discussion along the following lines: tasks that we observe to be well-researched, tasks that call for more research, and base techniques that could help to advance semantic process modeling.

Among the well-researched tasks, we regard the identification and refactoring of label grammar (C1 and C2), the calculation of similarity (C7 and C21), the identification of a semantic fragment (C9), and the search for particular models (C22) as mature tasks. Approaches addressing tasks of C1 and C2 perform well with real-world data and have a high accuracy in processing the labels. A similar observation can be made for approaches of label similarity (C7) and model similarity (C21). In particular for the latter, research approaches have incorporated the element labels, the model structure, and the model behavior as relevant aspects of model similarity and proposed several metrics for its calculation. With regard to the identification of semantic fragments (C9) and to the search of process models (C22), we identify a considerable number of approaches covering several requirements with regard to these tasks. Thus, we also conclude that these tasks are well understood and supported by recent approaches.

Turning to the tasks that require more research, we want to highlight the tasks that relate to the specificity of labels (C6), the alignment of text and model (C12, C13, C19), and the ontology- related tasks (C24, C25). As outlined before, specificity-related tasks try to adjust the label components depending on their level of detail within a process model landscape. In such as setting, finding the appropriate level of granularity is still an open challenge [DVR11] and despite prior efforts not addressed in sufficient detail. Regarding the alignment of mod-

els and text, we observe that non-analysts increasingly work with process models and require a solid understanding of the underlying process. In order to support non-analysts in understanding and problem-solving tasks with reference to the process at hand, textual descriptions of the processes are maintained as a complement to the process models process models. However, we observe a notable gap of approaches that provide an alignment of process models and textual descriptions. Similarly, we find approaches for integrating ontologies with process models [HLD+05, HR07]. While the creation of ontologies is work-intensive, difficult and often domain-specific [PSFGP10], it would be desirable to support these task in an automatic fashion. We identified only a very small number of approaches that address this challenge. Thus, we call for more approaches to learn ontologies from process models and to link process models to existing ontologies or taxonomies.

The challenges also revealed several base techniques from which existing solutions of semantic process modeling would potentially benefit. Among them, we identify the integration of text corpora, such as Wikipedia or related repositories, as well as the and extending the set of semantic relationships as most promising. The integration of large text corpora and corpus-based techniques might be a suitable direction for working around the limitations of general purpose databases like WordNet in terms of its vocabulary. The rich spectrum of semantic relationships might support the discovery of an ontology as well as the search and categorization of process models. So far, only a limited amount of semantic relations have been used. Specifically, homonym and synonym relations have been used to correct ambiguous terminology in process models, while meronym relations have been proven as useful to find semantic fragments. However, there are still semantic relations left might support specific tasks. Future research should consider the usage of a broader range of semantic relationships, including hypernyms, hyponyms, meronyms, holonyms, antonyms, or troponyms.

7. Conclusion

In this paper, we shed light onto the challenges that relate to the analysis of the textual content in process models. We identified a number of 25 challenges that arise when dealing with the textual content on the level of a single label, on the level of a process model and on the level of a process model collection. For each challenge, we identified necessary input information, further specified the challenge with the help of examples, and explain how related work has addressed the challenge so far. In light of these challenges we hope to increase the interest and the awareness of future research streams towards the textual content of process models. We expect our list of challenges to help in positioning current research activities and in fostering innovative ideas to address the identified gaps.

Cómo citar este artículo / How to cite this paper

Mendling, J., Leopold, H., & Pittke, F. (2014). 25 Challenges of Semantic Process Modeling. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 78-94. Consultado el [dd/mm/aaaa] en www.ijisebc.com

Referencias

[ABDG14] Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Behavioral comparison of process models based on canonically reduced event structures. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, volume 8659 of Lecture Notes in Computer Science, pages 267-282. Springer, 2014.

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings, volume 5240 of Lecture Notes in Computer Science, pages 326-341. Springer, 2008.

[AMGG14] Mohsen Asadi, Bardia Mohabbati, Gerd Gröner, and Dragan Gasevic. Development and validation of customized process models. Journal of Systems and Software, 96:73-92, 2014.

[APW08] Ahmed Awad, Artem Polyvyanyy, and Mathias Weske. Semantic querying of business process models. In Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE, pages 85-94. IEEE, 2008.

[BDH+09] J. Becker, P. Delfmann, S. Herwig, L. Lis, and A. Stein. Towards Increased Comparability of Conceptual Models - Enforcing Naming Conventions through Domain Thesauri and Linguistic Grammars. In ECIS 2009, June 2009.

[BDW07] M. Born, F. Dörr, and I. Weber. User-Friendly Semantic Annotation in Business Process Modeling. In WISE 2007 Workshops, volume 4832 of LNCS, pages 260-271. Springer, 2007.

[BESL+13] Kimon Batoulis, Rami-Habib Eid-Sabbagh, Henrik Leopold, Mathias Weske, and Jan Mendling. Automatic business process model translation with bpmt. In Advanced Information Systems Engineering Workshops, pages 217-228. Springer, 2013.

[BSPW08] Andreas Bögl, Michael Schre, Gustav Pomberger, and Norbert Weber. Semantic annotation of epc models in engineering domains to facilitate an automated identification of common modelling practices. In ICEIS, pages 155-171, 2008.

[CDD+13] Ugur Cayoglu, Remco M. Dijkman, Marlon Dumas, Peter Fettke, Luciano García-Banñuelos, Philip Hake, Christopher Klinkmüller, Henrik Leopold, André Ludwig, Peter Loos, Jan Mendling, Andreas Oberweis, Andreas Schoknecht, Eitam Sheetrit, Tom Thaler, Meike Ullrich, IngoWeber, and Matthias Weidlich. Report: The process model matching contest 2013. In Lohmann et al. [LSW14], pages 442-463.

[CGB06] Juan Carlos Corrales, Daniela Grigori, and Mokrane Bouzeghoub. BPEL processes matchmaking for service discovery. In Robert Meersman and Zahir Tari, editors, On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I, volume 4275 of Lecture Notes in Computer Science, pages 237-254. Springer, 2006.

[CHSB13] Nico Clever, Justus Holler, Maria Shitkova, and Jörg Becker. Towards auto-suggested process modeling-prototypical development of an auto-suggest component for process modeling tools. In EMISA, pages 133-145, 2013.

[Cos10] Ahmet Coskuncay. An Approach for Generating Natural Language Specifications by Utilizing Business Process Models. Master's thesis, Middle East Technical University, 2010.

[dAGSB09] Joao Carlos de A.R. Goncalves, Flavia Maria Santoro, and Fernanda Araujo Baiao. Business process mining from group stories. International Conference on Computer Supported Cooperative Work in Design, pages 161-166, 2009.

[DB13] Wassim Derguech and Sami Bhiri. Business process model overview: Determining the capability of a process model using ontologies. In Witold Abramowicz, editor, Business Information Systems - 16th International Conference, BIS 2013, Poznan, Poland, June 19-21, 2013. Proceedings, volume 157 of Lecture Notes in Business Information Processing, pages 62-74. Springer, 2013.

[DDvD+11] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of Business Process Models: Metrics and Evaluation. Information Systems, 36(2):498-516, 2011.

[DGD09] Marlon Dumas, Luciano García-Bañuelos, and Remco M. Dijkman. Similarity search of business process models. IEEE Data Eng. Bull., 32(3):23-28, 2009.

[DGR+06] Islay Davies, Peter Green, Michael Rosemann, Marta Indulska, and Stan Gallo. How do practitioners use conceptual modeling in practice? Data & Knowledge Engineering, 58(3):358-380, 2006.

[DHLS09] A. Delfmann, S. Herwig, L. Lis, and A. Stein. Supporting Distributed Conceptual Modelling through Naming Conventions - A Tool-based Linguistic Approach. Enterprise Modelling and Information Systems Architectures, 4(2):3-19, 2009.

[DRMR13] M. Dumas, M.L. Rosa, J. Mendling, and H. Reijers. Fundamentals of Business Process Management. Springer, 2013.

[DvDD+13] Remco M. Dijkman, Boudewijn F. van Dongen, Marlon Dumas, Luciano García-Bañuelos, Matthias Kunze, Henrik Leopold, Jan Mendling, Reina Uba, Matthias Weidlich, Mathias Weske, and Zhiqiang Yan. A short survey on process model similarity. In Janis A. Bubenko Jr., John Krogstie, Oscar Pastor, Barbara Pernici, Colette Rolland, and Arne SØlvberg, editors, Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE, pages 421-427. Springer, 2013.

[DVR11] Remco Dijkman, Irene Vanderfeesten, and Hajo A Reijers. The road to a business process architecture: an overview of approaches and their use. The Nederlands: Einhoven University of Technology, 2011.

[EG07] Rik Eshuis and Paul W. P. J. Grefen. Structural matching of bpel processes. In Fifth IEEE European Conference on Web Services

(ECOWS 2007), 26-28 November 2007, Halle (Saale), Germany, pages 171-180. IEEE Computer Society, 2007.

[EKO07] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring similarity between semantic business process models. In John F. Roddick and Annika Hinze, editors, Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 30 - February 2, 2007, Proceedings, volume 67 of CRPIT, pages 71-80. Australian Computer Society, 2007.

[FMP11] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. Process model generation from natural language text. In Advanced Information Systems Engineering, pages 482-496. Springer, 2011.

[Fri09] Fabian Friedrich. Measuring semantic label quality using wordnet. In Proceedings of the 8th Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK), pages 42-57, 2009.

[FT09] Chiara Francescomarino and Paolo Tonella. Supporting ontology-based semantic annotation of business processes with automated suggestions. In Enterprise, Business-Process and Information Systems Modeling, volume 29 of LNBIP, pages 211-223. Springer Berlin Heidelberg, 2009.

[FWAW03] C. Fillies, G. Wood-Albrecht, and F. Weichhardt. Pragmatic applications of the Semantic Web using SemTalk. Computer Networks, 42(5):599-615, 2003.

[GBPG13] Gerd Gröner, Marko Boskovic, Fernando Silva Parreiras, and Dragan Gasevic. Modeling and validation of business process families. Inf. Syst., 38(5):709-726, 2013.

[GKC07] A.K. Ghose, G. Koliadis, and A. Chueng. Process Discovery from Model and Text Artefacts. In 2007 IEEE Congress on Services, pages 167-174. IEEE Computer Society, 2007.

[GL11] V. Gruhn and R. Laue. Detecting Common Errors in Event-Driven Process Chains by Label Analysis. Enterprise Modelling and Information Systems Architectures, 6(1):3-15, 2011.

[HKO07] Thomas Hornung, Agnes Koschmider, and Andreas Oberweis. Rule-based autocompletion of business process models. In CAiSE Forum, volume 247, 2007.

[HLD+05] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on, pages 535-540. IEEE, 2005.

[HR07] Martin Hepp and Dumitru Roman. An ontology framework for semantic business process management. Wirtschaftinformatik Proceedings 2007, page 27, 2007.

[JWW+10] T. Jin, J. Wang, N. Wu, M. La Rosa, and A. ter Hofstede. Eficient and accurate retrieval of business process models through indexing. In Proceedings of the 18th CoopIS, Part I, volume 6426 of Lecture Notes in Computer Science, pages 402-409. Springer, 2010.

[KB04] M. Klein and A. Bernstein. Towards high-precision service retrieval. IEEE Internet Computing, 8(1):30-36, 2004.

[KB07] Agnes Koschmider and Emmanuel Blanchard. User assistance for business process model decomposition. In Proceedings of the 1st IEEE International Conference on Research Challenges in Information Science, pages 445-454, 2007.

[KFSO14] Agnes Koschmider, Michael Fellmann, Andreas Schoknecht, and Andreas Oberweis. Analysis of process model reuse: Where are we now, where should we go from here? Decision Support Systems, 66:9-19, 2014.

[KLW+] Christopher Klinkmüller, Henrik Leopold, Ingo Weber, Jan Mendling, and André Ludwig. Listen to me: Improving process model matching through user feedback. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, volume 8659 of Lecture Notes in Computer Science, pages 84-100. Springer.

[KRG07] Jochen Malte Ku ster, Ksenia Ryndina, and Harald Gall. Generation of business process models for object life cycle compliance. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings, volume 4714 of Lecture Notes in Computer Science, pages 165-181. Springer, 2007.

[KWW11] Matthias Kunze, Matthias Weidlich, and Mathias Weske. Behavioral similarity - a proper metric. In Proceedings of the 9th

International Conference on Business Process Management, volume 7481 of Lecture Notes in Computer Science, pages 166-181. Springer, 2011.

[LD05] Y. Lin and H. Ding. Ontology-based Semantic Annotation for Semantic Interoperability of Process Models. In CIMCA 2005, pages 162-167, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[Leo13] Henrik Leopold. Natural Language in Business Process Models: Theoretical Foundations, Techniques, and Applications, volume 168 of LNBIP. Springer, 2013.

[LESM+13] Henrik Leopold, Rami-Habib Eid-Sabbagh, Jan Mendling, Leonardo Guerreiro Azevedo, and Fernanda Araujo Baiao. Detection of naming convention violations in process models for different languages. Decision Support Systems, 56:310-325, 2013.

[LM12] Henrik Leopold and Jan Mendling. Automatic derivation of service candidates from business process model repositories. In Business Information Systems, pages 84-95, 2012.

[LMP14] H. Leopold, J. Mendling, and A. Polyvyanyy. Supporting process model validation through natural language generation. Software Engineering, IEEE Transactions on, 40(8):818-840, Aug 2014.

[LMRR14] Henrik Leopold, Jan Mendling, Hajo A. Reijers, and Marcello La Rosa. Simplifying process model abstraction: Techniques for generating model names. Inf. Syst., 39:134-151, 2014.

[LNW+12] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco M. Dijkman, and Heiner Stuckenschmidt. Probabilistic optimization of semantic process model matching. In Proceedings of the 10th international conference on Business process management, pages 319-334, 2012.

[LPM13] Henrik Leopold, Fabian Pittke, and Jan Mendling. Towards measuring process model granularity via natural language analysis. In Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, pages 417-429, 2013.

[LPM14] Henrik Leopold, Fabian Pittke, and Jan Mendling. Towards measuring process model granularity via natural language analysis. In Business Process Management Workshops, pages 417-429. Springer, 2014.

[LSM10] H. Leopold, S. Smirnov, and J. Mendling. Refactoring of Process Model Activity Labels. In NLDB 2010, volume 6177 of LNCS, pages 268-276. Springer, 2010.

[LSM12] Henrik Leopold, Sergey Smirnov, and Jan Mendling. On the refactoring of activity labels in business process models. Information Systems, 37(5):443-459, 2012.

[LSW14] Niels Lohmann, Minseok Song, and Petia Wohed, editors. Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers, volume 171 of Lecture Notes in Business Information Processing. Springer, 2014.

[LVD09] Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - A survey. T. Petri Nets and Other Models of Concurrency, 2:46-63, 2009.

[MB13] Saleem Malik and ImranSarwar Bajwa. Back to origin: Transformation of business process models to business rules. In Marcello La Rosa and Pnina Soffer, editors, Business Process Management Workshops, volume 132 of Lecture Notes in Business Information Processing, pages 611-622. Springer Berlin Heidelberg, 2013.

[MDM13] Monika Malinova, Remco M. Dijkman, and Jan Mendling. Automatic extraction of process categories from process model collections. In Lohmann et al. [LSW14], pages 430-441.

[Men08] J. Mendling. Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness, volume 6 of LNBIP. Springer, 2008.

[Moo09] Daniel L. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Software Eng., 35(6):756-779, 2009.

[MRR10] J. Mendling, H. A. Reijers, and J. Recker. Activity Labeling in Process Modeling: Empirical Insights and Recommendations. Information Systems, 35(4):467-482, 2010.

[MRvdA10] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven Process Modeling Guidelines (7PMG). Information and

Software Technology, 52(2):127-136, 2010.

[PLM13] Fabian Pittke, Henrik Leopold, and Jan Mendling. Spotting terminology deficiencies in process model repositories. In Enterprise, Business-Process and Information Systems Modeling, 2013.

[PLM14] Fabian Pittke, Henrik Leopold, and Jan Mendling. When language meets language: Anti patterns resulting from mixing natural and modeling language. In 5th International Workshop on Process Model Collections: Management and Reuse (PMC-MR 2014). Proceedings, LNBIP. Springer, 2014.

[PLM15] Fabian Pittke, Henrik Leopold, and Jan Mendling. Automatic detection and resolution of lexical ambiguity in process models. IEEE Transactions on Software Engineering, 2015.

[PSFGP10] María Poveda, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Péerez. Common pitfalls in ontology development. In Current Topics in Artificial Intelligence, pages 91-100. Springer, 2010.

[PW11] N. Peters and M. Weidlich. Automatic Generation of Glossaries for Process Modelling Support. Enterprise Modelling and Information Systems Architectures, 6(1):30-46, 2011.

[QAR11] Mu Qiao, Rama Akkiraju, and Aubrey J. Rembert. Towards eficient business process clustering and retrieval: Combining language modeling and structure matching. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings, volume 6896 of Lecture Notes in Computer Science, pages 199-214. Springer, 2011.

[RMD11] Hajo A. Reijers, Jan Mendling, and Remco M. Dijkman. Human and automatic modularizations of process models to enhance their comprehension. Inf. Syst., 36(5):881-897, 2011.

[RMKL12] Stefanie Rinderle-Ma, Sonja Kabicher, and Linh Thao Ly. Activity-oriented clustering techniques in large process and compliance rule repositories. In Business Process Management Workshops, pages 14-25. Springer, 2012.

[Ros06] M. Rosemann. Potential Pitfalls of Process Modeling: Part A. Business Process Management Journal, 12(2):249-254, 2006.

[SDMW10] S. Smirnov, R.M. Dijkman, J. Mendling, and M. Weske. Meronymy-Based Aggregation of Activities in Business Process Models. In ER 2010, volume 6412 of LNCS, pages 1-14. Springer, 2010.

[Sil11] Bruce Silver. BPMN Method and Style, with BPMN Implementer's Guide. Cody-Cassidy Press, 2nd edition, January 2011.

[SP10] A. Sinha and A. Paradkar. Use Cases to Process Specifications in Business Process Modeling Notation. In 2010 IEEE International Conference on Web Services, pages 473-480. IEEE, 2010.

[SRW12] Sergey Smirnov, Hajo A. Reijers, and Mathias Weske. From fine-grained to abstract process models: A semantic approach. Inf. Syst., 37(8):784-797, 2012.

[SRWN12] Sergey Smirnov, Hajo A. Reijers, Mathias Weske, and Thijs Nugteren. Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases, 30(1):63-99, 2012.

[SWM12] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. Business process model abstraction based on synthesis from consistent behavioural profiles. International Journal of Cooperative Information Systems, 21, 2012.

[SWMW12] Sergey Smirnov, Matthias Weidlich, Jan Mendling, and Mathias Weske. Action patterns in business process model repositories. Computers in Industry, 63, 2012.

[TF07] Oliver Thomas and Michael Fellmann. Semantic business process management: Ontology-based process modeling using event-driven process chains. IBIS, 4:29-44, 2007.

[van00] W.M.P. van der Aalst. Business Process Management, volume 1806 of LNCS, chapter Workow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques, pages 161-183. Springer, 2000.

[vdA13] Wil MP van der Aalst. Business process management: A comprehensive survey. ISRN Software Engineering, 2013, 2013.

[vdVGvdR97] Bram van der Vos, Jon Atle Gulla, and Reind van de Riet. Verification of conceptual models based on linguistic knowledge. Data & Knowledge Engineering, 21(2):147-163, 1997.

[WDM10] M. Weidlich, R.M. Dijkman, and J. Mendling. The ICoP Framework: Identification of Correspondences between Process Models. In CAiSE 2010, volume 6051 of LNCS, pages 483-498. Springer, 2010.

[Wes12] Mathias Weske. Business Process Management: Concepts, Languages, Architectures. Springer, 2nd edition, 2012.

[WHM10] I. Weber, J. Ho_mann, and J. Mendling. Beyond Soundness: on the Verification of Semantic Business Process Models. Distributed and Parallel Databases, 27(3):271-343, 2010.

[WMW11] Matthias Weidlich, Jan Mendling, and Mathias Weske. Eficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Software Eng., 37(3):410-429, 2011.

[WRMR11] Barbara Weber, Manfred Reichert, Jan Mendling, and Hajo A. Reijers. Refactoring large process model repositories. Computers in Industry, 62(5):467-486, 2011.

[WRR08] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features - enhancing exibility in process-aware information systems. Data Knowl. Eng., 66(3):438-466, 2008.

[WW90] Y. Wand and R. Weber. Studies in Bunge's Treatise on Basic Philosophy, chapter Mario Bunge's Ontology as a Formal Foundation for Information Systems Concepts, pages 123-149. the Poznan Studies in the Philosophy of the Sciences and the Humanities. Rodopi, 1990.

[WW95] Y. Wand and R. Weber. On the deep structure of information systems. Information Systems Journal, 5:203-223, 1995.

[WW02] Y. Wand and R. Weber. Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda. Information Systems Research, 13(4):363-376, 2002.

[YDG12] Zhiqiang Yan, Remco Dijkman, and Paul Grefen. Fast business process similarity search. Distributed and Parallel Databases, 30:105-144, 2012.

[ZWW+10] Haiping Zha, Jianmin Wang, Lijie Wen, Chaokun Wang, and Jiaguang Sun. A workow net similarity measure based on transition adjacency relations. Computers in Industry, 61(5):463-471, 2010.

CRITERIOS DE CALIDAD COMO MEDIO CIENTÍFICO DE COMUNICACIÓN

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» cuenta con un Comité Científico Internacional de 10 investigadores internacionales y un Consejo Científico de Revisores Internacionales de más de 50 miembros. El Comité Científico asesora y evalúa la publicación, avalándola científicamente y proyectándola internacionalmente. El Comité de Revisores somete a evaluación ciega los manuscritos estimados en la publicación.

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» ofrece información detallada a sus autores y colaboradores sobre el proceso de revisión de manuscritos y marca criterios, procedimientos, plan de revisión y tiempos máximos de forma estricta:

1) Fase previa de estimación/desestimación de manuscritos (máximo 30 días);
2) Fase de evaluación de manuscritos con rechazo/aceptación de los mismos (máximo 150 días);
3) Edición de los textos en digital.

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» acepta para su evaluación manuscritos en español e inglés, editándose todos los trabajos a texto completo en bilingüe.

CRITERIOS DE CALIDAD DEL PROCESO EDITORIAL

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» edita sus números con una rigurosa periodicidad semestral (en los meses de junio y diciembre). Mantiene, a su vez, una estricta homogeneidad en su línea editorial y en la temática de la publicación.

Todos los trabajos editados en «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» se someten a evaluaciones previas por expertos del Comité Científico así como investigadores independientes de reconocido prestigio en el área.

Las colaboraciones revisadas en «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» están sometidas, como mínimo requisito, al sistema de evaluación ciega por pares, que garantiza el anonimato en la revisión de los manuscritos. En caso de discrepancia entre los evaluadores, se acude a nuevas revisiones que determinen la viabilidad de la posible edición de las colaboraciones.

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» notifica de forma motivada la decisión editorial que incluye las razones para la estimación previa, revisión posterior, con aceptación o rechazo de los manuscritos, con resúmenes de los dictámenes emitidos por los expertos externos.

«International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» cuenta en su organigrama con un Comité Científico, Consejo de Revisores y Consejo Técnico, además del Editor, Editores Adjuntos, Centro de Diseño y Gestión Comercial.

CRITERIOS DE LA CALIDAD CIENTÍFICA DEL CONTENIDO

Los artículos que se editan en «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» están orientados básicamente al progreso de la ciencia en relación con el uso de la Ingeniería de Software en las grandes empresas, y las Tecnologías de la Información y la comunicación (TIC).

Los trabajos publicados en «International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC)» acogen aportaciones variadas de expertos e investigadores de todo el mundo, velándose rigurosamente en evitar la endogamia editorial, especialmente de aquéllos que son miembros de la organización y de sus Consejos.

© ISSN: 2387-0184

ua
JOURNALS

United Academic Journals
www.uajournals.com

IJISEBC

International Journal of Information Systems and Software Engineering for Big Companies

www.ijisebc.com

Empresas colaboradoras:

Steelmood
www.steelmood.com

International Journal of Information Systems and Software Engineering for Big Companies

IJISEBC

EDITA

Grupo de Investigación de las Tecnologías de la Información y las Comunicaciones en la Empresa

GITICE

www.gitice.com