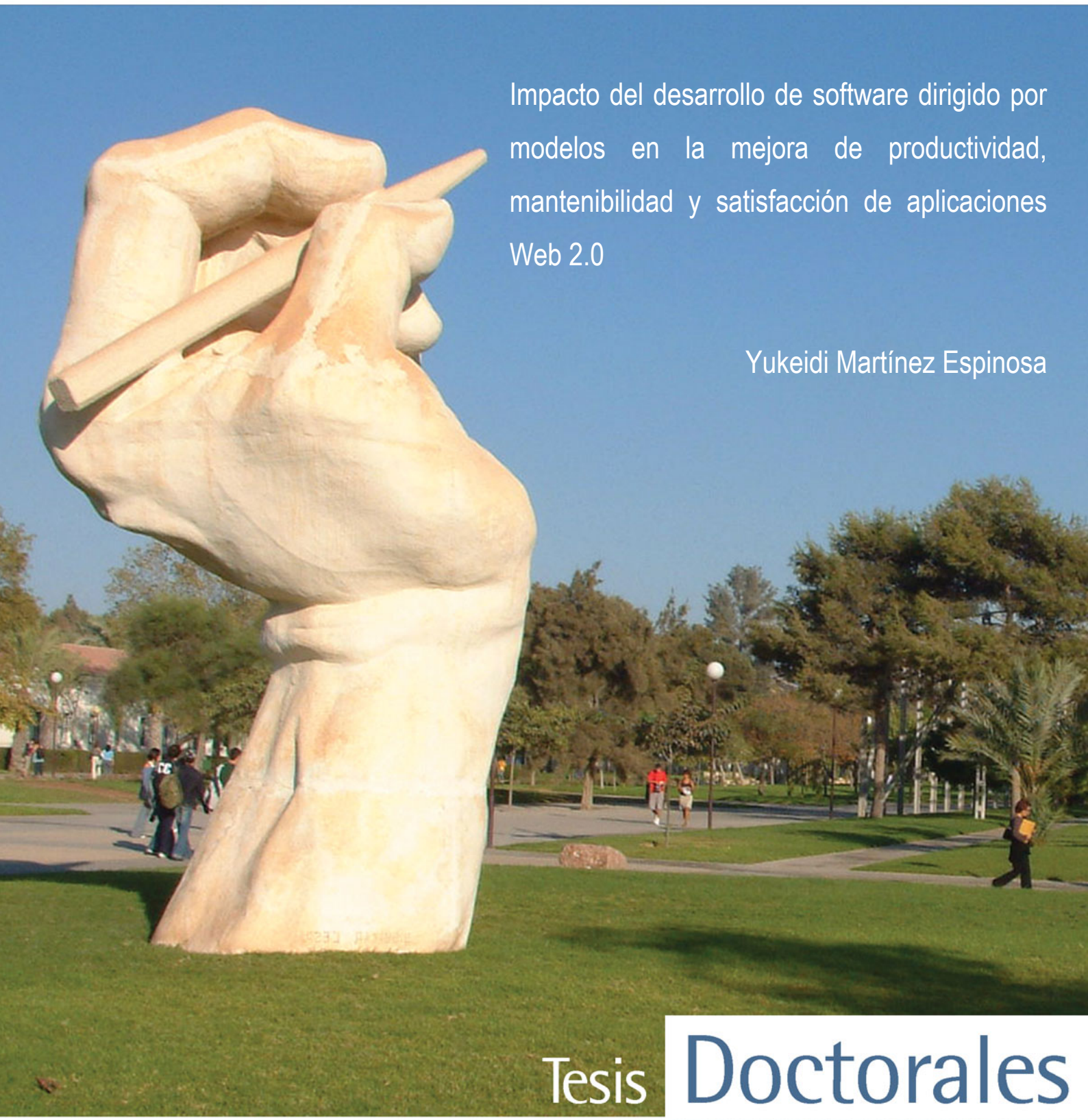
Universitat d'Alacant
Universidad de Alicante



Impacto del desarrollo de software dirigido por modelos en la mejora de productividad, mantenibilidad y satisfacción de aplicaciones Web 2.0

Yukeidi Martínez Espinosa

Tesis Doctorales

www.eltallerdigital.com

UNIVERSIDAD de ALICANTE

Universidad d'Alacant

Universidad de Alicante

Departamento de Lenguajes y Sistemas Informáticos

Memoria de Tesis Doctoral

Impacto del desarrollo de software dirigido por modelos en la mejora de productividad, mantenibilidad y satisfacción de aplicaciones Web 2.0

Autor: Yulkeidi Martínez Espinosa

Directores: Dr.C. Cristina Cachero Castro
Dr.C. Santiago Meliá Beigbeder

Universidad de Alicante

2012

Universidad d'Alacant

Universidad de Alicante

Departamento de Lenguajes y Sistemas Informáticos

Memoria de Tesis Doctoral

Impacto del desarrollo de software dirigido por modelos en la mejora de productividad, mantenibilidad y satisfacción de aplicaciones Web 2.0

Autor: Yulkeidi Martínez Espinosa
email: yulkeidi@gmail.com

Directores: Dr.C. Cristina Cachero Castro Prof. Titular de Universidad
Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Alicante, UA
email: ccachero@dlsi.ua.es

Dr.C. Santiago Meliá Beigbeder Prof. Contratado Doctor
Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Alicante, UA
email: santi@ua.es

Universidad de Alicante

2012

PENSAMIENTO

"Las evaluaciones deberían basarse en evidencias directas del producto, y no en evidencias circunstanciales del proceso."

*Maibaum, T. y Wassyng, A. (*2008*).*

DEDICATORIA

A mi madre, **Ana**,

que siempre quiso un médico en la familia:

¡Mami, igual seré doctora!.

AGRADECIMIENTOS

Quisiera expresar mi más sincero agradecimiento a todos los que, a nivel profesional y, sobre todo, a nivel personal, han hecho posible este trabajo.

A mis directores de tesis:

- Dra. Cristina Cachero Castro por la tutela a lo largo de estos 5 años a pesar de la distancia, que con sus buenos consejos, siempre buen humor y enfado disimulado; me tenían siempre a la expectativa de un "*bien hecho yul*", "*felicitaciones*" o "*lo hicimos*", que la verdad eran muy pocos.... je je je.

- Dr. Santiago Meliá, pues cuando todo se creía perdido, sacaba un "*As*" de debajo de la manga.

A la Dra. Maristella Matera, Dra. Silvia Abrahao y al Dr. Sergio Luján por cooperar en el desarrollo del primer experimento, y permitirme compartir sus experiencias y bastos conocimientos.

A mis amig@s y mis compañer@s de trabajo: Yeleny, Yanitza, Juliett, Yaimara, Yanirys, Eglys, Dignora, Liuva, Huber, Ichi y tantos otros; por sus buenas vibras y soportar mis cambios de humor. Y en especial a la "*jefa*" Dra. Raquel Diéguez, por su maternal ternura y al Dr. Raúl Fernández Aedo por cubrirme la espalda.

A la Dra. Nancy González, por sus sabios consejos y ayuda desinteresada.

A mis profesores del doctorado, por iniciarme en el camino de la investigación.

A José Javier Martinez y Juan Antonio Osuna quienes contribuyeron al desarrollo de la herramienta OOH4RIA IDE y permitir de esta forma la realización de los experimentos; así como a los alumnos que se tomaron el tiempo de participar en los estudios empíricos.

A mi familia toda, los más cercanos y lejanos, que me tendieron la mano en momentos difíciles.

Y sobre todo a los tres hombres de mi vida:

- A mi *bro*: Yunieski, por su amor y apoyo incondicional, por llamarme la atención cuando estaba distraída, y sobretodo hacerme notar que está orgulloso de su hermana.

- A mi padre: Armindo, por motivarme siempre a seguir adelante, por su sonrisa franca y la bondad de su corazón.

- A mi novio: Alex, por su infinita paciencia y amor, por ser el hombro amigo que me esperaba en casa y por su frase célebre "*cuando te hagas doctora...*".

Sin ustedes, nada de esto habría sido posible.

Gracias a todos,

yul

Alicante (España), 2012

TABLA DE CONTENIDO

Página

ÍNDICE DE TABLAS

Tabla Página

ÍNDICE DE FIGURAS

Figura Página

INTRODUCCIÓN

La comunidad de Ingeniería Web aboga por el uso de modelos con el fin de mejorar los procesos de desarrollo de *software* para aplicaciones Web. Una forma de clasificar las prácticas de modelado en la industria es de acuerdo al grado en que se utilizan los modelos para apoyar el proceso de desarrollo. Fowler [19] describe tres modos diferentes en que los lenguajes de modelado -y sobre todo UML (*Unified Modeling Language*)- se pueden utilizar: boceto, plano y lenguaje de programación.

- Los bocetos (*sketches*) son diagramas informales utilizados para comunicar ideas. Por lo general, se centran en un aspecto particular del sistema y no están destinados a mostrar todos los detalles. Es el uso más común del UML, y recomendado en prácticas de desarrollo ágil, centradas en el código (*code-centric*) en *frameworks* como Scrum, donde las herramientas son poco usadas.
- Los planos (*blueprints*) son diagramas que muestran la mayoría de los detalles de un sistema con el fin de fomentar su conocimiento o para proporcionar puntos de vista del código de forma gráfica. Los planos son ampliamente utilizados en prácticas de desarrollo basado en modelos (MBD, *Model-Based Development*), como los promovidos por *frameworks* como RUP (*Rational Unified Process*) [8].
- Los modelos también pueden ser utilizados para caracterizar completamente la aplicación (*programming language*). Si tal es el caso, los diagramas reemplazan el código, y estos son compilados directamente en archivos binarios ejecutables. Este uso del modelado tiene como punto de partida la ingeniería Web que se crea bajo el enfoque de desarrollo dirigido por modelos (MDD, *Model Driven Development*).

El MDD, también reconocido por las siglas MDE (del inglés, *Model Driven Engineering*), es una aproximación al desarrollo de *software* basado en: (a) la creación de modelos del sistema a distintos niveles de abstracción y, (b) su uso como base de un proceso de generación automática de código [75]. Entre las reivindicaciones de este paradigma de desarrollo se encuentran [41, 76]:

1. Mayor simplicidad del proceso. El desarrollador se puede aislar de la complejidad tecnológica y centrarse en la estructura y comportamiento deseado de la aplicación.
2. Mejora de la productividad del proceso de desarrollo. El uso de modelos permite especificar el sistema a distintos niveles de abstracción y favorece de este modo el reuso. La definición de transformaciones modelo a modelo y modelo a código automatiza gran parte del proceso de codificación.
3. Mejora de la calidad externa de la aplicación resultante (Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad) [32].
4. Mejora de la satisfacción de los desarrolladores con el proceso de desarrollo.

La ingeniería Web es un dominio específico donde el desarrollo de *software* dirigido por modelos puede ser aplicado con éxito [60]. Los enfoques de Ingeniería Web Dirigidos por Modelos (MDWE, *Model-Driven Web Engineering*), entre los que se destacan RUX [40], WebML [10], OOH4RIA [48], OOHDM [82], UWE [70] y OOWS [84], proporcionan un conjunto de métodos y herramientas adecuadas al diseño y desarrollo de la mayoría de los tipos de aplicaciones Web.

Sin embargo, y a pesar de las numerosas llamadas de atención en la comunidad de Ingeniería del *Software* acerca de la necesidad de acompañar este tipo de afirmaciones con evidencias empíricas [22, 68], la gran mayoría de las contribuciones en el campo del MDE siguen siendo la definición de nuevas metodologías, técnicas y herramientas que, aunque viables, no llegan a demostrar de manera fiable su utilidad y ventajas respecto a sus predecesoras.

Objetivos de Investigación e Hipótesis Inicial

Con el fin de contribuir a que la comunidad de Ingeniería Web proporcione estas evidencias empíricas, el área de experimentación en Ingeniería del *Software* ha desarrollado guías exhaustivas que ayudan a los investigadores en el proceso de obtención de datos fiables acerca de las ventajas o desventajas de los distintos métodos, técnicas o herramientas empleadas en la construcción de sistemas *software* [25, 38]. Sin embargo, en ausencia de estos datos empíricos, se sigue corriendo el riesgo de sostener conclusiones erróneas [34], perjudicando de esta manera tanto la toma de decisiones en el ámbito empresarial [69] como la propia imagen de la disciplina, tal y como ya ha sucedido en el pasado [85].

Aún más importante, es necesario proporcionar un acceso rápido, claro y conciso a las evidencias empíricas de las que se dispone, de manera que ese conocimiento llegue a los encargados de decidir acerca de su adopción en la práctica. En el caso de MDE, esta falta de organización de la evidencia empírica, así como, cuando existe, su falta de relación con metodologías bien definidas, puede estar perjudicando su adopción por parte de las empresas. La decisión de adoptar una nueva aproximación o utilizar nuevas herramientas en el desarrollo de *software* debe venir avalada por un proceso fiable y repetible, de manera que se maximicen las probabilidades de éxito de su implantación en la industria.

Por tanto, el objetivo de esta investigación puede resumirse como *aumentar el acervo empírico existente respecto al impacto de las prácticas MDE sobre la mantenibilidad, productividad, satisfacción y, en último término, intención de uso de los desarrolladores y mantenedores de aplicaciones Web*. Este objetivo ha sido desglosado como sigue:

- Identificar y clasificar la evidencia empírica existente con respecto a las mejoras en productividad y mantenibilidad que reporta el uso del paradigma MDE. Para ello se realiza un mapeo sistemático, un tipo de estudio secundario que está diseñado específicamente para abordar este tipo de objetivos.
- Comparar la mantenibilidad de aplicaciones cuando se realizan tareas de mantenimiento (a) directamente sobre código (*code-centric*) o (b) directamente sobre modelos (MDE).

- Comparar la satisfacción de los mantenedores de *software* cuando realizan las tareas de mantenimiento (a) directamente sobre código (*code-centric*) o (b) directamente sobre modelos (MDE).
- Comparar la productividad de los desarrolladores de *software* cuando realizan las tareas de desarrollo (a) directamente sobre código (*code-centric*), (b) con la ayuda de modelos (MBD) o (c) directamente sobre modelos (MDE).
- Comparar la satisfacción de los desarrolladores de *software* cuando realizan las tareas de desarrollo (a) directamente sobre código (*code-centric*), (b) con la ayuda de modelos (MBD) o (c) directamente sobre modelos (MDE).
- Comparar la intención de adopción de los distintos paradigmas de desarrollo (*code-centric*, MBD y MDE).

Para ello, se han diseñado un conjunto de quasi-experimentos basados en el *framework* para la experimentación en la Ingeniería del *Software* sugerido en [89]. Las hipótesis de partida de la presente investigación han sido las siguientes:

Hipótesis 1: El uso de MDE para el mantenimiento de aplicaciones mejora la mantenibilidad de la aplicación resultante con respecto al mantenimiento directamente sobre código.

Hipótesis 2: El uso de MDE para el desarrollo de aplicaciones mejora la productividad y satisfacción del desarrollador con respecto al desarrollo basado en modelo y centrado en el código.

Hipótesis 3: El uso de MDE para el mantenimiento de aplicaciones incrementa la satisfacción del desarrollador e incide positivamente en la intención de adopción de este método para futuros desarrollos, con respecto al mantenimiento de aplicaciones basadas en modelos y centradas en el código.

Publicaciones Pertenecientes a esta Tesis Doctoral

Los artículos que por su contribución científica forman parte de esta tesis doctoral son:

Capítulo 2:

Martínez, Y., Cachero, C. y Meliá, S. - Evidencia empírica sobre mejoras en productividad y calidad en enfoques MDD: un mapeo sistemático. Revista Española de Innovación, Calidad e Ingeniería del Software. Volumen 7, No. 2, octubre, 2011.

Este trabajo, aplicando el proceso de mapeo sistemático, clasifica la evidencia empírica existente respecto a la mejora en productividad y calidad -con especial énfasis en mantenibilidad- de las aplicaciones, e identifica un conjunto de asunciones en este campo de investigación que carecen a día de hoy de evidencia empírica que las sustente o, si la hay, es conflictiva. Este trabajo justifica por tanto la novedad y relevancia de las contribuciones de esta tesis, y es la base de las hipótesis formuladas en ella.

Capítulo 3:

Martínez, Y. and Cachero, C. and Matera, M. and Abrahao, S. and Luján, S. - Impact of MDE approaches on the maintainability of Web applications: an experimental evaluation. Proceedings of Conceptual Modeling ER 2011*: 30th International Conference on Conceptual Modeling, Brussels, Belgium, October* 31*-November* 3*,* 2011*, pp* 233-246.

Dadas las discrepancias existentes en la literatura sobre cuán significativo es el impacto de enfoques MDE sobre el tiempo necesario para realizar un cambio en el *software*, que fueron identificadas en el capítulo anterior, en este capítulo se presenta un estudio empírico que compara una metodología MDE (WebML) y una metodología centrada en el código (PHP) con respecto al rendimiento y satisfacción de desarrolladores noveles mientras llevan a cabo las tareas de analizabilidad y cambiabilidad tanto correctivas como perfectivas en la capa de presentación de una aplicación Web. Los resultados muestran que los sujetos que realizan las tareas de mantenimiento con WebML obtienen un mayor rendimiento, aunque

muestran una ligera preferencia por realizar las tareas de mantenibilidad directamente en el código fuente.

Capítulo 4:

Martínez, Y., Cachero, C. and Meliá, S. - Evaluating the impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams. Proceeding of 12th International Conference on Web Engineering ICWE 2012: Berlin, Germany, july 23-27, 2012, pp 223-237.

Entre las evidencias que se presentan en el capítulo 2 se reportan tanto ganancias como pérdidas en la productividad de los desarrolladores cuando utilizan el paradigma MDE. Con el fin de aumentar el acervo empírico en la comunidad de la Ingeniería Web, este trabajo presenta un estudio empírico que compara la productividad y satisfacción de desarrolladores noveles durante el desarrollo de la lógica de negocio de una aplicación Web 2.0, usando tres aproximaciones de desarrollo: *code-centric* (lenguaje *C#* en *Visual Studio* .NET 2010), MBD (empleando UML en RSM, para comprender los objetivos del sistema, pero sin generación de código) y MDE (OOH4RIA). Los resultados muestran un significativo incremento de la productividad y la satisfacción de los sujetos cuando usan OOH4RIA. También se demuestra que las actividades de modelado que no están acompañadas de un fuerte ambiente de generación de código (UML con RSM), la satisfacción disminuye con respecto a las prácticas *code-centric* con .NET.

Capítulo 5:

Martínez, Y., Cachero, C. and Meliá, S. - MDD vs.Traditional Software Development: a practitioner's subjective perspective. Information and Software Technology. Published by Elsevier B.V. http://dx.doi.org/10.1016/j.infsof.2012.07.004.

Más allá del impacto del método de desarrollo utilizado sobre la efectividad y la eficiencia de las tareas que se llevan a cabo, cualquier método que aspire a ser adoptado por la industria tiene que garantizar altos niveles de satisfacción de los desarrolladores. Un aspecto importante a tener en cuenta acerca de cambios en el comportamiento humano es

que son las impresiones subjetivas (satisfacción), más que los datos objetivos (productividad), los que inciden en la intención de adopción de un método de desarrollo de *software* y, en última instancia, en su nivel de adopción final [58].

En este capítulo se vuelven a comparar mediante un quasi-experimento tres métodos, cada uno siguiendo un enfoque de desarrollo diferente (*code-centric*, MBD y MDE), respecto a su potencial de adopción por desarrolladores noveles mientras desarrollan la lógica de negocio de una aplicación Web 2.0. Como base a la predicción de la intención de adopción de las tecnologías por los desarrolladores se utiliza una adaptación del Modelo de Adopción de Tecnologías (TAM) [57]. Los resultados muestran que OOH4RIA (como enfoque MDE) es percibido como más útil, aunque también es considerado el menos compatible con las experiencias previas de los desarrolladores. Además se evidencia que los desarrolladores de *software* noveles se sienten cómodos con el uso de modelos, y que estos son preferidos siempre que estén acompañados por un ambiente de desarrollo y generación automática de código MDE.

El último trabajo que forma parte de esta tesis doctoral y que actualmente se encuentra bajo proceso de revisión en revista de alto impacto se presenta en el Apéndice A.

Apéndice A

Martínez, Y., Cachero, C. and Meliá, S. - Model Driven maintainability improvement of Web Applications.

La generalizabilidad de los resultados obtenidos mediante experimentos realizados en ambientes controlados, p.e. su validez externa, se encuentra limitada por el propio entorno experimental. Es por ello que es fundamental realizar réplicas donde se cambien detalles de dicho ambiente con el fin de comprobar si las mismas hipótesis pueden seguir siendo refutadas.

En este trabajo se realiza un nuevo experimento de mantenibilidad, comparando el rendimiento y la satisfacción de desarrolladores noveles mientras llevan a cabo tareas de

mantenibilidad correctivas y perfectivas en la lógica de negocio de una aplicación Web 2.0, cuando usan diferentes enfoques de desarrollo: *code-centric* (lenguaje *C#* en *Visual Studio* .NET 2010) y MDE (usando OOH4RIA). Los resultados muestran que la metodología OOH4RIA mejora el rendimiento de los desarrolladores y la satisfacción, aunque esta última no significativamente.

Estructura de la tesis

La presente tesis se estructura en tres partes, distribuidas de la siguiente manera: en la primera parte (compuesta por el Capítulo 1 y resumen global de la tesis por compendio de artículos), se introduce el campo de investigación MDE y sus principales reivindicaciones en la industria de desarrollo de *software*, reseñando sus conceptos fundamentales. Además, se describen los estudios empíricos realizados y los resultados obtenidos, así como las amenazas a la validez que limitan su generalización. A continuación aparecen las conclusiones, trabajos futuros y referencias bibliográficas.

Los artículos y sus contribuciones son presentados por capítulos en la Parte II de la tesis. Por último, en la Parte III se relacionan los apéndices, que incluye un trabajo que se encuentra actualmente en proceso de revisión.

Parte I

SÍNTESIS EN CASTELLANO

.

CAPÍTULO 1
DESARROLLO Y MANTENIMIENTO DE APLICACIONES WEB UTILIZANDO ENFOQUE MDE

1.1. Introducción

La comunidad de Ingeniería de la Web ha dedicado muchos esfuerzos a la definición de enfoques MDE [83], fuertemente caracterizados por la adopción de modelos conceptuales, con el fin de mejorar los procesos de desarrollo de las aplicaciones Web. Estos enfoques permiten capturar las características más sobresalientes de las aplicaciones Web sin necesidad de bajar a detalles de implementación, y por tanto permiten acelerar el proceso de desarrollo de este tipo de aplicaciones. MDE también hace énfasis en la importancia de las transformaciones de modelos (modelo a modelo y modelo a código), que permiten la generación automática del código de la aplicación a partir de los modelos conceptuales de alto nivel. Entre los argumentos más comúnmente esgrimidos en favor de los enfoques MDE se encuentran: facilidad de mantenimiento, incremento de la productividad y una mayor satisfacción de los desarrolladores.

Con el objetivo de proveer a la comunidad de desarrolladores de Ingeniería Web y a los desarrolladores de *software* en general, con nuevas evidencias empíricas que respalden/refuten los supuestos de este paradigma, en esta tesis doctoral se reporta una familia de experimentos centrada en la comparación de la mantenibilidad, productividad, satisfacción e intención de adopción del MDE con respecto a otros paradigmas existentes. A continuación se presenta un resumen global de esta tesis: el estado actual de la investigación en

este campo (obtenido a partir de un mapeo sistemático de la literatura), los experimentos realizados, los resultados obtenidos y las principales conclusiones que ha sido posible extraer.

1.2. Estado actual

Para consolidar el avance del desarrollo MDE y poner a disposición de la comunidad de Ingeniería del *Software* (IS) las evidencias empíricas existentes, acerca de cómo MDE contribuye a mejorar la calidad de la aplicación resultante -con especial énfasis en la mantenibilidad- y la productividad del proceso de desarrollo, se ha realizado un mapeo sistemático [37, 67] para organizar los datos publicados hasta el momento a través de un mapa conceptual.

La búsqueda se ha centrado en el período 2001-2010, pues la OMG (*Object Management Group*) propuso la adopción de MDA (*Model Driven Architecture*) como estándar para las actividades involucradas en el MDE en el año 2001, aunque la guía oficial no fue publicada hasta junio del 2003 [65]. Las principales fuentes de datos utilizadas han sido cuatro: *Google Scholar* como motor de búsqueda líder en el seno de la comunidad científica de investigadores académicos [63] y los tres foros más significativos en la IS: ACM, IEEE y *Springer*.

En este estudio, de las 599 publicaciones encontradas que se relacionan con el impacto del paradigma MDE sobre calidad del producto y productividad del proceso, solo el $8,35\,\%$ (50 publicaciones) han resultado relevantes para la investigación. Estos trabajos se han clasificado de acuerdo a: las medidas evaluadas (calidad del proceso y del producto) [32], el tipo de estudio empírico realizado (caso de estudio, experimento, encuesta o meta-análisis) [35, 51] y el enfoque de investigación utilizado (validación en entornos controlados o evaluación en entornos reales) [88]; además, se ha recogido el año y tipo de publicación mediante el que fue diseminado.

El resultado sintetizado de esta actividad se puede observar de manera gráfica en el diagrama de burbuja de la Figura 1–1, que a través de un diagrama de dispersión XY

[67] visualiza la frecuencia de estudios empíricos, según el tipo de estudio, ordenados por años y por medidas de calidad. En este diagrama la burbuja ilustra el número de artículos que están en el par de categorías que corresponden a la coordenada. Cuando un trabajo ha afectado más de una categoría (p.e. un meta-análisis de más de una característica de calidad, o presentación de más de un tipo de estudio empírico), el trabajo ha sido contabilizado en todas las características.

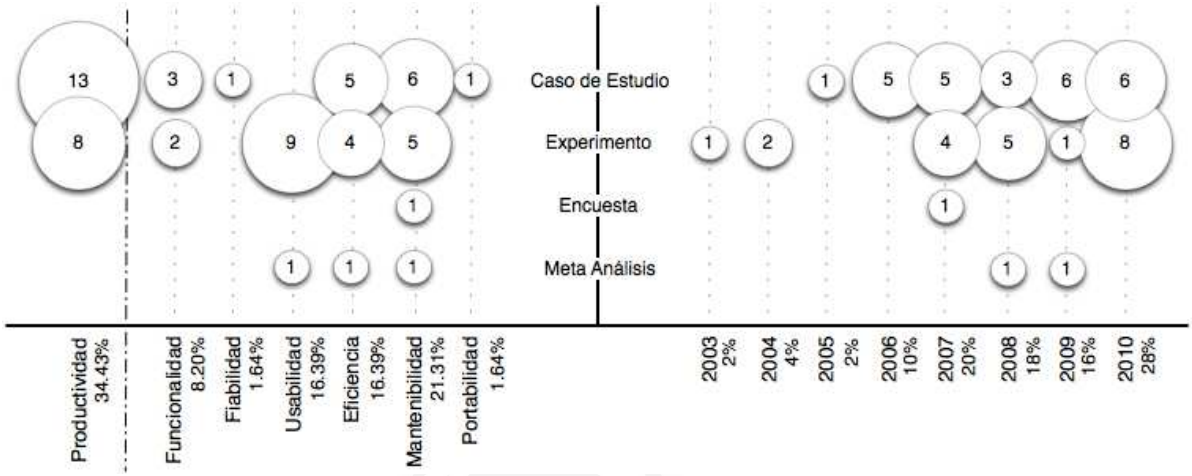


Figura 1–1: Diagrama de burbuja. Visualización del mapeo sistemático.

Como se puede apreciar (ver Figura 1–1) el porcentaje de estudios que realmente proporcionan evidencia empírica acerca de mejoras de calidad y productividad en MDE es bastante bajo. De manera general, los resultados más relevantes reportados por los estudios empíricos existentes se pueden resumir como sigue:

- Los estudios que validan el uso de MDE en entornos académicos reportan una productividad de 2 a 9 veces superior a la obtenida con otros paradigmas de desarrollo [16]. La productividad puede llegar a ser hasta 20 veces superior según se va aumentando el tamaño del proyecto de desarrollo. Estos resultados contrastan con los reportados por experimentos en entornos industriales [56], donde los resultados son mucho más heterogéneos, y van desde los que directamente reportan una pérdida de productividad de un 10 % [53], a los que coinciden con los estudios académicos y reportan ganancias que oscilan entre un 20 % y un 35 % de productividad [39, 42, 53, 80].

- Con respecto a la mantenibilidad, los artículos incluidos en este estudio reportan que el tiempo necesario para evaluar el impacto de un cambio es substancialmente más corto si se usa una visualización gráfica (diagramas) en comparación con la utilización de solo código [50]. Además, usando aproximaciones MDE completas para el proceso de mantenimiento se mejora aún más la mantenibilidad [24]. Estos datos contrastan en cierta medida con los reportados en [18], donde se expone que el uso de UML como lenguaje en el mantenimiento no tiene un impacto significativo en el tiempo necesario para realizar un cambio, pues también se debe contar el tiempo para poner al día la documentación de UML. Sin embargo, por lo que se refiere a la exactitud funcional de los cambios (introducción de nuevos errores en el *software* durante el mantenimiento), UML ha demostrado un impacto positivo a la hora de mejorar la calidad del código, incluso si los desarrolladores no están muy familiarizados con su uso.
- En cuanto al impacto de MDE sobre la eficiencia, funcionalidad, fiabilidad, usabilidad y portabilidad, la literatura provee resultados en su mayor parte anecdóticos, poco contrastados, lo que hace muy difícil su generalización. Los estudios se centran principalmente en presentar experiencias donde prácticas específicas (una determinada herramienta, aproximación, técnica, modelo, etc.) han mostrado sus bondades a la hora de mejorar la calidad del producto final.

Además, este mapeo sistemático ha detectado un interés creciente en estudiar el impacto de los factores sociales, técnicos y orgánicos en la productividad y satisfacción de los desarrolladores y, en último término, en el éxito o fracaso de la implantación de una aproximación MDE en la industria. Dentro de este campo, estudios recientes [26, 27] han encontrado una correlación positiva entre un entorno con mejores conocimientos organizativos y mejor comunicación en el equipo de desarrollo y las variables de productividad y mantenibilidad del proceso MDE, con un incremento de $66,7\,\%$ y $73,7\,\%$ respectivamente. Otras conclusiones de estos estudios remarcan la importancia del entrenamiento y la educación para alcanzar las promesas de este paradigma.

Por último, en relación al impacto de MDE sobre la satisfacción, en general se asume que usar métodos y herramientas MDE mejora la experiencia global del desarrollador, y esta a su vez mejora la intención del desarrollador de asumir el método, particularmente cuando los sistemas a desarrollar son grandes y complicados [5]. Sin embargo, la literatura consultada revela una falta de evidencias empíricas que corroboren que la satisfacción mejora realmente.

1.3. Modelo de calidad de la ISO

La evaluación de los productos de *software* es una de las actividades que se incluyen dentro el desarrollo y ejecución de cualquier ciclo de vida. Esta evaluación puede realizarse a través de un conjunto de métricas bien definidas que aseguran la calidad del producto de *software*. Según Pressman [71], la calidad del *software* es "la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo *software* desarrollado profesionalmente".

La investigación de las métricas (medidas) de la calidad del *software* se va adaptando a las necesidades y nuevas propuestas de desarrollo, nuevos lenguajes y nuevos paradigmas de programación [4]. Tal cometido exige de la ISO (*International Organization for Standardization*) y la IEC (*International Electrotechnical Commission*) una constante revaluación de las normas y estándares para definir las características de calidad necesarias para garantizar los objetivos de los sistemas y de los desarrolladores. De ahí la continua evolución de los estándares internacionales asociados a la calidad del producto de *software*: desde las series ISO/IEC 9126 [32], ISO/IEC 14598 [30] y la familia ISO/IEC SQuaRE 25000 [29] que incluye el modelo de calidad actual ISO/IEC 25010 [31].

El autor asume como modelo de calidad el estándar ISO/IEC 9126 [32] (ver Figura 1–2) para la caracterización de los principales conceptos, pues era la norma vigente cuando se comenzaron a desarrollar los experimentos que forman parte de esta tesis doctoral. Esta

norma de calidad ha sido ya cancelada y reemplazada por la ISO/IEC 25010 [31], que fue publicada en el mes marzo del 2011 (01/03/2011).

Según la ISO/IEC 9126, la calidad del producto *software* se puede evaluar desde tres perspectivas: calidad interna, externa y en uso.

- Calidad interna: total de características de un producto de *software* que pueden ser medidas por medio de calidad interna de los requisitos.
- Calidad externa: total de características de un producto de *software* que pueden ser apreciadas cuando el *software* se ejecuta. Por tanto, puede ser medida y evaluada por medio de pruebas en ambientes de simulación y datos. La calidad externa se ve influida por la calidad interna del producto.
- Calidad en uso: punto de vista del usuario con respecto a la calidad de un producto, cuando se utiliza en un ambiente y contexto específico. Mide si los usuarios pueden lograr sus objetivos en lugar de las características propias del *software*. La calidad en uso se ve influida por la calidad interna y externa de un producto.

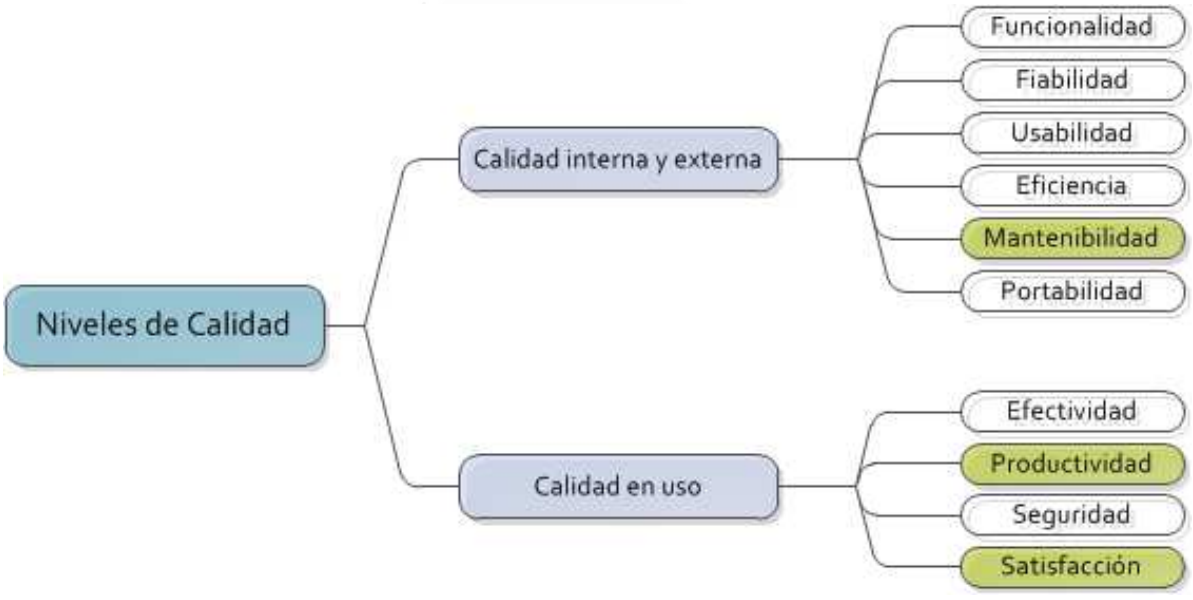


Figura 1–2: Niveles de calidad establecidos como criterios de evaluación del producto [32].

En la Figura 1–2 se representan sombreadas las características de calidad ISO que son analizadas en la presente investigación según las reivindicaciones que el paradigma

MDE provee (Sección 1.2) y las principales líneas de investigación detectadas en el mapeo sistemático realizado. Dichas características se definen como sigue:

- Mantenibilidad: "facilidad con la que un sistema o componente de *software* puede ser modificado para corregir fallos, mejorar su funcionamiento u otros atributos o adaptarse a cambios en el entorno" [28].
- Productividad: "cantidad del producto creado -en relación con artefactos producidos o servicios dados- por unidades de entrada usadas" [12].
- Satisfacción: "capacidad del producto *software* de satisfacer a los usuarios en un contexto especificado de uso" [32].
- Intención de uso: "el grado en que una persona tiene la intención de utilizar un sistema en particular" [58]; redefiniendo este concepto la Intención de adopción: "el grado en que una persona tiene la intención de adoptar[1] un sistema en particular, es decir el método sea utilizado con asiduidad (adoptado) en la práctica" [13].

1.3.1. Mantenibilidad

Actualmente a nivel mundial, el mantenimiento como estructura de apoyo ocupa un lugar importante dentro de las organizaciones, y es visto como pieza fundamental, dados los cambios tecnológicos y la vertiginosa competitividad entre las empresas. Además, su estudio es importante porque: a) absorbe entre el 45 % y el 60 % de los costes totales del proyecto [7, 74], y b) la incapacidad de realizar los cambios rápida y fiablemente significa grandes pérdidas de oportunidades de negocio a las empresas. Esta situación ha llevado a muchos investigadores a interesarse por la mantenibilidad desde diferentes perspectivas.

El mantenimiento de *software* requiere hacer cambios en una estructura o sistema existente. Una generalización de los conceptos o definiciones plantea que "la mantenibilidad es una característica de calidad del *software* relacionada con la facilidad de mantenimiento. A mayor mantenibilidad menores costes de mantenimiento" [77]. Respecto a los cambios

[1] Recibir haciéndolos propios, pareceres, métodos, doctrinas, ideologías, modas, etc., que han sido creadas por otras personas o comunidades". [72]

que pueden tener lugar durante el ciclo de vida de un sistema, algunos autores [11] distinguen entre cuatro diferentes tipos de modificaciones (o tareas de mantenibilidad):

- *Mantenibilidad preventiva*: capacidad del *software* para apoyar la reingeniería de procesos internos, sin efectos adversos.
- *Mantenibilidad correctiva*: capacidad de detectar errores, diagnosticar los problemas y solucionarlos.
- *Mantenibilidad adaptativa*: capacidad de modificar el *software* con el fin de enfrentar los efectos de los cambios de ambiente. Se incluyen los cambios para implementar nuevos requisitos de interfaz, de sistema o de *hardware*.
- *Mantenibilidad perfectiva*: capacidad de extender el *software* de acuerdo a las nuevos requisitos o mejoras.

Algunos autores utilizan una clasificación más sencilla, y se limitan a distinguir entre mantenibilidad correctiva (correcciones) y mantenibilidad perfectiva (que abarca mejoras, adaptaciones y prevenciones). Las mejoras en forma de cambios correctivos y perfectivos son las más comunes [55] y caras -pueden suponer gran parte de los costes de mantenimiento y por ende, del presupuesto de desarrollo-.

Cualquiera que sea el objetivo final, todas estas formas diferentes de la mantenibilidad pueden ser evaluadas a través de medidas asociadas a las siguientes sub-características [32]:

- *Analizabilidad*: capacidad del producto de *software* de ser diagnosticado para detectar las deficiencias o causas de fracasos, o para identificar las partes a ser modificadas.
- *Cambiabilidad o Modificabilidad*: capacidad del producto de *software* para permitir que una modificación específica sea implementada.
- *Estabilidad*: capacidad del producto de *software* para evitar los efectos inesperados de las modificaciones realizadas.
- *Capacidad de prueba*: capacidad del producto de *software* para permitir que el *software* modificado sea validado.

- *Cumplimiento*: capacidad del producto *software* para satisfacer las normas o convenciones relacionadas con la mantenibilidad.

1.3.2. Productividad, Satisfacción del usuario e Intención de adopción

Productividad

La productividad, definida como la cantidad de recursos usados en relación con los artefactos producidos; se caracteriza por ser uno de los valores principales a tener en cuenta para planificar un proyecto de *software* debido a su impacto durante la selección de un proceso de desarrollo en la industria [12]. Las ganancias de productividad son, a menudo, una las principales motivaciones para seleccionar nuevas tecnologías.

Con frecuencia se confunden entre sí los términos productividad, eficiencia y efectividad. Méndez *et al.* [52] definen y analizan cada término semánticamente de manera tal que establecen la productividad como la combinación de la efectividad (relacionada con el desempeño) y la eficiencia (la utilización de los recursos). Una definición más amplia de cada término aparece a continuación:

- Efectividad, sinónimo de eficacia: capacidad de lograr el efecto que se desea o se espera [72], es el "para qué se hace", es decir, la relación entre las metas puestas y las metas logradas. La efectividad entonces sería la capacidad de cumplir un objetivo trazado [1].
- Eficiencia: capacidad de disponer de alguien o de algo para conseguir un efecto determinado [72], es decir, es el "cómo se hace". También puede conceptualizarse como la relación entre los resultados obtenidos y los recursos empleados [1]. Si con determinado esfuerzo obtienes buenos resultados, se dice que eres una persona eficiente. Dicho de otra forma, la eficiencia es realizar una actividad o un trabajo al menor costo posible y en el menor tiempo, sin desperdiciar recursos económicos, materiales ni humanos, obteniendo un resultado de calidad.

Por otro lado, la productividad sirve para evaluar el rendimiento de los talleres, las máquinas, los equipos de trabajo y los empleados. El rendimiento como uno de los atributos de calidad, se define como "el grado al que un sistema de software cubre sus objetivos en

tiempo" [78], se refiere a la proporción entre el producto o el resultado obtenido y los medios utilizados [72].

Cabe destacar que el rendimiento también se encuentra vinculado al concepto de eficiencia o al de efectividad. Moody [58], respalda esta aseveración al expresar que el rendimiento puede ser mejorado de dos formas:

- Eficiencia: al reducir el esfuerzo necesario para completar la tarea.
- Efectividad: mejorando la calidad del resultado.

Satisfacción del usuario

Por otro lado, un usuario se define como una persona que parte de su trabajo regular es utilizar el sistema de información, método o herramienta producida [6]. Es aquella persona para quien se construyen realmente las tecnologías o métodos, y no tiene por qué ser necesariamente un profesional del *software*.

La satisfacción del usuario, sin embargo, ha sido un tema controvertido, y se ha conceptualizado como un resultado y como un proceso, como una respuesta tanto cognitiva como emocional. Una definición ampliamente aceptada es proporcionada por Oliver [64] quien la define como "una respuesta o evaluación post-consumo elicitada por factores tanto afectivos como cognitivos".

Intención de adopción

Uno de los objetivos fundamentales que tienen que ser resueltos en la IS es cómo determinar si un método es exitoso o no. Moody [58] argumenta que (al menos) hay dos dimensiones de "éxito" que tienen que ser consideradas para valorar los métodos de diseño en la IS: la efectividad real -si el método mejora el rendimiento de la tarea- y adopción en la práctica -si el método es usado en la práctica-. Es decir, un método que mejora el rendimiento pero que no es usado, no tendrá ningún efecto en las prácticas. De forma semejante, un método que las personas usan pero que reduce el rendimiento de la tarea tendrá un efecto en contra de las prácticas.

Existe un conjunto de modelos que establecen los factores determinantes de la aceptación de las tecnologías por los usuarios. Entre ellos destacan: TAM [15], TAM2 [86], PCI [59], TPB [3], y MPCU [81] por sus fundamentos teóricos. El principal propósito de estos modelos es explicar el comportamiento de un individuo hacia el uso de la computadora.

Más cercano a los objetivos de esta tesis, el Modelo de Evaluación de Métodos (*Method Evaluation Model*) [58], establece cuatro principales dimensiones: rendimiento, percepción, intención y comportamiento. Este modelo tiene como principal objetivo valorar los métodos que incluyen ambos aspectos de éxito (efectividad real y adopción en la práctica) mencionados anteriormente. Tal marco teórico (basado en [15]) apunta que el rendimiento de un desarrollador influye en su percepción, y esta a su vez contribuye a su adopción en la práctica.

En [73], se lleva a cabo un estudio comparativo donde se analizan cuán bien estos modelos predicen la intención de usar un método en particular. En él se demuestra empíricamente cómo algunos conceptos (variables) de adopción de tecnologías son igualmente eficaces para pronosticar la adopción de un método (p.e. utilidad, compatibilidad, norma subjetiva y voluntariedad), mientras que otros, dada las diferencias entre asumir un método y una herramienta, pueden ser dejados de lado (p.e. control del comportamiento percibido, etc.). Este estudio también desestima el impacto de la facilidad de uso sobre la adopción de un método. Esta desestimación realmente es sorprendente, ya que la facilidad de uso es uno de los componentes principales del modelo TAM [15] y está altamente correlacionado con la intención de uso en otros conocidos modelos teóricos de adopción de tecnologías [21, 54, 58, 87].

1.4. Evaluación Empírica

Los estudios empíricos tiene un papel fundamental en la ciencia moderna, pues ayudan a comprender cómo y por qué trabajan las cosas, y permiten que se use este conocimiento para cambiar el entorno de trabajo considerablemente. Son útiles por tanto para conseguir información para tomar decisiones bien fundamentadas. Según Perry *et al.* [66] "un

estudio empírico es realmente sólo una prueba que compara lo que se cree con lo que se observa". Los mismos autores proponen que los estudios empíricos tengan los siguientes componentes: contexto de investigación, hipótesis, diseño experimental, amenazas para la validez, análisis de los datos y su presentación, así como los resultados y conclusiones. Muchos autores han escrito acerca de la importancia de proporcionar evidencia empírica en la IS [17, 36]. Esta evidencia puede ser proporcionada en forma de encuestas, experimentos, estudios de casos o análisis *post-mortem* [89]. Concretamente en esta tesis se centra en la obtención de evidencias a partir de experimentos.

Un experimento se define como "una investigación donde las posibles variables perturbadoras han sido aleatorizadas" [35, 51]. Es un procedimiento mediante el cual se trata de comprobar (confirmar o verificar) una o varias hipótesis relacionadas con un determinado fenómeno, mediante la manipulación de las variables que presumiblemente son su causa. La experimentación (o evaluación experimental) constituye uno de los elementos claves del método científico y es fundamental para poder ofrecer explicaciones causales.

La evaluación empírica de las actividades de productividad, mantenibilidad y satisfacción en metodologías MDE ha ido ganando impulso en los últimos años. Esta afirmación se apoya en el mapeo sistemático llevado a cabo en [44], que recoge la evidencia empírica sobre la mejora de calidad del producto de *software* y productividad del proceso de desarrollo, durante el período 2001 a 2010 (ver Sección 1.2). Sin embargo, todavía existe muy poca evidencia empírica que avale sus beneficios y limitaciones con respecto a las prácticas basadas en modelos (MBD) o centradas en el código (*code-centric*).

El diseño de los experimentos que forman parte de esta tesis se basan en el conocido *framework* de experimentación sugerido por Wohlin [89], mientras que los análisis estadísticos se han realizado con el PASW *Predictive Statistics Application* v18 [62]. A continuación se resumen los principales experimentos realizados, sus preguntas de investigación, hipótesis a probar, limitaciones (amenazas de validez) y principales resultados encontrados.

1.4.1. Experimento de mantenibilidad: WebML vs PHP

El *objetivo* de este estudio empírico, siguiendo la plantilla *Goal/Question/Metric* (GQM) [66], es analizar WebML [9] y PHP *con el fin de* evaluar el uso de MDE contra las prácticas de mantenimiento centradas en el código, *con respecto a* su rendimiento y satisfacción *desde el punto de vista* de los desarrolladores noveles y los prospectivos adoptadores. El *contexto del estudio* es de dos cursos avanzados de IS en el Politecnico de Milano y la Universidad de Alicante, donde es enseñado un proceso de desarrollo Web usando MDE con la metodología WebML y un proceso de desarrollo *code-centric* con el lenguaje PHP, respectivamente. Las preguntas de investigación que apoyan el objetivo son:

RQ1: ¿Es el rendimiento de la mantenibilidad (eficiencia y efectividad real) de la metodología WebML mayor que usando PHP como método de desarrollo tradicional centrado en el código?

RQ2: ¿Es la satisfacción de realizar tareas de mantenibilidad (facilidad de uso y utilidad percibidas) de la metodología WebML mayor que usando PHP como método de desarrollo tradicional centrado en el código?

Planificación del experimento

El experimento se aplica durante el segundo período del año 2009-2010, seleccionándose por conveniencia a 32 estudiantes inscritos en el curso de "Tecnologías Web" en el Politécnico de Milano (Italia) y a 12 estudiantes inscritos en el curso de "Programación Web" en la Universidad de Alicante (España), quienes tenían experiencia previa en el desarrollo con PHP. Los preparativos necesarios para el experimento y las tareas experimentales quedaban al alcance de los cursos, pues entre sus objetivos se incluía enseñar a los estudiantes cómo mantener actualizadas aplicaciones en WebML y PHP, respectivamente. La diseminación geográfica de los estudiantes hizo necesario realizar un diseño experimental entre-sujetos (ver Tabla 1–1).

Tabla 1–1: Diseño experimental.

Grupo	Aplicación	Metodología
Milano	*Concert*	MDE
Milano	*Bookstore*	MDE
Alicante	*Concert*	*code-centric*
Alicante	*Bookstore*	*code-centric*

Para ambas partes del experimento se utilizaron dos muestras de aplicaciones Web para evaluar las hipótesis: una aplicación de concierto (*Concert*, adaptación de un ejemplo usado en [2]) y una aplicación de librería (*Bookstore*, adaptación de un ejemplo usado en [9]). Los sujetos dentro de cada grupo fueron asignados aleatoriamente a cualquiera de estas aplicaciones. Para hacer los resultados obtenidos en Milano (con WebML) y Alicante (con PHP) comparables, los instrumentos de los experimentos se elaboraron tan similares como fue possible en relación al diseño y al contenido de la información. El tiempo límite de realización del experimento fue de dos horas -verificado a través de una prueba piloto- y además, se controló que no existiera interacción alguna entre los participantes.

La variable o factor independiente (VI), manipulada experimentalmente de este estudio es la metodología utilizada para realizar las tareas de mantenibilidad: MDE (metodología WebML) o *code-centric* (tareas que se realizan directamente en el código PHP). Por lo tanto, en el experimento se definen dos tratamientos: las tareas de mantenibilidad sobre los modelos WebML y las tareas de mantenibilidad sobre el código PHP. Los datos recogidos experimentalmente permiten comparar los efectos de ambos tratamientos.

El modelo teórico que subyace a la definición de las Variables de Dependientes (VD) y la formulación de las hipótesis de este experimento, constituye una adaptación de [58] y presentado en [43]. Según este modelo, el rendimiento real de los desarrolladores influye en su satisfacción, y esto a su vez contribuye a su intención de utilizar el método y, eventualmente, a su uso real.

Como VD para comparar los tratamiento, se definen dos tipos: basadas en el rendimiento (precisión, exhaustividad -más conocida por *recall*- y tiempo) y relacionadas con la satisfacción (complejidad, certeza de los resultados y estabilidad percibida de las tareas

correctivas y perfectivas). Además, en el post-cuestionario de este experimento se miden la satisfacción global (S) como la media de 11 pares de adjetivos que presentan un alto grado de fiabilidad ($\alpha = 0{,}861$) y el aprendizaje (PL). Todas estas medidas han sido usadas para probar las siguientes hipótesis:

- **HAA** (Analizabilidad real): AA(MDD)>AA(*code-centric*), analizar la mantenibilidad sobre modelos WebML permite un mejor rendimiento que analizarla directamente sobre el código PHP.
- **HACC** (Cambiabilidad correctiva real): ACC(MDD)>ACC(*code-centric*), corregir los errores sobre modelos WebML permite un mejor rendimiento que corregirlos directamente sobre el código PHP.
- **HAPC** (Cambiabilidad perfectiva real): APC(MDD)>APC(*code-centric*), mejorar una aplicación Web sobre modelos WebML permite un mejor rendimiento que mejorarla directamente sobre el código PHP.
- **HPA** (Analizabilidad percibida): PA(MDD)>PA(*code-centric*), los sujetos sienten que encontrar los errores sobre modelos WebML es menos complicado que encontrar los errores sobre el código PHP. Asimismo, se sienten más seguros de sus resultados.
- **HPCC** (Cambiabilidad correctiva percibida): PCC(MDD)>PCC(*code-centric*), los sujetos sienten que corregir los errores sobre modelos WebML es menos complicado y más estable que hacerlo directamente sobre el código PHP; también se sienten más seguros sobre los resultados.
- **HPPC** (Cambiabilidad perfectiva percibida): PPC(MDD)>PPC(*code-centric*), los sujetos sienten que realizar las mejoras sobre modelos WebML es menos complicado y más estable que hacerlo directamente sobre el código PHP; también se sienten más seguros sobre los resultados.
- **HS** (Satisfacción): S(MDD)>S(*code-centric*), en términos generales, los sujetos se sienten más satisfechos cuando realizan las tareas de mantenibilidad con WebML que con PHP.
- **HPL** (Aprendizaje): PL(MDD)<PL(*code-centric*), en términos generales, los sujetos creen que PHP es más fácil de aprender que WebML.

Análisis de los datos e interpretación de los resultados

Por el bien de la confiabilidad de los resultados del análisis estadístico y debido al bajo número de sujetos en el grupo que realizaron las tareas de mantenibilidad en PHP, se aplica la prueba no paramétrica U de *Mann Whitney*, que no hace suposición acerca de la normalidad de la distribución de los datos. Todos los análisis fueron llevados a cabo en el PASW *Statistics Application* v18 [62]. Es importante notar que no todas las tareas fueron realizadas por todos los sujetos, razón por la cual los procedimientos de pruebas de las diferentes hipótesis pueden presentar diferentes grados de libertad. En el Anexo B se aprecian los resultados de la estadística descriptiva -medias y desviación estándar (DS)- de cada una de las variables analizadas.

Para darle respuesta a la **RQ1: *Rendimiento real***, se evalúan las siguientes hipótesis HAA, HACC, HACP, relacionadas con al precisión, exhaustividad y el tiempo para realizar las tareas de analizabilidad y cambiabilidad correctivas y perfectivas.

El resultado de la prueba de hipótesis HAA muestra que ambos grupos difieren significativamente en la precisión ($U(41) = 80{,}5; Z = 3{,}011; p = 0{,}003$) y el tiempo ($U(41) = 345{,}5; Z = 4{,}609; p < 0{,}001$), y en ambos casos el grupo de WebML obtuvo un mejor rendimiento que el grupo de PHP. Sin embargo, la exhaustividad de la analizabilidad no era significativamente diferente ($U(41) = 170; Z = 0{,}286; p = 0{,}775$). En otras palabras, *llevar a cabo tareas de analizabilidad sobre WebML parece acelerar la identificación de los errores, y evita clasificar mal una característica como un error en la aplicación. No obstante, no parece ayudar significativamente para detectar los errores en la aplicación.*

Relacionado con HACC, el resultado revela que, otra vez, ambos grupos difieren significativamente en la precisión ($U(40) = 104; Z = 2{,}163; p = 0{,}031$) y en el tiempo ($U(39) = 252; Z = 2{,}484; p = 0{,}013$), donde el grupo de WebML mostró mejor rendimiento que el grupo PHP para ambos casos. La exhaustividad de la cambiabilidad correctiva (es decir el número de los errores realmente corregidos del número total de los errores) no era significativamente diferente ($U(40) = 132; Z = 1{,}31; p = 0{,}19$). Es decir, *llevar a cabo*

las tareas de cambiabilidad correctivas con WebML parecen acelerar la corrección de los errores, y evita proponer las rectificaciones que no corrigen nada en realidad. Sin embargo, no parecen ayudar significativamente para corregir los errores reales en la aplicación.

Por otro lado, los resultados de la prueba de hipótesis HAPC indican que, la precisión del grupo WebML es ligeramente más alta que la del grupo PHP ($U(40) = 137{,}5; Z = -1{,}22; p = 0{,}222$), aunque no significativamente. De forma semejante se comporta la exhaustividad ($U(40) = 165; Z = -0{,}283; p = 0{,}777$), no siendo significativas las diferencias entre las medias de los dos grupos al realizar las tareas de cambiabilidad perfectivas. Sin embargo, otra vez, los sujetos fueron significativamente más rápidos ($U(40) = 336; Z = 4{,}647; p < 0{,}001$) cuando realizan mejoras con WebML que con PHP.

En la **RQ2: *Satisfacción de los tratamiento***, se prueban las siguientes hipótesis HPA, HPCC, HPPC, HS y HP; relacionadas con la complejidad, certeza y estabilidad percibida por los sujetos cuando realizan las tareas de analizabilidad y cambiabilidad correctivas y perfectivas.

Respecto a HPA, el resultado de la prueba muestra que ambos grupos difieren significativamente en la valoración de la complejidad percibida ($U(37) = 84; Z = -2{,}195; p = 0{,}028$), pero no de la certeza percibida ($U(37) = 92; Z = -1{,}871; p = 0{,}061$) de los resultados. *Los sujetos estiman el trabajo sobre modelos como más simple que trabajando directamente sobre código, pero ellos se sienten igualmente seguros/inseguros de los errores detectados.*

Los resultados de la prueba de hipótesis HPCC revela que los usuarios perciben los cambios correctivos cuando usaban PHP ligeramente menos complicados que cuando utilizan WebML ($U(36) = 140, Z = -0{,}179, p = 0{,}858$), aunque esta diferencia no era significativa. También, los usuarios se sentían ligeramente más seguros sobre sus cambios en WebML ($U(36) = 125; Z = -0{,}355; p = 0{,}723$). Por último, aunque no por ello menos importante, debido a que en este caso los usuarios estaban cambiando realmente la aplicación, se pregunta por la estabilidad percibida. Los usuarios sentían que

la estabilidad del sistema era significativamente mejor conservada cuando usaban PHP ($U(36) = 224, Z = 3{,}152, p = 0{,}002$). O sea, *los usuarios parecen sentirse más seguros sobre los cambios realizados con WebML, pero parecen sentirse más "control" de los cambios cuando trabajan directamente en el código PHP.*

Sin embargo, los resultados de HPPC indican que WebML fue considerado ligeramente más simple para realizar los cambios perfectivos que PHP, aunque la prueba estadística no era significativa ($U(37) = 105; Z = -1{,}55; p = 0{,}121$). Por lo contrario, los sujetos que usaban PHP se sentían ligeramente más confiados sobre la corrección de los cambios realizados, aunque, otra vez, la diferencia no era significantivas con respecto al grupo de WebML ($U(37) = 187{,}5; Z = 1{,}31; p = 0{,}19$). Respecto a la estabilidad, ambos grupos mostraron niveles similares de confianza sobre la estabilidad de la solución ($U(36) = 136{,}5; Z = 0{,}053; p = 0{,}958$).

Respecto a HS, como se ha mencionado anteriormente, el cuestionario post-experimento incluía una balanza de la satisfacción, que permite comparar los índices de satisfacción de ambos tratamientos con una prueba T (*T-test*). Los resultados muestran un efecto no significativo de la metodología ($t(38) = -0{,}208; p = 0{,}836$). En resumen, *las evaluaciones eran medianamente positivas para ambas metodologías, con una diferencia media de* 0,06 *favoreciendo ligeramente el grupo de PHP.*

Por último se evalúa HPL, el índice de la satisfacción global (S) está altamente correlacionado a la medida del aprendizaje (PL) de la metodología ($r = 0{,}663; p < 0{,}001$). Esto indica que *mientras más fácil la metodología fue percibida, mejor fue valorada.*

Amenazas para la validez

Una cuestión fundamental concerniente a los resultados de un experimento es cuán válidos son estos resultados. En general una validez adecuada se refiere a que los resultados son válidos para la población de interés. En [14] se delimitan cuatro tipos de amenazas de validez de los estudios empíricos: interna, externa, de constructo y de conclusión. En cada estudio empírico realizado se han tenido en cuenta estas amenazas, que permiten valorar

las condiciones en las que son aplicables los experimentos, los beneficios que ofrece y bajo qué circunstancias podrían fallar.

Las amenazas para la validez de conclusión, se refieren a la relación entre los tratamientos y las salidas. En este experimento se utilizan las líneas de código para controlar el tamaño de la aplicación, medida que es preferida sobre los puntos de función debido a la alta confiabilidad [89]. Adicionalmente, las pruebas estadísticas han sido escogidas conservadoramente, sin hacer ningún tipo de suposición en las distribuciones de las variables. Sin embargo, teniendo en cuenta el limitado tiempo que los estudiantes tenían para realizar el cuestionario del experimento, es posible que los sujetos hayan sentido algún tipo de presión que pueda afectar a los datos. No obstante, debido a que ambos grupos sufrían de las mismas limitaciones de tiempo, se puede suponer que tal efecto, si se presenta, ha afectado todos los niveles del tratamiento equitativamente.

Las amenazas para la validez interna, concernientes con la posibilidad de que existan factores escondidos que puedan comprometer la conclusión que es efectivamente el tratamiento el que causa las divergencias en el resultado. El hecho que los estudiantes pertenecían a diferentes países podía predisponer los resultados. Este efecto de falta de aleatoriedad de los sujetos asignados a los tratamientos es limitado, garantizando que la edad media para el grupo de estudiantes de WebML ($M = 22{,}47$) no fuera significativamente diferente de la edad de media para el grupo de estudiantes de PHP ($M = 23{,}17$). También se garantiza, que los sujetos habían sido inscritos en un número similar de cursos relacionados sobre los temas de modelado/programación, y habían desarrollado un número similar de aplicaciones Web ($M(WebML) = 3{,}93; M(PHP) = 4{,}25; t(27) = -0{,}183; p = 0{,}18$). Solamente la experiencia con los métodos no pudo ser controlada ($M(WebML) = 7{,}27, M(PHP) = 69{,}27$); esta diferencia puede ser explicada por la estructura de las carreras en las universidades, que se concentran más en programación que en las destreza de modelado.

Por otro lado, para tratar de disminuir el efecto de diferentes facilitadores se realiza el experimento en las dos ubicaciones, teniendo en cuenta las instrucciones que evitaron las diferencias no intencionadas en las condiciones del experimento. Otra amenaza para

la validez interna de este estudio está relacionada con los instrumentos. Debido a que las medidas de precisión y exhaustividad tuvieron que ser manualmente computadas por diferentes evaluadores (cada uno por un experto en WebML y PHP respectivamente), existe el riesgo de que los criterios fueran aplicados de manera diferente a cada grupo. Este riesgo se ha tratado de evitar definiendo una taxonomía clara de los errores que fueron acordados antes de la evaluación.

Las amenazas para la validez del constructo, se refieren a la relación entre la teoría y la observación. En este sentido, tanto los tratamientos como las medidas utilizadas para valorar la mantenibilidad han sido ampliamente usados en la literatura. No obstante, todavía se pueden correr algunos riesgos. Primero, se han usado aplicaciones de tamaño similar, que pertenecen al mismo dominio (prejuicio de una sola medida u observación). Por lo tanto no se pueden generalizar los resultados a aplicaciones de tamaños o dominios diferentes. Segundo, se ha observado un resultado positivo entre la mantenibilidad y WebML, pero no se puede garantizar que usar WebML no entorpece otras características de calidad (generalizabilidad restringida).

Por último, las amenazas para la validez externa, están relacionadas con la generalizabilidad de los resultados. En este caso, el hecho de que se han usado estudiantes universitarios, es decir, sujetos sin experiencia en la industria de desarrollo, así como tanto las metodologías específicas y los lenguajes usados, constituyen un ambiente limitado; riesgo inevitable para este tipo de experimentos, en los cuales los sujetos tienen que usar un enfoque predeterminado para llevar a cabo las tareas. La más fácil manera de minimizar esta amenaza es a través de la reproducción del experimento con lenguajes y enfoques MDE diferentes.

1.4.2. Experimento de productividad y satisfacción

Durante los meses de enero y febrero 2011, un experimento fue dirigido y realizado en la Universidad de Alicante. A raíz de la plantilla GQM [66], el estudio empírico tiene como

objetivo analizar .NET (sobre la base de *C#* con *Visual Studio* .NET 2010), UML (soportado por la herramienta RSM -*Rational Software Modeler*-, para comprender los objetivos del sistema, pero sin generación de código) y OOH4RIA (con la herramienta OOH4RIA IDE) *con el propósito* de evaluar los enfoques centrados en el código (*code-centric*), basado en modelo (MBD) y dirigido por modelos (MDE) *con respecto a* su productividad y satisfacción *desde el punto de vista* de desarrolladores de *software* noveles. *El contexto del estudio* es un grupo de estudiantes de maestría que desarrollan la lógica de negocio de una aplicación Web 2.0. Las preguntas de investigación que permiten respaldar el objetivo de esta experimentación son:

RQ1: ¿Es la productividad del equipo significativamente diferente entre los métodos, sin considerar el módulo de la aplicación que están desarrollando?

RQ2: ¿Es la satisfacción de los desarrolladores significativamente diferente entre los métodos, sin considerar el módulo de la aplicación que están desarrollando?

Planificación del experimento

Para responder a las preguntas de investigación, se definen las siguientes VI: Meth (método, variable categórica con tres niveles: *code-centric*, MBD, MDE) y Mod (módulo, una variable categórica con tres valores posibles: Grupos, Eventos y Organización de una aplicación Web 2.0). Las VD son: la productividad del equipo con cada método y módulo -P(Meth, Mod)- y la satisfacción de los desarrolladores -S(Meth, Mod)-. Las medidas han sido usadas para probar las siguientes hipótesis:

- **HP** (Productividad): P(MDD)>P(MBD)>P(*code-centric*), los equipos desarrolladores son significativamente más productivos con el método MDE (OOH4RIA), seguido del método MBD (UML), seguido del *code-centric* (.NET).
- **HS** (Satisfacción): S(MDD)>S(MBD)>S(*code-centric*), los desarrolladores están significativamente más satisfechos con el método MDE (OOH4RIA), seguido del método MBD (UML), seguido del *code-centric* (.NET).

El experimento se realiza con 30 individuos divididos en seis grupos que desarrollan tres módulos específicos de una red social (dominio de la aplicación Web 2.0), cada uno aplicando diferentes métodos. Los datos que correspondían al equipo 6 tuvieron que ser retirados debido a que algunos de sus componentes abandonaron la maestría por razones de trabajo poco después de que el experimento había empezado. Por lo tanto, la muestra final es de 26 sujetos. En la Tabla 1–2 se refleja el diseño experimental.

Tabla 1–2: Diseño experimental: factorial e intra-sujetos.

Equipo-Módulo	Aplicación	Grupo	Eventos	Organización
1	*Trips*	*code-centric*	MBD	MDE
2	*Events*	*code-centric*	MDE	MBD
3	*Hospital*	MBD	MDE	*code-centric*
4	*Academics*	MDE	MBD	*code-centric*
5	*Facework*	MBD	*code-centric*	MDE
6*	*Automobile*	MDE	*code-centric*	MBD
(El grupo marcado con * no terminó el experimento)				

Análisis de los datos e interpretación de los resultados

Para el análisis estadístico, se aplica el $3 * 5$ *Mixed Design ANOVA* -dado el alto grado de robustez del ANOVA para violaciones en algunas de sus asunciones [61]- para un diseño factorial, en los cuales el módulo (Grupos, Eventos y Organización) se considera como variables entre-sujetos, y la P y S calculada por cada método se considera la variable intra-sujeto (indistintamente). Los resultados de la estadística descriptiva -medias y DS- aparecen en Tabla 1–3.

Tabla 1–3: Estadística descriptiva de las variables S y P

Variables	***code-centric***		**MBD**		**MDE**	
	Media	DS	Media	DS	Media	DS
HP	0,80	0,29	2,30	1,10	4,60	1,17
HS	4,17	0,72	3,48	0,96	4,76	0,73

Los resultados del análisis estadístico indican que en ninguna de las hipótesis probadas se detecta que la interacción entre Mod*Meth es significativa, pasando a probar cada variable (Mod y Meth) independientemente, sin tener que matizar los resultados por la existencia de una interacción significativa.

Para darle respuesta a la primera pregunta de investigación (**RQ1: *Impacto del método sobre la productividad del equipo***), y en aras de garantizar que aplicar este método estadístico tiene sentido, primero se verifica que no se viole el principio de esfericidad aplicando la prueba W de *Mauchly* ($W = 0{,}005; p > 0{,}05$) [47]. Los resultados muestran que OOH4RIA produjo una productividad más alta, seguida de UML y luego .NET (ver Tabla 1–3), y que estas diferencias son significativas ($F = 25{,}395; p = 0{,}001$). En cambio, el efecto principal del módulo no fue significativo ($F = 0{,}538; p > 0{,}05$). Es decir, *las diferencias en P están significativamente afectadas por el método usado, sin considerar el módulo especial desarrollado.*

En cuanto a la **RQ2: *Impacto del método sobre la satisfacción del desarrollador***, de nuevo la prueba W de *Mauchly* ($W = 0{,}838; p = 0{,}142$) [47] garantiza la no violación del principio de esfericidad. Luego, los resultados revelan diferencias significativas ($F = 18{,}04; p < 0{,}01$) donde OOH4RIA produjo los resultados más altos, seguidos del desarrollo .NET y por último UML. Por otro lado, se puede observar que la influencia de Mod, nuevamente, no fue significativa ($F = 0{,}167; p > 0{,}05$). *Las diferencias en S están significativamente afectadas por el método usado, sin considerar el módulo especial desarrollado.*

Amenazas para la validez

El análisis de las amenazas para la validez valora bajo qué condiciones es aplicable y brinda beneficios el experimento, y bajo qué circunstancias podría fallar. Las amenazas descritas a continuación siguen las pautas marcadas en [14] y descritas en la Sección 1.4.1.

Amenazas a la validez de conclusión, en este sentido se ha intentado capturar la mayor cantidad de medidas como fue posible automáticamente, con la ayuda de sistemas de rastreos (seguimiento) conocidos como JIRA o SVN. Adicionalmente los *test* estadísticos han sido seleccionados conservadoramente, sin hacer ningún tipo de suposición sobre la distribución de las variables. Sin embargo, el hecho de que los estudiantes reportaron sus

propias medidas, junto con la duración del experimento (seis semanas) y el bajo número de sujetos, limitan la validez de las conclusiones.

Para minimizar las amenazas a la validez interna, todos los grupos aplicaron todos los tratamientos a módulos diferentes en distintos momentos, lo que reduce muchas amenazas internas como la selección, la historia, la maduración o amenazas sociales como la rivalidad compensatoria o la desmoralización resentida. Sin embargo, siendo un diseño intra-sujetos, otros efectos podrían haber ocurrido.

Relacionadas con las amenazas a la validez del constructo, los tratamientos y las medidas para valorar la productividad y la satisfacción han sido ampliamente usadas en la literatura. No obstante, queda la posibilidad de una interacción de la prueba y los tratamientos, es decir, la necesidad de que los propios estudiantes informaran sobre ciertas medidas podrían haber cambiado su comportamiento. Por otro lado, el hecho de que el experimento tomó más de seis semanas minimiza este riesgo, ya que es muy difícil mantener un comportamiento "potencialmente anormal" durante un largo período de tiempo sin ser detectado. También, las hipótesis del experimento (es decir, mayor productividad y satisfacción en ambientes MDE) eran muy fáciles de adivinar, así que los estudiantes podrían haberse sentido influenciados a reportar menos tiempo cuando usan MBD o MDE. De todos modos, los observadores del experimento tuvieron especial cuidado para no revelar estas hipótesis a los estudiantes. Adicionalmente, el experimento se ve afectado por una generalizabilidad restringida entre los conceptos: se ha verificado un resultado positivo entre la productividad y MDE, pero no se puede garantizar que no se vea afectado por otras características del *software* desarrollado, como la modularidad, la reuzabilidad, o cualquier otro atributo de calidad.

Por último, se han identificado un conjunto de amenazas para la validez externa como: falta de representatividad de la muestra (estudiantes de maestría), ambiente académico, una arquitectura estricta y un dominio y complejidad restringida. También, debido al acoplamiento existente entre el método y la herramienta, todos los métodos fueron acompañados por herramientas específicas. Aunque se trató de seleccionar ambientes de

desarrollo ampliamente conocidos y -cuando fue posible- usar estándares (p.e. UML para la actividad de modelado en MBD), el uso de las diferentes herramientas añaden un "sabor especial" a los métodos. Por lo tanto, este experimento tiene que ser reproducido para asegurar que es el método usado y no la herramienta la causa de las diferencias observadas.

1.4.3. Experimento de satisfacción e intención de adopción

Durante los meses de enero y febrero de 2011, un experimento controlado se lleva a cabo en la Universidad de Alicante *con el objetivo* de valorar las intención de adopción, *desde el punto de vista* de desarrolladores de *software* noveles usando tres enfoques diferentes: *code-centric* (sobre la base de *C#* con *Visual Studio* .NET 2010), MBD (basado en UML y soportado por la herramienta RSM, para comprender los objetivos del sistema, pero sin generación de código) y MDE (utilizando OOH4RIA [48] y su herramienta OOH4RIA IDE [49]). *El contexto de estudio* son un grupo de estudiantes de maestría que desarrollan la lógica de negocio de una aplicación Web 2.0. La elección de herramientas específicas para cada método disminuye la generalizabilidad de los resultados, pero tal selección es necesaria para reproducir un ambiente real en el que cada método es usualmente aplicado e incrementar la validez de constructo del experimento. Las preguntas de investigación para dirigir este estudio son las siguientes:

RQ1: ¿La utilidad del método, percibida por el desarrollador, es significativamente diferente entre los métodos, sin considerar la aplicación desarrollada?

RQ2: ¿La facilidad de uso del método, percibida por el desarrollador, es significativamente diferente entre los métodos, sin considerar la aplicación especial desarrollada?

RQ3: La compatibilidad con el método, percibida por el desarrollador, es significativamente diferente entre los métodos?

RQ4: ¿Están las medidas utilidad percibida, facilidad de uso percibida y la compatibilidad correlacionadas?

RQ5: ¿Las intenciones de adopción de tecnología de los desarrolladores son diferentes significativamente entre los tres enfoques (*code-centric*, MBD y MDE), sin considerar la aplicación particular desarrollada?

RQ6: ¿Cuáles son las principales ventajas/desventajas percibidas de cada enfoque?

Estas preguntas de investigación tienen como precedente las dimensiones o componentes principales del Modelo de Evaluación de Métodos [58] y además, permite refinar el modelo TAM [15] a partir de la experiencia ganada en el estudio comparativo de los modelos de adopción de tecnologías [73]. Las variables conceptuales que conforman la dimensión percepción incluyen: Utilidad percibida (PU), Facilidad de uso percibida (PEU), Compatibilidad percibida (PC), Norma subjetiva (SN) y Voluntariedad (V). Estas variables guardan una influencia directa con la intención de adoptar un método de desarrollo determinado. El experimento limita su estudio a las tres primeras variables conceptuales. Las medidas asociadas a estas variables han sido agrupadas en escalas, calculando estadísticamente su fiabilidad y correlación:

- Para la escala PU, todos los *items* indican una correlación mayor que 0,3 y el *alpha de Cronbach* global es de 0,817; demostrando suficiente consistencia interna de los datos.
- Para la escala PEU, todos los *items* indican una correlación más alta que 0,3 y un *alpha* de Cronbach global de 0,896.
- Para la escala PC, se encontraron muy bajos niveles de correlación (aunque significativos) entre la percepción del desarrollador de la rareza de un método y su nivel de experiencia reportado con ese método. En este sentido, se opta por depender únicamente del nivel de la experiencia informada para medir PC en los análisis siguientes. Esta decisión es tomada, desde un punto de vista personal, pues la experiencia previa refleja la definición de la compatibilidad dada en el modelo teórico propuesto en [45] más evidentemente.

Planificación del experimento

Para responder a las preguntas de investigación, se definen las siguientes VI manipuladas experimentalmente: Meth (método, variable categórica con tres niveles: *code-centric*, MBD, MDE) y Aplicación -dominio de aplicación-, una variable categórica con cinco valores posibles: Viajes, Hospitales, Eventos, Académico, *Facework*. Las VD definidas son: PU (utilidad percibida de cada método), PEU (facilidad de uso percibida de cada método),

PC (compatibilidad de cada método) e IA (intención de asumir un método en particular). Estas medidas han sido usadas para probar las siguientes hipótesis:

- **HPU** (Utilidad percibida): PU(MDE) > PU(MBD) > PU(*code-centric*). La utilidad percibida de los métodos difiere significativamente. Este hecho reside, sin considerar la aplicación real que están desarrollando.
- **HPEU** (Facilidad de uso percibida): PEU(MDE) < PEU(MBD) < PEU(*code-centric*). La facilidad de uso percibida de los métodos difiere significativamente. Este hecho reside, sin considerar la aplicación real que están desarrollando.
- **HPC** (Compatibilidad percibida): PC(MDE) < PC(MBD) < PC(*code-centric*). La compatibilidad percibida de los métodos difiere significativamente. Este hecho reside, sin considerar la aplicación real que están desarrollando.

Por último, la IA se mide indirectamente, pues luego de terminado el experimento se le pide al equipo que seleccione el método preferido para continuar el resto de los módulos del proyecto asignado.

Análisis de los datos e interpretación de los resultados

El experimento se realiza a través de un diseño factorial intra-sujetos, con 26 estudiantes divididos en cinco grupos, donde cada grupo desarrolla una red social para los cinco diferentes dominios que permanecieron en el diseño factorial del experimento de productividad (ver Tabla 1–2, Sección 1.4.2).

Respecto al nivel de experiencia de los sujetos con las diferentes tecnologías y métodos usados durante el experimento, el cuestionario muestra que el 81 % tenían conocimientos de UML, mientras que el 12 % consideraban que tenían un alto nivel. Además, el 76 % de los sujetos habían programado con *Visual Studio* (VS) y tecnología .NET durante sus cursos de pregrado, aunque solamente 12 % lo habían aplicado con anterioridad en la industria. Definitivamente, los sujetos no reconocieron tener conocimientos prácticos previos de MDE, aunque el 56 % de ellos eran conscientes de la existencia del paradigma.

Nuevamente, teniendo en cuenta el diseño experimental planificado, se aplica el $3 * 5$ *Mixed Design ANOVA* para probar HPU y HPEU, mientras que para HPC se deja de lado el factor inter-sujeto y se aplica un *One-Way RM ANOVA*, teniendo en cuenta que todas las aplicaciones tienen los mismos requisitos funcionales y pertenecen al mismo dominio, la compatibilidad del método no depende de la aplicación desarrollada, por lo que se precisa una transformación sobre una escala de 3-puntos (1 = bajo, 2 = medio, 3 = alto). En la Tabla 1–4 se ilustran los resultados de la estadística descriptiva (medias y DS).

Tabla 1–4: Estadística descriptiva de las variables

Variables	***code-centric***		**MBD**		**MDE**	
	Media	DS	Media	DS	Media	DS
HPU	4,541	0,973	3,48	1,19151	4,940	0,795
HPEU	3,98	0,757	3,545	0,927	4,87	0,832
HPC	2,0	0,40	2,12	0,326	1,54	0,508

Para darle respuesta a la **RQ1: *Utilidad percibida de los enfoques de desarrollo***, que está directamente relacionada con la prueba de hipótesis de HPU, ante todo se debe garantizar que la aplicación de este método estadístico tiene sentido. Primero se verifica la distribución normal de las VD con el contraste de Levene ($p > 0{,}05$ para todas las variables), que permite asumir la normalidad de la distribución de los datos. También se verifica la homogeneidad de covarianza entre los grupos con la prueba M de *Box* ($F = 1{,}137; p = 0{,}295$) y por último, se verifica que se cumpla la no violación del principio de esfericidad aplicando la prueba W de *Mauchly* ($W = 0{,}909; p = 0{,}387$). Como todos estos supuestos muestran un parámetro de significancia mayor que 0,05 permiten continuar los análisis.

Los resultados muestran que OOH4RIA produjo el PU superior, seguido de .NET y luego UML (ver Tabla 1–4) y además, se evidencia que la interacción Meth*Aplicación no es significativa ($F(8, 42) = 1{,}753; MSE = 1{,}311; p = 0{,}114$); lo que permite analizar los efectos principales de las dos VI (Aplicación y Meth) sobre la base de las medias sin peligro de necesitar la modificación de los resultados por la existencia de una interacción importante. El efecto principal de la aplicación no consiguió significado ($F(4, 21) =$

1,126; $MSE = 1{,}459; p = 0{,}371$), aunque por lo contrario, el efecto principal del método resultó significativo ($F(2, 42) = 19{,}411; MSE = 14{,}516; p \leqslant 0{,}001$). O sea, *las diferencias en PU están significativamente afectadas por el método usado, sin considerar la aplicación especial que están desarrollando.*

Teniendo en cuenta la trascendencia de la variable Meth, el último paso del análisis consiste en estudiar las diferencias por pares entre los métodos a través de una prueba T (*T-test*). En un esfuerzo de no incrementar el riesgo de un error de tipo-I, un ajuste de Bonferroni fue aplicado. Esto implica reducir el umbral de significado a 0,0167 ($p = 0{,}05/3 = 0{,}0167$). Con este ajuste, *las diferencias de PU entre OOH4RIA y los enfoques tanto .NET y UML son significativas, es decir, PU(MDE) es significativamente mejor que PU(MBD) y que PU(code-centric); mientras que la diferencia de PU entre .NET y OOH4RIA no lo son.*

En la **RQ2: *Facilidad de uso percibida de los enfoques de desarrollo*** se prueba la hipótesis de HPEU, con un procedimiento similar a HPU. Los resultados muestran que los supuestos de normalidad ($p > 0{,}05$ para todas las variables), homogeneidad de covarianza entre grupos ($F = 0{,}768; p = 0{,}78$) y principio de esfericidad ($W = 0{,}877; p = 0{,}269$) fueron cumplidos.

Los resultados revelan que OOH4RIA produjo mejores valores de PEU, seguidos de .NET y luego UML (ver medias y SD en Tabla 1–4). Los resultados también muestran que la iteracción Meth*Aplicación es significativa ($F(8, 42) = 2{,}801; MSE = 1{,}436; p = 0{,}014$); esto quiere decir que los resultados varían de acuerdo con el dominio de aplicación especial para el que la red social estaba siendo desarrollada.

Otro hallazgo es que mientras que el efecto principal de la aplicación no consiguió significado ($F(4, 21) = 1{,}033; MSE = 0{,}826; p = 0{,}414$), el efecto principal del método si es significativo ($F(2, 42) = 24{,}704; MSE = 25{,}329; p < 0{,}001$); *las diferencias en PU están significativamente afectadas por el método usado, sin considerar la aplicación especial que desarrollada.* Este resultado, sin embargo, debe ser interpretado con precaución, porque

el efecto del método es diferente en cada uno de los dominios de la aplicación. Además, aunque la variable principal fue encontrada significativa, la existencia de una interacción entre Meth*App hace cualquier análisis adicional de las comparaciones por parejas desaconsejable.

La tercera pregunta de investigación (**RQ3: *Compatibilidad percibida de los enfoques de desarrollo***) pone a prueba la hipótesis de HC, aplicando como se dijo anteriormente un *One-Way RM ANOVA*. Como premisa, se tiene que como todas las aplicaciones tienen el mismo juego de requisitos funcionales y pertenecen al mismo dominio, la compatibilidad del método no depende de la aplicación especial a desarrollar.

Los desarrolladores evaluaron UML como el método más compatible, seguido por un ligero margen de .NET y, finalmente OOH4RIA (ver medias y SD en Tabla 1–4). Esta compatibilidad es basada, exclusivamente, en la experiencia previa de los sujetos con los métodos de desarrollo empleados. El chequeo de la esfericidad W de *Mauchly* ($W = 0{,}471; p < 0{,}001$) fue significativo, por lo que es necesario evaluar las diferencias con una prueba más conservadora: F de *Greenhouse-Geisser*. Los resultados muestran que, efectivamente, el efecto principal del método es significativo ($F(1{,}308, 32{,}706) = 15{,}492; MSE = 3{,}704; p < 0{,}001$); es decir, *las diferencias en PC están significativamente afectadas por el método usado*. Una continuación del análisis en parejas usando T-*test* con un ajuste de Bonferroni muestra que *OOH4RIA es percibido como significativamente menos compatible que .NET y UML mientras que las diferencias entre .NET y UML no son relevantes.*

RQ4: *Correlación entre las variables conceptuales de la dimensión Satisfacción*. Como se ha mencionado anteriormente, ANOVA presenta a alto grado de robustez para escalas nominales y ordinales [61]. Sin embargo, la robustez de las correlaciones con respecto a las violaciones de la falta de normalidad de las distribuciones no es sencillo. Por esta razón, para estudiar la correlación entre las tres variables conceptuales del modelo teórico incluidas en este estudio, se lleva a cabo el análisis no paramétrico de correlación de *Spearman* previa transformación de todas las escalas, por razones de compatibilidad.

Los resultados indican que PC está correlacionado tanto con PEU como PU negativamente, esto quiere decir que, en este experimento, *los desarrolladores miraron el método de desarrollo con el que estaban menos familiarizados como más útil y fácil de usar*. Estos resultados son difíciles de explicar, y contradicen el modelo teórico y precisan de investigación adicional con una escala de medición de PC más compleja. Además se puede observar como PU y PEU están fuertemente correlacionados: *el método más útil es percibido como el más fácil de usar (y viceversa)*. Debido a que las dos variables son VD en este experimento, no se puede averiguar si hay causalidad, ni decir en qué dirección podría estar. Investigación más específica tiene que realizarse sobre estos aspectos de la teoría, y preferiblemente, tener los conceptos más elaborados que sean independiente entre sí.

Por otro lado, los resultados a la **RQ**5**: *Intención de adopción del método*** indican, que 20 de los 26 sujetos (para un 76,9 % de la muestra) decidieron usar el enfoque OOH4RIA para culminar sus proyectos.

Mientras que, para darle respuesta a la **RQ**6**: *Ventajas y desventajas percibidas de los métodos***, se utiliza una encuesta de opinión que permite comprender, aún más, los resultados previos obtenidos. A continuación se resumen las principales apreciaciones:

- *C#* con *Visual Studio* .NET 2010 como enfoque tradicional *code-centric*:
  - La desventaja principal de éste método es que requiere un esfuerzo de desarrollo alto (20 sujetos dieron respuestas que entraron en esa categoría) y tiene una mantenibilidad muy baja (5 sujetos).
  - Por el contrario se puede observar que el 50 % de los sujetos percibían el sentimiento del control cuando trabajaron con este enfoque como una ventaja importante, mientras que nadie lo catalogó como desventaja. Además, los sujetos informaron que .NET posee una curva de aprendizaje muy baja (5), más alta personalización (4) y compatibilidad cuando comparaban con su experiencia previa (3).
- UML con RSM para comprender los objetivos del sistema (pero sin generación de código) como enfoque MBD:
  - La principal desventaja informada de UML fue la falta de la confiabilidad percibida (significativa con 12 respuestas), atribuibles a algunos errores notados en el proceso de generación del código fuente por RSM.
  - Las ventajas del método variaban enormemente, encontrándose entre las más importantes: esfuerzo de desarrollo reducido (10), mejor mantenibilidad (6) y una curva de aprendizaje reducida (5). Sin embargo, estas ventajas no fueron apreciadas universalmente; esfuerzo de desarrollo y curva de aprendizaje, por ejemplo, eran apuntados como las desventajas por 5 y 3 sujetos, respectivamente.
- OOH4RIA como paradigma MDE:
  - Las desventajas principales informaron que los sujetos consideran que este método tiene una alta curva de aprendizaje (6), una compatibilidad baja con su experiencia previa (5), y falta de la confiabilidad (7), que pueden atribuirse a algunos defectos de la herramienta electrónica integrada en dispositivo de OOH4RIA que apareció durante la fase de desarrollo.
  - Por lo contrario la mayoría de los estudiantes (24 de 26) reconocieron un aumento importante en la velocidad de desarrollo, y al mismo tiempo una reducción sustancial

en el esfuerzo con las tareas repetitivas. Además, la mantenibilidad es percibida como una ventaja por 3 sujetos.

Amenazas para la validez

Siguiendo la clasificación de las amenazas para la validez de los experimentos (propuestas en [14]), a continuación se presentan los beneficios, así como limitaciones y condiciones en las que podría fallar el presente experimento.

Con el fin de reducir al mínimo las amenazas a la validez de las conclusiones, todos los análisis estadísticos han sido precedidos de pruebas que garanticen que las suposiciones del procedimiento no fueran violados. El alfa ha sido ajustada siguiendo un enfoque más conservador (Bonferroni), que también aporta validez a la conclusión. El diseño intra-sujetos contribuye a proteger los resultados contra la heterogeneidad de los sujetos, ya que todos realizaron las tareas con los tres métodos. Sin embargo, el número de observaciones por celda (5) era el mínimo necesario para poder aplicar este tipo de análisis. También, dado la duración de los tratamientos diferentes (cada sujeto trabajó en cada método durante 2 semanas), impertinencias aleatorias en el escenario experimental podrían haber afectado los datos. Se puede suponer, sin embargo, que tales impertinencias han afectado a todos los niveles del tratamiento equitativamente.

Para incrementar la validez interna del diseño (amenaza a la validez interna), los sujetos recibieron suficiente entrenamiento en los tres métodos. Todos los tratamientos fueron aplicados en un orden aleatorio (solamente un orden no fue cubierto en el experimento, MDE -*code-centric*- MBD, debido a que un grupo dejara el curso poco después de comenzar el experimento). Por consiguiente no había razón para la rivalidad compensatoria, el igualamiento o la desmoralización. Todas las aplicaciones y módulos sobre los que los sujetos trabajaron eran similares en complejidad. La falta de influencias de la aplicación especial sobre los resultados fue evaluada estadísticamente. Sin embargo, incluso cuando se balancea el diseño aleatoriamente en sentido inverso, los sujetos todavía aplicaron los

tres métodos de forma consecutiva; esto puede provocar efectos transferidos (p.e. los sujetos podrían beneficiarse en el transcurso del experimento de creciente familiaridad con el escenario del experimento, los ambientes de desarrollo, etc.). O, sin embargo, puede haber efectos de fatiga (p.e. a través de una pérdida del interés en el registro de los datos de experimento). Es posible que los usuarios imiten las prácticas (copiar) del método previamente usado (aprendizaje previo) cuando aplican el próximo método asignado. Para disminuir tales riesgos, se supervisa todo el proceso, manteniendo la información suministrada por los usuarios a un mínimo. También el proceso de recogida de los datos fue automatizado, para prevenir los errores de codificación.

Para minimizar el impacto de las amenazas para la validez del constructo, durante el experimento, tanto el modelo teórico como las hipótesis fueron ocultadas cuidadosamente de los sujetos. Ningún énfasis especial fue puesto sobre los beneficios y prejuicios de ninguno de los métodos hasta que el experimento fue terminado y las opiniones recogidas. Los métodos fueron aplicados junto con sus herramientas de respaldo, y las tecnologías usadas para evaluar cada enfoque de desarrollo son ampliamente empleadas en la práctica. Sin embargo, según criterio personal, no existe ningún instrumento estandarizado para medir los componentes del modelo teórico aplicado en este experimento. Aunque la confiabilidad de las escalas ha sido verificada para aumentar este tipo de validez; no obstante el número de artículos (*items*) de algunas de las escalas usadas deben ser incrementadas y validadas a través de investigación adicional para convertirlas en escalas más robustas. Particularmente, un estudio cuidadoso de todos los aspectos que pueden tener un rol importante en la utilidad percibida, la facilidad de uso y la compatibilidad (p.e. la calidad percibida del resultado del producto como parte de la escala utilidad percibida) deben realizarse para apoyar la inclusión de nuevos *items* subjetivos a cada escala.

Para terminar, las amenazas para la validez externa que están relacionadas con la generalización de los resultados. El tipo de las aplicaciones usadas para el experimento, lejos de ser un ejemplo hipotético, es una aplicación real, definida en base a requisitos de verdaderos cliente. Los sujetos son estudiantes post-graduados, muchos de los cuales están

trabajando como desarrolladores y por tanto constituyen verdaderos representantes de desarrolladores noveles. Por otro lado, dado que la muestra es relativamente pequeña (26 sujetos), y el hecho de que su selección no fue aleatoria, sino sobre la base de su participación en un estudio de máster, este experimento todavía presenta falta de representatividad de la muestra. Además, el hecho de que se han utilizado aplicaciones de tamaños similares y pertenecientes al mismo tipo (redes sociales), no se pueden generalizar los resultados a aplicaciones de tipos y tamaños diferentes sin necesidad de realizar más réplicas. Definitivamente, con respecto a esta clase de amenaza, aunque las herramientas y metodologías son ampliamente usadas entre los profesionales, no son las únicas alternativas de selección. El experimento tiene que ser reproducido para asegurarse de que el método usado y no la herramienta, es la que efectivamente fuera la causa de las diferencias observadas.

1.4.4. Experimento de mantenibilidad: OOH4RIA vs .NET

En un esfuerzo de incrementar las pruebas empíricas respecto al impacto de los enfoques MDE sobre la mantenibilidad, este trabajo presenta, siguiendo la misma plantilla de investigación que los experimentos anteriores (GQM [66] -Secciones 1.4.1, 1.4.2, 1.4.3-), un quasi-experimento que tiene como *objetivo* analizar OOH4RIA [48] y .NET (sobre la base de *C#* con *Visual Studio* .NET 2010) *con el propósito de* valorar las prácticas de mantenimiento MDE contra las prácticas centradas en el código, *con respeto* al rendimiento y satisfacción sobre la lógica de negocios de una aplicación Web 2.0 *desde el punto de vista* de desarrolladores noveles. El *contexto del estudio* son los estudiantes de post-grado (maestría) en el desarrollo de *software* Web en la Universidad de Alicante, donde tanto un proceso de desarrollo Web centrado en el código (basado en lenguaje .NET) como un proceso de desarrollo MDE (basado en la metodología OOH4RIA) son enseñados.

Las preguntas de investigación para dirigir este estudio han sido formuladas de la siguiente manera:

RQ1: ¿Cómo se comporta el rendimiento de las tareas de mantenibilidad (efectividad y eficiencia real) de la metodología OOH4RIA con respecto al rendimiento de realizar

tareas de mantenibilidad utilizando la metodología tradicional, centrado en el código, .NET?

RQ2: ¿Cómo se comporta la satisfacción de los sujetos al realizar tareas de mantenibilidad (efectividad, eficiencia, complejidad, certeza y estabilidad percibidas y así como la satisfacción en general) con la metodología de OOH4RIA respecto a la satisfacción de realizar las tareas de mantenibilidad utilizando la metodología tradicional, centrado en el código, .NET?

La correlación y fiabilidad de las escalas (aplicando el Alfa de *Cronbach*) han sido estadísticamente calculadas:

- Para la escala PSatisf (una escala semántica diferencial compuesto por 11 artículos), todos los *items* muestran una correlación superior a 0,3 para el constructo general, mientras que el global Alfa de Cronbach es de 0,89, dando prueba de una alta consistencia interna entre los elementos de PSatisf. Por tanto, es significativo para calcular la media, con el fin de utilizarlo como una clasificación global de PSatisf.
- Para la escala PU (compuesto por tres elementos: PEffv, PCert y PStab, ver Figura 1–3), todos los *items* mostraron una correlación superior a los 0,3 para el constructo general, el Alfa de Cronbach se mantuvo bastante baja ($\alpha = 0{,}676$), dando prueba de la baja consistencia interna entre los elementos.
- Para la escala PEU (compuesto por cuatro elementos: PSatisf, PCompl, Plearn y PEffc, ver Figura 1–3), el Alfa de Cronbach mostró un moderado grado de fiabilidad ($\alpha = 0{,}764$), con todos los elementos que muestran una correlación

  -total superior a los 0,3. Esta coherencia interna también hace posible calcular un valor global de PEU (PEU-Total) promediando los valores para los *items*.

- Las medidas subjetivas restantes constituyen escalas de un solo elemento.

Planificación del experimento

Teniendo en cuenta las preguntas de investigación presentadas se definen las siguientes VI: Meth (método, una variable categórica con dos niveles: *MDE y code-centric*) y App (aplicación, variable categórica con dos valores posibles: Petstore y Mediaplayer). Se puede caracterizar a Meth como un factor fijo (debido a que incluye los dos métodos a evaluar), mientras que App puede ser concebido como un factor aleatorio (pues estas aplicaciones son sólo dos ejemplos de la población de las aplicaciones Web).

Mientras que las VD se definen en base al Modelo de Evaluación de Métodos de [58] el cual sienta sus bases en pruebas de campos relacionados [57] y, posteriormente refinado en [45]. De acuerdo con este modelo (Modelo Teórico de Adopción de Métodos), el rendimiento se determina por las medidas (objetivas): Efectividad real (AEffv) y Eficiencia real (AEffc). Por otro lado, la satisfacción del desarrollador es determinada por la Utilidad percibida (PU) -Efectividad percibida (PEffv), Certeza percibida de los resultados (PCert), Estabilidad percibida (PStab)- y las Facilidad de uso percibida (PEU) -Eficiencia percibida (PEffc), Complejidad percibida (PCompl), Aprendizaje percibida (PLearn) y Satisfacción percibida (PSatisf)- (ver Figura 1–3).

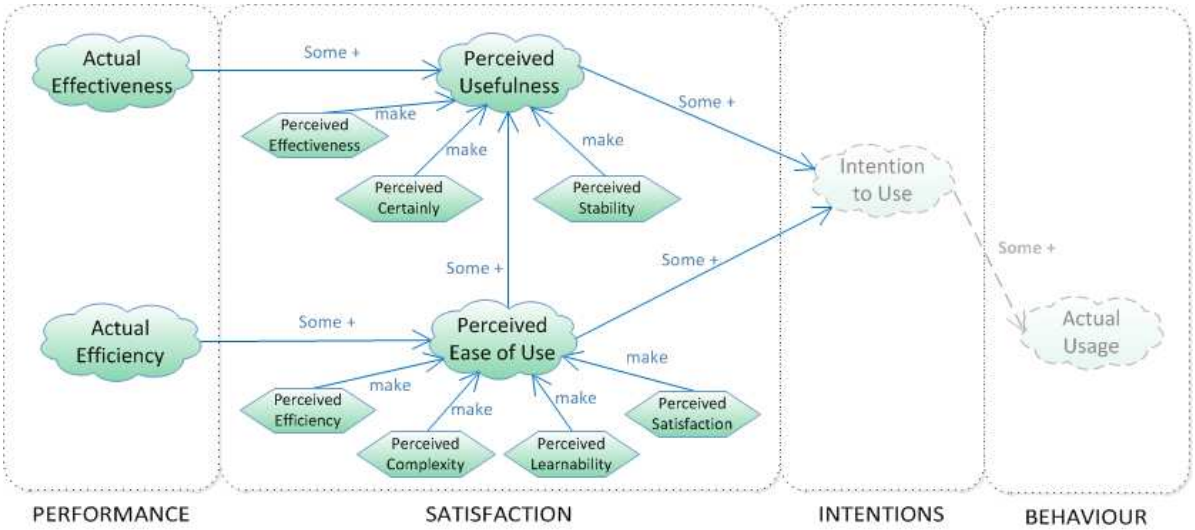


Figura 1–3: Componentes del Modelo Teórico de Adopción de Métodos, adaptado de [58].

En esta figura se representan las cuatro dimensiones del modelo -rendimiento, satisfacción, intención y comportamiento-, las variables (nubes), las medidas (hexágonos) y sus

influencias (flechas). Las variables y medidas que se han utilizado para probar las hipótesis de este experimento se representan en colores y líneas sólidas, mientras que las partes de este modelo que han sido excluidas de las pruebas experimentales se representan en colores transparentes y líneas discontinuas. La Figura (1–3) también refleja la dirección del efecto: positiva (*some* +) o negativa (*some* −).

Las variables definidas fueron medidas por separado para las tareas tanto correctivas como perfectivas y se utilizan para evaluar las siguientes hipótesis:

- **HAEffv** (Efectividad real): AEffv(MDE) > AEffv(*code-centric*). La efectividad de realizar tareas de mantenibilidad sobre modelos OOH4RIA y ambiente de generación es mejor con respecto a la mantenibilidad sobre código .NET.
- **HAEffc** (Eficiencia real): AEffc(MDE) > AEffc(*code-centric*). La eficiencia al realizar tareas de mantenibilidad sobre modelos OOH4RIA y ambiente de generación es mejor con respecto a la mantenibilidad sobre código .NET.
- **HPEffc** (Eficiencia percibida): PEffc(MDE) > PEffc(*code-centric*). Al realizar tareas de mantenibilidad sobre modelos OOH4RIA y ambiente de generación hacen a los desarrolladores sentirse más eficaces respecto a realizar la mantenibilidad directamente sobre código .NET.
- **HPCompl** (Complejidad percibida): PCompl(MDE) > PCompl(*code-centric*). Llevar a cabo tareas de mantenibilidad con OOH4RIA es percibido por los sujetos como significativamente más complicado que realizar estas tareas con .NET.
- **HPLearn** (Aprendizaje percibida): PLearn(MDE) > PLearn(*code-centric*). Aprender a realizar las tareas de mantenibilidad con OOH4RIA es percibido por los sujetos como significativamente más difícil que aprender a hacerlos directamente sobre código .NET.
- **HPSatisf** (Satisfacción percibida): PSatisf(MDE) > PSatisf(*code-centric*). Usando OOH4RIA para realizar las tareas de mantenibilidad es percibido por los sujetos como significativamente más fácil que realizarlos directamente sobre código .NET.

- **HPEU** (Facilidad de uso percibida): PEU(MDE) > PEU(*code-centric*). Usando OOH4RIA para realizar las tareas de mantenibilidad es percibido por los sujetos como significativamente más fácil de usar que realizarlos directamente sobre código .NET.
- **HPEffv** (Efectividad percibida): PEffv(MDE) > PEffv(*code-centric*). Al realizar tareas de mantenibilidad sobre modelos OOH4RIA y ambiente de generación hacen a los desarrolladores sentirse más efectivos respecto a realizar las tareas de mantenibilidad directamente sobre código .NET.
- **HCert** (Certeza percibida): PCert(MDE) > PCert(*code-centric*). Llevar a cabo tareas de mantenibilidad con OOH4RIA hacen a los sujetos sentirse significativamente más seguros sobre los resultados, en comparación al realizarlos sobre código .NET..
- **HPStab** (Estabilidad percibida): PStab(MDE) > PStab(*code-centric*). Realizar las tareas de mantenibilidad con OOH4RIA hacen a los sujetos sentirse significativamente más seguro sobre la estabilidad del resultado de las modificaciones realizadas (p.e. no introducir nuevos errores) directamente sobre código .NET.
- **HPU** (Utilidad percibida): PU(MDE) > PU(*code-centric*). Usando OOH4RIA para realizar las tareas de mantenibilidad es percibido por los sujetos como significativamente más útil que realizarlos directamentesobre código .NET.

Análisis de los datos e interpretación de los resultados

Todos los análisis fueron realizados con el PASW (*Predictive Analytics SoftWare*) v18 [62].

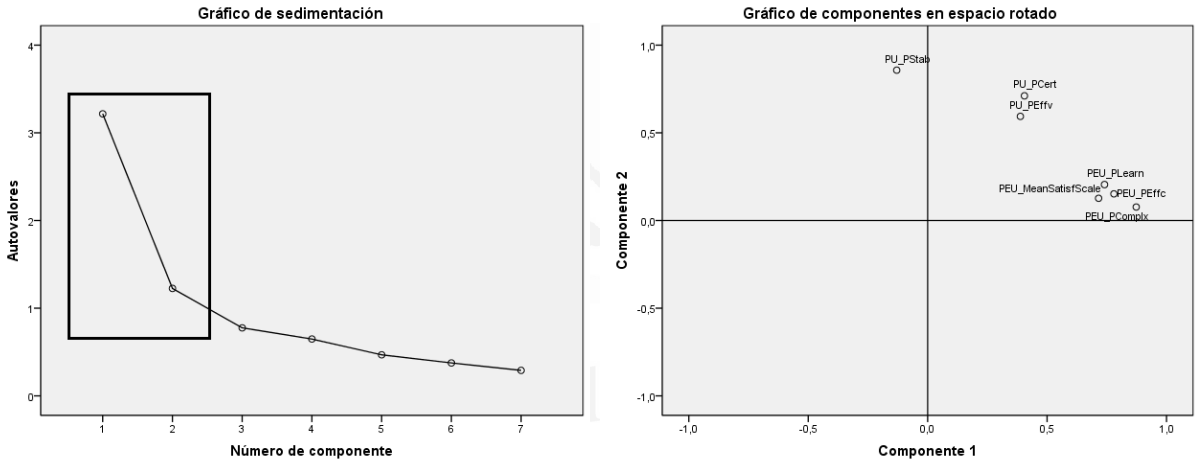
- Validación del modelo teórico

Con el fin de chequear si la agrupación de las variables (PEU y PU) de la dimensión satisfacción, sugeridas por el modelo teórico (ver Figura 1–3), satisface en realidad los datos del experimento, se realiza un Análisis de los Componentes Principales (PCA, *Principal Component Analysis*) [33].

El primer paso en este tipo de análisis es determinar el número óptimo de componentes perdiendo la menor cantidad de información posible, a partir del total de las siete medidas

iniciales que forman la dimensión. La Figura 1–4 (a) revela dos componentes principales -como se puede observar en la abrupta pendiente-, mientras que los componentes que poseen la caída de la pendiente más suave contribuyen poco a la solución.

Por otro lado, la matriz de componentes rotados, permite determinar cuales variables pertenecen a cada componente. El gráfico de dispersión -Figura 1–4 (b)-, muestra como las medidas asociadas a la variable PEU en el modelo teórico forman un grupo, mientras que todas las medidas relacionadas con PU forman un segundo grupo, por lo que se puede concluir que los datos se ajustan bien con las variables de agrupación del Modelo Teórico de Adopción de Métodos.



(a) Gráfico de sedimentación. (b) Gráfico de disperción matricial.

Figura 1–4: Análisis de los Componentes Principales. Gráficos.

El diagrama -Figura 1–4 (b)- también sugiere un análisis adicional de PU-Stab y PEU-Compl como las medidas más correlacionadas con los componentes extraídos PU y PEU, respectivamente.

- ***Estadística descriptiva***

El experimento se realiza a través de un diseño factorial intra-sujetos (ver Tabla 1–5), usando dos diferentes muestras de aplicaciones Web 2.0 para probar las hipótesis: Tienda de mascotas (*Petstore*, adaptación de un ejemplo usado en [79]) y Reproductor de música

(*Mediaplayer* adaptación de un ejemplo usado en [20, 23]). La muestra experimental es de 27 estudiantes.

Tabla 1–5: Diseño experimental: factorial e intra-sujetos

Sujetos	**Correcciones**	**Mejoras**	**Aplicación**
S1-S6	OOH4RIA	.NET	*Petstore*
S7-S13	.NET	OOH4RIA	*Petstore*
S14-S20	OOH4RIA	.NET	*Mediaplayer*
S20-S27	.NET	OOH4RIA	*Mediaplayer*

Las características de los sujetos que componen la muestra utilizada en este experimento, son similares a los sujetos del experimento de satisfacción e intención de adopción. Aunque no en cantidad, al pertenecer al mismo grupo del máster en el desarrollo de *software* Web en la Universidad de Alicante, los niveles de conocimientos con respecto a las diferentes tecnologías y métodos usados son similares a los descritos en la Sección 1.4.3, donde: el 81 % conocían UML, el 76 % de los estudiantes habían programado con la tecnología .NET durante sus cursos de grado, y a pesar de que ninguno tenía conocimientos prácticos previos de MDE, el 56 % de los sujetos eran consciente de la existencia del paradigma. Adicionalmente, como parte del programa de la maestría, y antes de realizar el experimento los sujetos reciben entrenamiento en los tres métodos: 30 horas en programación *C#* usando *Visual Studio* 2010 y 30 horas de entrenamiento en el modelado de UML y la herramienta OOH4RIA.

Para probar las hipótesis, se aplica el *test* estadístico *Two-Way ANOVA* con ($\alpha = 0{,}05$), que permite proveer información relevante de cómo interactúan las variables, o su combinación, así como el efecto que ellas tienen en las VD. En la Tabla 1–6 se aprecian los resultados de la estadística descriptiva (medias y DS) para cada una de las medidas.

Los resultados del análisis estadístico indican que ninguna de las hipótesis probadas detecta interacción significativa entre Meth*App, permitiendo probar cada variable (Meth y App) independientemente sin peligro de modificar los resultados por la existencia de una interacción importante. El hecho de que el número de observaciones por celda (combinación Meth*App) es tan igual como fuera posible contribuye a la robustez de estos análisis.

Tabla 1–6: Estadística descriptiva del experimento de mantenibilidad OOH4RIA vs .NET

Variables	**.NET**		**OOH4RIA**		***Petstore***		***Mediaplayer***	
	Mean	DS	Mean	DS	Mean	DS	Mean	DS
AEffv	3,33	1,11	4,22	0,89	3,92	1,02	3,64	1,16
AEffc	3,84	2,07	12,34	6,21	9,73	7,34	6,56	4,75
PU-PEffv	3,74	0,91	4,23	0,87	4,21	1,03	3,81	0,80
PU-PCert	3,33	0,78	3,74	0,81	3,73	0,72	3,36	0,87
PU-PStab	3,26	0,76	3,07	1,0	3,46	0,58	2,89	1,03
PU-Total	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
PEU-PEffc	4,57	1,16	5,37	1,21	5,00	1,33	4,96	1,20
PEU-PLearn	4,27	1,37	4,62	1,52	4,84	1,49	4,07	1,33
PEU-PCompl	3,96	1,40	4,88	1,42	4,68	1,40	4,18	1,52
PEU-PSatisf	4,63	0,96	4,56	0,94	4,69	1,00	4,51	0,88
PEU-Total	4,35	0,88	4,83	1,05	4,77	1,10	4,43	0,86
(NA: No aplicable)								

A continuación se presentan los análisis de las diferentes hipótesis. Es importante señalar que, como ANOVA presenta un alto grado de robustez para escalas nominales y ordinales [61], el tratamiento de todas las escalas asociadas con las diferentes variables subjetivas como medidas del intervalo para la refutación de las hipótesis derivadas de las preguntas de investigación (RQ1 y RQ2), no plantea una amenaza importante para la validez de las conclusiones del estudio. Esto también es coherente con la opinión general de las escalas de Likert que se definen por medio de suficientes, simétricas y equidistantes *items* tipo Likert (por lo general de 7 o 9-puntos) se aproxima a una medida de intervalo de nivel. Por otra parte, el tratamiento de la escala como intervalo puede ser beneficioso, porque de lo contrario alguna información valiosa (p.e. la "distancia" entre las opiniones) podría perderse [61].

RQ1: *Rendimiento de los métodos*

Para poner a prueba HAEffv, primeramente se calcula el coeficiente de Levene, el resultado ($F(3,50) = 0{,}919; p > 0{,}05$) permite refutar la hipótesis nula y asumir la homogeneidad de varianza y continuar con el análisis de ANOVA. Como la interacción Meth*App no es significativa, se analizan los principales efectos de las variables independientemente.

Para la variable Meth, la efectividad de los sujetos cuando usan OOH4RIA es significativamente más grande que la efectividad cuando utilizan .NET ($F(1,50) = 729; p < 0{,}05$). Además, el tamaño del efecto es muy alto, pues el parcial Eta cuadrado es de 0,999, lo que significa que el factor Meth por sí mismo representa el $99,9\,\%$ de la varianza general (efecto + error). Los resultados también muestran que la efectividad de los sujetos es ligeramente más grande con *Petstore* que con *Mediaplayer*, sin embargo esta diferencia no es importante ($F(1,50) = 72{,}25; p > 0{,}05$). En resumen, *las diferencias en la efectividad están significativamente afectadas por el método usado, independientemente de la aplicación particular que están desarrollando.*

A continuación, se pone a prueba la hipótesis HAEffc. El estadístico de Levene ($F(3,50) = 5{,}788; p < 0{,}05$) indica una violación de la suposición de homogeneidad de varianza. Para superar este problema, se aplica una transformación logarítmica de los datos: considerando la variable LAEffc ahora como el Ln(AEffc). Con la variable transformada, el coeficiente de Levene ($F(3,50) = 0{,}421; p > 0{,}05$) permite aceptar la hipótesis y continuar con el análisis.

Como la interacción Meth*App resulta no significativa, se pasa directamente a analizar los efectos principales de las dos variables (App y Meth) de forma independiente. Para Meth, el Ln de la eficiencia del sujeto cuando utiliza OOH4RIA es significativamente más grande que el Ln de la eficiencia del sujeto cuando usa el enfoque *code-centric* .NET ($F(1,50) = 1197{,}29; p < 0{,}05$); siendo una vez más, muy alto el tamaño del efecto. El parcial Eta cuadrado es 0,999, lo que significa que el factor Meth por sí mismo representa $99{,}9\,\%$ de la varianza general (efecto + error). Los resultados también indican que la eficiencia del sujeto no es significativamente diferente entre las aplicaciones: *Petstore* y *Mediaplayer* ($F(1,50) = 141{,}42; p > 0{,}05$). Esto significa que, *las diferencias en la eficiencia están significativamente afectadas por el método usado, sin considerar la aplicación especial que desarrollan.*

Tal como se presenta en la Figura 1–3, para darle respuesta a la **RQ2: *Satisfacción de los sujetos con el método de desarrollo***, se deben analizar los dos componentes

(variables) que conforman la dimensión satisfacción: PU y PEU. Previo a los análisis, se han revertido algunas de las preguntas del cuestionario a fin de que los rangos más altos correspondieran siempre a sentimientos más positivos (p.e. más fácil de aprender, menos compleja, más eficiente, y así sucesivamente). A continuación, se presentan los análisis de datos.

- *Utilidad percibida*

Para probar la hipótesis HPEffv, primero se calcula el estadístico de Levene para comprobar si las varianzas poblacionales son iguales. El resultado ($F(3,41) = 0{,}66; p > 0{,}05$), permite asumir la homogeneidad de varianza. Como la interacción Meth*App no es significativa, se analizan los principales efectos de las variables independientemente. Para la variable Meth, la efectividad percibida de los sujetos cuando se utiliza. NET es ligeramente menor que la efectividad percibida cuando realizan las tareas de mantenibilidad sobre OOH4RIA. Sin embargo, esta diferencia no es significativa: ($F(1,41) = 14{,}74; p > 0{,}05$). Además, los resultados revelan que los sujetos perciben la efectividad como ligeramente mayor con *Petstore* que con *Mediaplayer*, aunque esta diferencia, de nuevo, no es significativa: ($F(1,41) = 9{,}62; p > 0{,}05$). *Las diferencias en la efectividad percibida no están afectadas significativamente ni por el método usado ni por la aplicación implementada.*

Respecto a la hipótesis HPCert, el coeficiente de Levene ($F(3,50) = 3{,}025; p < 0{,}05$) indica una violación del supuesto de homogeneidad. Con el fin de superar este problema, se aplica una transformación logarítmica de los datos: considerando ahora la variable LPCert como Ln(PCert). Una vez más el resultado de la estadística de Levene ($F(3,50) = 2{,}970; p < 0{,}05$) advierte una violación de la homogeneidad de la varianza del error. Por esta razón, se aplica otra transformación logarítmica, y ahora LPCert es igual a $LnGamma$(PCert). Con la variable transformada, el coeficiente del Levene ($F(3,50) = 2{,}078, p = 0{,}115$) permite aceptar de homogeneidad de varianza y continuar con el análisis.

Para la variable Meth, el *LnGamma* de la certeza percibida por el sujeto con respecto a la corrección de las modificaciones propuestas utilizando OOH4RIA es ligeramente más alto que el *LnGamma* de la certeza percibida cuando usan .NET ($F(1,50) = 12{,}548; p > 0{,}05; \beta = 0{,}781$), pero el efecto principal del método no es significativo. Además, el *LnGamma* de la certeza percibida por los sujetos es ligeramente superior con *Petstore* que con *Mediaplayer* ($F(1,50) = 8{,}159; p > 0{,}05; \beta = 0{,}823$), aunque esta diferencia, otra vez, no es significativa. Es decir que, *las diferencias en la certeza percibida no es afectado significativamente ni por el método de desarrollo ni la aplicación especial desarrollada.*

La prueba de hipótesis de HPStab comienza verificando la distribución normal de las variables dependientes con el contraste de Levene. El resultado ($F(3,50) = 2{,}455; p > 0{,}05$) permite refutar la hipótesis nula y asumir la homogeneidad de la discrepancia de error. Como ha sido mencionado con anterioridad la interacción Met*App no es significativa, por lo que se procede a analizar los efectos de las variables de forma independiente.

Para Meth, los sujetos perciben la estabilidad de las modificaciones propuestas con .NET como ligeramente superior que la estabilidad percibida cuando usan OOH4RIA ($F(1,50) = 37{,}099; p > 0{,}05$), aunque no significativamente. Los resultados también revelan que la estabilidad percibida por los sujetos es significativamente mayor con *Petstore* que con *Mediaplayer* ($F(1,50) = 354{,}124; p < 0{,}05$). El valor alcanzado por el parcial Eta cuadrado (0,997) asegura que el tamaño del efecto de App es muy alto, lo que significa que este factor por sí mismo es capaz de explicar el 99,7 % del total de la varianza (efecto + error). *Las diferencias en la estabilidad percibida son afectadas significativamente por la aplicación especial que se está desarrollando, sin considerar el método usado.*

- *Facilidad de uso percibida*

Para la hipótesis HPEffc, el resultado del estadístico de Levene ($F(3,43) = 1{,}931; p > 0{,}05$) permite asumir la homogeneidad de varianza. Como la interacción Meth*App no es significativa ($F = 0{,}007; p > 0{,}05$), se puede proceder a revisar los efectos principales

de las dos variables (App y Meth) independientemente sin modificar los resultados por la existencia de una interacción importante.

Para la variable Meth, los sujetos perciben la eficiencia cuando se utiliza OOH4RIA como significativamente superior a la eficiencia percibida al usar .NET ($F = 770{,}438; p < 0{,}05$). El tamaño del efecto es muy alto donde el parcial Eta cuadrado es de 0,999, lo que significa que el factor Meth por sí mismo representa el $99,9\,\%$ de la varianza general (efecto + error). Además, la eficiencia percibida por los sujetos es ligeramente mayor con *Petstore* que con *Mediaplayer* ($F = 0{,}211; p > 0{,}05$), aunque no significativamente. Esto se puede explicar pues *las diferencias en la eficiencia percibida están significativamente afectadas por el método usado, sin considerar la aplicación especial desarrollada.*

La prueba de hipótesis de HPLearn, al igual que las pruebas de hipótesis anteriores, comienza con el cálculo del coeficiente de Levene, el resultado ($F(3, 48) = 0{,}123; p > 0{,}05$) permite asumir la homogeneidad de la discrepancia de error. Luego, se procede a analizar las variables Meth y App de forma independiente, debido a la ausencia de interacción significativa entre estas variables, basados en los resultados de las medias (ver Tabla 1–6). En cuanto a la variable Meth, la percepción de los sujetos de la facilidad de aprendizaje utilizando el método OOH4RIA es ligeramente mejor que la facilidad de aprendizaje percibido con .NET, pero esta diferencia no es significativa ($F(1, 48) = 0{,}973; p > 0{,}05$). Los resultados también muestran que la capacidad de aprendizaje de los sujetos es mayor con *Petstore* que con *Mediaplayer*, aunque no de forma significativa: ($F(1, 48) = 3{,}876; p > 0{,}05$). Es decir que, *las diferencias en el aprendizaje percibido no se afecta significativamente ni por la metodología ni por la aplicación desarrollada.*

En cuanto a la hipótesis HPCompl, otra vez se calcula el coeficiente de Levene para verificar si la discrepancia del error es constante. El resultado ($F(3, 49) = 1{,}358; p > 0{,}05$) permite que se refute la hipótesis nula y por lo tanto asumir la homogeneidad de la varianza del error. Para la variable de Meth, la complejidad percibida del individuo al realizar tareas de mantenibilidad con OOH4RIA es inferior (más sentimientos positivos) que la aparente complejidad de llevar a cabo las tareas de mantenibilidad con .NET, aunque no de forma

significativa: ($F(1,49) = 9{,}582; p > 0{,}05$). Además, los resultados también manifiestan que la aparente complejidad del sujeto es ligeramente inferior con *Petstore* que con *Mediaplayer* ($F(1,49) = 2{,}946; p > 0{,}05$) aunque, otra vez, no significativamente. *Las diferencias en la complejidad percibida cuando los sujetos llevan las tareas de mantenibilidad no se afectan significativamente ni por la metodología ni por la aplicación.*

Para probar la hipótesis de HSatisf (relacionado a la existencia de las diferencias significativas de la satisfacción percibida por el sujeto cuando realizan las tareas de mantenibilidad con los dos método) y garantizar que la aplicación del análisis estadístico tiene sentido, primero se verifica el contraste de Levene. El resultado ($F(3,50) = 4{,}058; p < 0{,}05$) muestra una violación de la homogeneidad de la varianza. Para superar este problema, se aplica una transformación logarítmica de los datos: considerando ahora la variable LPSatisf como el Ln(PSatisf). Con la variable transformada, el coeficiente de Levene $F(3,50) = 2{,}189; p > 0{,}05$, permite aceptar la suposición y continuar con el análisis.

Al analizar los efectos de las variables independientemente, para Meth el Ln de la satisfacción percibida por los sujetos con OOH4RIA es ligeramente inferior al Ln de la satisfacción percibida con .NET, aunque no de forma significativa: ($F(1,50) = 0{,}783; p > 0{,}05$). Mientras que el Ln de la satisfacción percibida por los sujetos es ligeramente superior con *Petstore* que con *Mediaplayer* ($F(1,50) = 3{,}831; p > 0{,}05$), aunque, otra vez, no significativamente. Esto significa que, *las diferencias en la satisfacción percibida no se afecta significativamente ni por la metodología ni por la aplicación.*

Por último, para probar la hipótesis HPEU-Total, el estadístico de Levene ($F(3,50) = 3{,}26; p = 0{,}029$) revela una violación de la presunción. Con el fin de superar este problema, se ha aplicado una transformación logarítmica de los datos: se considera la variable LPEU-Total ahora como Ln(PEU-Total). Con la variable transformada, el coeficiente de Levene ($F(3,50) = 2{,}292; p > 0{,}05$) permite aceptar la suposición y continuar con el análisis. Dada la ausencia de efecto significativo entre la interacción de las variables (App y Meth), se pueden analizar las dos VI sin modificar los resultados por la existencia de una interacción importante.

Para la variable Meth, los sujetos perciben el *Ln* de la facilidad de uso percibido (global) cuando realizan las tareas de mantenibilidad con OOH4RIA como ligeramente superior al *Ln* de la facilidad de uso percibida con .NET, aunque no de forma significativa: ($F = 25{,}419; p > 0{,}05$). Los resultados también muestran que el *Ln* de la facilidad de uso percibida por los sujetos es ligeramente superior con *Petstore* que con *Mediaplayer*, pero el efecto principal de la aplicación, de nuevo, no fue significativo ($F = 10{,}062; p > 0{,}05$). *Las diferencias en la facilidad de uso global percibida del sujeto no se afecta significativamente ni por la metodología ni por la aplicación.*

Amenazas para la validez

La evaluación de la validez del presente experimento, sigue la clasificación propuesta en [14], delimitando las amenazas de los estudios empíricos en cuatro diferentes tipos: interna, externa, de constructo y de conclusión. Debido a las similares características de este experimento en cuanto a cantidad de la muestra (27 sujetos) y características de los sujetos que la componen con el experimento de satisfacción e intención de adopción (26 sujetos); muchas de las amenazas que limitan la generalización de estos resultados han sido descritas con anterioridad (Sección 1.4.3). A continuación se detallan particularidades que difieren las amenazas de este experimento:

Para minimizar las amenazas para la validez de conclusion en este experimento se ha realizado un análisis de sensibilidad que asegura una potencia de 0,7 para tamaños de efectos mayores de 0,34. Los principales supuestos de las pruebas estadísticas se han comprobado, y los datos se han transformados de ser necesario para cumplirlas. Aunque el examen de las escalas de Linkert de 5 y 7 puntos como variables de intervalos es una cuestión controversial, el uso del ANOVA reduce considerablemente estas amenazas para la validez de las conclusiones del estudio [61]. La fiabilidad de las medidas relativas a las escalas han sido probadas y las hipótesis para las escalas que no mostraron un grado suficiente de fiabilidad quedaron fuera del estudio. A pesar de ello, el hecho de que algunas variables se han medido a través de un único elemento en el cuestionario constituye una amenazas para la validez de la conclusión, que solo se pueden superar con la definición y

validación de escalas estándares para medir tales constructos; aunque este tipo de escalas de medida aún no están disponibles para los investigadores.

Los tratamientos se aplican de forma constante entre los sujetos. Sin embargo, el hecho de que los resultados objetivos del experimento se utilizan para clasificar a los estudiantes pueden haber introducido ruido en los resultados (p.e. estando algunos estudiantes nerviosos por su rendimiento). A pesar de esto, como todos los sujetos aplicaron los tratamientos (métodos), se puede suponer que dicho efecto, si está presente, se ven afectados por igual a todos los niveles del tratamiento. Asimismo, este hecho reduce el riesgo de una heterogeneidad aleatoria de los sujetos, ya que ambos tratamientos se aplicaron al mismo grupo de sujetos.

Amenazas a la validez interna, en este experimento todos los sujetos inscritos en la maestría tenían que participar en el experimento (existe un sesgo de selección más allá de aquellos inherente a los presente en los cuasi-experimentos). Los sujetos aplicaron los tratamientos con diferentes tipos de tareas, lo que disminuye el riesgo de la historia. El tiempo limitado de duración del experimento (dos horas) es lo suficientemente corto para evitar un efecto de aprendizaje, mientras que el hecho de que tenían que aplicar dos tratamientos que intervienen un conjunto completamente diferente de habilidades limita el efecto de maduración. Además, dos instructores supervisaron todo el proceso con el fin de minimizar el sesgo de interacción. Sin embargo, existe una amenaza de efecto que se ha tratado de controlar con la realización de la prueba piloto, que aporta la confianza para asumir que dos horas es tiempo suficiente para que los sujetos terminaran las tareas.

Otras amenazas que afectan la validez interna del experimento es el hecho de que los sujetos no están obligados a responder todas las preguntas del cuestionario y que a su vez, ellos eran los encargados de reportar el tiempo que les llevó terminar las tareas. Además, las tareas fueron corregidas manualmente por uno de los instructores. El riesgo de subjetividad es controlado para asegurarse que la aplicación manipulada por los sujetos después de que se realizaron las tareas de mantenibilidad funcionan como era esperado. El hecho de que se aplicaron los tratamientos en dos aplicaciones diferentes también representa una

amenaza para la validez interna que ha sido controlado mediante el uso de aplicaciones de similares complejidad, medidas en número de construcciones conceptuales (para el tratamiento OOH4RIA) y en líneas de código (para el tratamiento *Visual Studio* .NET). Esta última medida se prefiere generalmente sobre los puntos de función, debido a su mayor fiabilidad [89].

Relacionadas con las amenazas a la validez de constructo, en este sentido, el modelo teórico está bien definido. Un PCA se ha realizado para comprobar que los datos se ajustan bien a la agrupación de las variables del modelo teórico. Sin embargo, la fiabilidad de algunas de las medidas utilizadas no han sido probadas, y por tanto pueden haber introducido un sesgo de medición, incluso si han sido utilizadas ampliamente en la literatura. También, la condición de examen puede haber planteado tensión adicional sobre los sujetos (temor a la evaluación) o, por el contrario, puede haber causado que los sujetos se comporten más eficientes y efectivos que lo usual debido a que se están calificando. Además, se han utilizado aplicaciones de tamaños similares, todos ellos pertenecientes al mismo dominio (sesgo de mono-operación). Por eso no se pueden generalizar los resultados a las aplicaciones de tamaños o dominios diferentes.

Por otro lado, la hipótesis del experimento (es decir, una mayor mantenibilidad de entornos MDE) eran bastante fáciles de adivinar, por lo que los estudiantes pueden sentirse obligados a reportar menos tiempo al utilizar OOH4RIA. De todos modos, los observadores del experimento tuvieron especial cuidado de no revelar esta hipótesis a los alumnos. Además, el experimento se ve afectado de una generalización limitada a través del constructo: se ha comprobado un resultado positivo entre la mantenibilidad y OOH4RIA, pero es imposible asegurar que esto no obstaculice otros atributos de calidad o cualquier otra característica adicional.

Por último, las amenazas a la validez externa tienen que ver con la generalización de los resultados a la práctica industrial. Los sujetos son estudiantes post-graduados, muchos de ellos ya trabajan como desarrolladores y por tanto constituyen verdaderos representantes de desarrolladores noveles, pero la pequeña muestra utilizada y el hecho de que estos

sujetos son personas altamente motivadas pueden no ser representativos de la población de desarrolladores noveles. Además, las metodologías y lenguajes particulares seleccionados, a pesar de ser ampliamente utilizados en el desarrollo industrial, constituyen un entorno restringido. Las aplicaciones elegidas, aunque limitadas en tamaños, son características de la clase de aplicaciones que se están desarrollando en la industria, pero el alcance de las tareas de mantenibilidad tenía que ser limitado debido a las restricciones de tiempo, por lo que existe el riesgo de que diferentes resultados se puedan obtener si las aplicaciones son más grandes o diferentes tareas de mantenibilidad hubiesen sido utilizadas para realizar el experimento. Por lo tanto, este experimento debe ser repetido con diferentes lenguajes, herramientas, aplicaciones, tareas y enfoques MDE. Para este propósito, el paquete de replicación de este experimento se puede encontrar en [46].

1.5. Discusión de los resultados obtenidos de la experimentación

Los resultados de esta tesis aumentan el depósito de los datos empíricos comparando productividad, mantenibilidad, satisfacción e intención de adopción de métodos de las aproximaciones MDE con respecto a los enfoques tradicionales: basados en modelos y/o centrados en el código.

Por un lado, se propone el Modelo Teórico de Adopción de Métodos (Figura 1–3) adaptado de [58], inicialmente propuesto en [43] y refinado en [45], que puede ser usado como punto de partida para estudiar las posibilidades de adopción de los métodos de desarrollo existentes en la industria de *software*. Las dimensiones, variables y medidas de este modelo, así como sus influencias se han identificado sobre la base de evidencia de diferentes campos de investigación [15, 21, 54, 58, 73, 87].

Las evaluaciones empíricas (Secciones 1.4.1 1.4.2, 1.4.3 y 1.4.4) realizadas en el contexto académico avalan el modelo teórico al considerar positiva la influencia del rendimiento, con los métodos de desarrollo, en la satisfacción del desarrollador. Mientras que el estadístico PCA garantiza la acertada agrupación de las medidas asociadas a la dismensión satisfacción en dos variables (PU, utilidad percibida y PEU, facilidad de uso percibida).

Sin embargo, este análisis también detecta la necesidad de realizar estudios más profundos con las medidas PU-Stab (estabilidad) y PEU-Compl (complejidad) pues son las medidas más correlacionadas a las variables PU y PEU, respectivamente.

Por otro lado, se proveen algunas contribuciones al proceso de desarrollo y mantenimiento de aplicaciones Web desde el punto de vista de los adoptadores de métodos, que permiten comprobar algunos supuestos del enfoque MDE. Los principales resultados obtenidos y su discusión se relacionan según las reivindicaciones a las que hace referencia:

- Relacionadas con el incremento de la productividad:

  El uso de las prácticas de ingeniería MDE como OOH4RIA permiten duplicar la productividad del equipo de desarrollo con respecto al uso de enfoques MBD (con UML) e incrementar hasta un 5,75 veces aproximadamente respecto a las prácticas de codificación, (*code-centric*), como .NET.

- Relacionadas con la mejora de la mantenibilidad:
  - Experimento WebML vs PHP:

    El uso de los enfoques de ingeniería MDE como WebML mejoran el rendimiento de los desarrolladores cuando llevan a cabo actividades de mantenibilidad sobre la capa de presentación de una aplicación Web, permitiendo que las tareas de analizabilidad (hasta 5,18 veces), cambiabilidad correctiva (hasta 2,17 veces) y cambiabilidad perfectiva (hasta 3,17 veces) se realicen significativamente más rápido en comparación con el lenguaje tradicional PHP. Además, la precisión de los cambios en las tareas de analizabilidad y mantenibilidad correctivas también resultan significativas.
  - Experimento OOH4RIA vs .NET:

    Los métodos MDE como OOH4RIA mejoran significativamente el rendimiento de los sujetos cuando llevan a cabo tareas de mantenibilidad, incrementando la efectividad y la eficiencia real en 1,27 y 3,21 veces, respectivamente, que cuando usan un lenguaje centrado en el código como .NET.

- Relacionadas con el incremento de la satisfacción de los desarrolladores:
  - Durante el desarrollo de aplicaciones Web, (experimento de productividad, satisfacción e intención de adopción):

    El uso de enfoques MDE como OOH4RIA incrementa significativamente la satisfacción de desarrolladores Web noveles con respecto al desarrollo centrado en el código como *Visual Studio* .NET. Sin embargo, las actividades de modelado que no están acompañadas de un fuerte ambiente de generación de código (MBD, como el uso de UML en RSM) hacen disminuir la satisfacción por debajo de las prácticas de codificación (*code-centric*), .NET.

    Este hallazgo respalda las investigaciones de [18], quien expone que el uso de UML como lenguaje de modelado no tiene un impacto significativo en el tiempo necesario para realizar un cambio, pues también se debe contar el tiempo para poner al día la documentación de UML.
  - Al realizar tareas de mantenibilidad de aplicaciones Web, (experimento de mantenibilidad, WebML vs PHP y OOH4RIA vs .NET):

    La satisfacción, no generó resultados significativos para ninguno de los dos enfoques de desarrollos comprobados (MDE o centradas en el código). En el primer experimento de mantenibilidad, que compara WebML vs PHP, los sujetos muestran una ligera preferencia por realizar las tareas de mantenibilidad directamente en el código fuente; mientras que en el segundo experimento de mantenibilidad, que compara OOH4RIA vs .NET, la balanza de satisfacción se inclina ligeramente en favor de p OOH4RIA como enfoque MDE.

    Una coincidencia entre estos dos experimentos, es que los sujetos sienten que las modificaciones no tendrán efectos inesperados (estabilidad de la solución) cuando realizan las tareas de mantenibilidad con un enfoque centrados en el código como PHP y .NET.

    Estos resultados, pueden deberse, como se ha mencionado anteriormente, a que los sujetos eran principiantes desarrolladores con el uso de enfoques MDE, sin embargo

tenían un alto nivel de experiencia previa en estudios de grado con los enfoques *code-centric* como PHP e incluso en la industria con tecnologías como .NET. Todo señala que si los sujetos tuviesen mayor experiencia con la metodología MDE, la única evolución natural de las medidas habrían sido con diferencias significativas más grande en favor de WebML y OOH4RIA.

- Relacionadas con Intención de adopción de los desarrolladores:

Los métodos MDE como OOH4RIA muestran un gran potencial de adopción, donde el 76,9 % de los sujetos, luego de terminado el experimento deciden continuar el proyecto con esta aproximación de desarrollo, a pesar de ser el método menos compatible con las experiencias previas de los desarrolladores. Los resultados, también muestran que los desarrolladores de *software* noveles se sienten cómodos con el uso de modelos, lo consideran como el método más útil y que es probable que ellos los usen si los modelos se acompañan por un ambiente de desarrollo MDE.

1.6. Conclusiones parciales

Este capítulo presenta un conjunto de cuatro experimentos, así como los principales aportes y limitaciones. Estos experimentos incrementan el repositorio de evidencia empírica del enfoque MDE en el desarrollo de aplicaciones Web, en aras de aseverar/refutar los supuestos que este paradigma ofrece a la comunidad de desarrolladores. La relevancia de los experimentos realizados se justifica a partir de un mapeo sistemático de las publicaciones en el campo MDE y de los resultados tan dispares encontrados en la literatura tanto de ganancias como pérdidas en entornos académicos e industriales.

CONCLUSIONES Y TRABAJOS FUTUROS

Conclusiones

En esta tesis se verifican las hipótesis y los beneficios del enfoque MDE. Los resultados alcanzados son significativos en el contexto de los métodos y herramientas utilizados, y para el tipo, tamaño y complejidad de las aplicaciones desarrolladas. Sin embargo, cabe mencionar que aplicaciones con similares características son comunes en muchas organizaciones dado el auge que ha cobrado Internet.

Entre las principales contribuciones de esta tesis doctoral se destacan:

- Una instantánea, actualizada a fecha de diciembre del 2010, del estado de la investigación empírica en el campo de la calidad, productividad y satisfacción de los enfoques MDE respecto a otros paradigmas de desarrollo.
- Un Modelo Teórico de Adopción de Métodos que integra las variables sobre las que existe un mayor grado de consenso entre la comunidad investigadora acerca de su poder de predicción del grado de intención de adopción de una nueva metodología.
- Una familia de experimentos que ha permitido comprobar:
  - Un incremento significativo de la productividad, así como de la mantenibilidad (correctiva y perfectiva) de los desarrolladores cuando llevan a cabo tareas de desarrollo y mantenimiento de aplicaciones Web 2.0 con enfoques MDE, en comparación con el desarrollo *code-centric* o MBD.
  - Ausencia de impacto significativo en la satisfacción de los desarrolladores. Los resultados muestran que los distintos enfoques no impactaron de manera significativa en la satisfacción, aunque la balanza se inclina ligeramente en favor del uso de enfoques de desarrollo MDE como WebML y OOH4RIA. Este resultado es prometedor, ya

que la experiencia previa de los desarrolladores era más fuerte con enfoques centrados en el código como los bien conocidos PHP y .NET. Sin embargo el uso de enfoques MBD (como UML) hacen disminuir la satisfacción, incluso por debajo de las las prácticas de codificación *code-centric*.

- Un buen potencial de adopción de los enfoques MDE como OOH4RIA por parte de los desarrolladores noveles.

En las actuales condiciones, es necesario seguir realizando réplicas de los experimentos para separar el efecto de los métodos de sus ambientes de desarrollo acompañantes y poder generalizar los resultados a diferentes poblaciones, métodos, lenguajes, tipos y tamaños de aplicaciones. Las ventajas y desventajas percibidas de los métodos y su influencia en la decisión definitiva de adoptarlos no solo proveen información valiosa para refinar la base teórica del modelo, sino también dan algunas pistas respecto a los aspectos que habría que mejorar en ambientes de desarrollo como la herramienta OOH4RIA IDE.

Trabajos Futuros

Esta tesis no es el final de los esfuerzos en este área de investigación. Muchas actividades experimentales están actualmente en marcha y otras quedan como investigaciones complementarias. En general, el mapeo sistemático y los experimentos realizados destapan las siguiente líneas de investigación en el campo de la evaluación empírica de MDE en el contexto de la ingeniería Web:

- Definir un método que ayude a los investigadores de IS a demostrar empíricamente las cualidades (su impacto sobre calidad de producto y proceso) de las herramientas y métodos MDE (en especial aquéllos pertenecientes al ámbito de la Ingeniería Web) en comparación con el resto de alternativas.
- Realizar réplicas de los experimentos con muestras de profesionales del desarrollo de *software*, es decir, evaluaciones en entornos industriales.
- Mejora y evaluación del modelo teórico de adopción de métodos propuesto.

Respecto a la primera línea de investigación abierta, durante el desarrollo de esta tesis una de las dificultades principales ha sido la falta de instrumentos de medición y tipologías de tareas estándares para realizar las distintas comparativas. Este hecho ha obligado a definir dichas tareas y medidas, lo que impide cualquier tipo de comparación con respecto a estudios realizados por otros investigadores o paquetes experimentales.

En este sentido, es posible basarse en la experiencia y los elementos de los paquetes experimentales, elaborados como parte de los estudios empíricos realizados, para la sistematización y estandarización de dichos materiales. El desarrollo de instrumentos de medición y tareas estándares pueden facilitar sin duda la realización de estudios empíricos por parte de los investigadores y desarrolladores de métodos de desarrollo Web basados en el paradigma MDE.

La realización de estos estudios es fundamental de cara a ser capaces de responder a la pregunta ¿Qué enfoque MDE debería seleccionar en función de mi contexto de uso y mis necesidades de calidad?

Además, es necesario continuar trabajando para extender estos estudios a otras características de calidad del producto y del proceso más allá de la mantenibilidad, productividad y satisfacción. Los resultados, en su mayor parte anecdóticos y poco contrastados del mapeo sistemático realizado en [44], sobre la evidencia empírica respecto al impacto del MDE en características como la eficiencia, funcionalidad, fiabilidad, usabilidad y portabilidad, demuestran la necesidad de que la comunidad de ingeniería realice otros estudios empíricos que provean evidencia sobre las ventajas/desventajas que este paradigma ofrece, sobre estas características, al desarrollo de *software*.

La segunda línea de investigación, está estrechamente relacionada a la primera. Dadas las amenazas para la validez de la generalización de los resultados obtenidos de los experimentos realizados, es necesaria su reproducción utilizando muestras representativas de la comunidad de desarrolladores Web. Por otra parte, es necesario realizar réplicas con

aplicaciones de tipos y tamaños diferentes, así como con lenguajes y enfoques MDE diferentes, en aras de garantizar que es el método usado y no otras variables la causante las diferencias observadas.

Por último y no menos importante (tercera línea de investigación), por lo que respecta al modelo teórico de adopción de métodos, es necesario acompañarlo de un instrumento de medición válido y fiable. Este instrumento es el primer paso para validar el modelo teórico y evaluar su poder predictivo.

REFERENCIAS BIBLIOGRÁFICAS

[1] *Aclarando Conceptos: Eficiencia Vs. Eficacia, Aptitud Vs. Actitud Y Precisión Vs. Exactitud.*, 2010.

[2] S Abrahão, E Mendes, J Gomez, and E Insfran, *A model-driven measurement procedure for sizing Web applications: design, automation and validation. Model*, Driven Engineering Languages and Systems (2007), 467–481.

[3] I Ajzen and M Fishbein, *The Influence of Attitudes on Behavior*, The handbook of attitudes **173** (2005), 173–221.

[4] S L Baigorria and G Montejano, *MÉTRICAS APLICADAS A LA PROGRAMACIÓN ORIENTADA A ASPECTOS*, Workshop de Investigadores en Ciencias de la Computación WICC-2006, 2006.

[5] K Balasubramanian, A Gokhale, G Karsai, J Sztipanovits, and S Neema, *Developing applications using model-driven design environments*, Computer **39** (2006), no. 2, 33–40.

[6] H Barki and J Hartwick, *Measuring user participation, user involvement, and user attitude*, MIS Quarterly **18** (1994), no. 1, 59–82.

[7] K H Bennett and V T Rajlich, *Software maintenance and evolution: a roadmap*, Proceedings of the conference on The future of Software engineering, ACM, 2000, p. 87.

[8] G. Booch, I. Jacobson, and J. Rumbaugh, *The unified software development process*, Addison Wesley, 1999.

[9] S Ceri, P Fraternali, and A Bongio, *Web Modeling Language (WebML): a modeling language for designing Web sites*, Computer Networks **33** (2000), no. 1-6, 137–157.

[10] S Ceri, P Fraternali, and M Matera, *Conceptual modeling of data-intensive Web applications*, Internet Computing, IEEE **6** (2002), no. 4, 20–30.

[11] N Chapin, J E Hale, K M Khan, J F Ramil, and W G Tan, *Types of software evolution and software maintenance*, Journal of Software Maintenance and Evolution: Research and Practice **13** (2001), no. 1, 3–30.

[12] CMU/SEI, *CMMI Product Development Team, CMMI for Development verion 1.2*, 2006.

[13] N. Condori-Fernández and O. Pastor, *Re-evaluando la intención de uso de un procedimiento de medición basado en cosmic-ffp*, Revista Española de Innovación, Calidad e Ingeniería del Software **2** (2006), no. 3.

[14] T D Cook, D T Campbell, and A Day, *Quasi-experimentation: Design & analysis issues for field settings*, Houghton Mifflin Boston, 1979.

[15] F D Davis, *Perceived usefulness, perceived ease of use, and user acceptance of information technology*, MIS quarterly **13** (1989), no. 3, 319–340.

[16] O Diaz and F M Villoria, *Generating blogs out of product catalogues: An MDE approach*, The Journal of Systems and Software **83** (2010), no. 10, 1970–1982.

[17] T Dyba, B A Kitchenham, and M Jorgensen, *Evidence-based software engineering for practitioners*, Software, IEEE **22** (2005), no. 1, 58–65.

[18] W J Dzidek, E Arisholm, and L C A Briand, *Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance*, IEEE Transactions on Software Engineering **34** (2008), no. 3, 407–432.

[19] M Fowler, *UML distilled: a brief guide to the standard object modeling language*, 3rd editio ed., Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2004.

[20] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, *Cans: composable, adaptive network services infrastructure*, Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems-Volume 3, USENIX Association, 2001, pp. 12–12.

[21] H Fujita and I Zualkernan, *Evaluating Software Development Methodologies based on their Practices and Promises*, New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Seventh Somet_08 **182** (2008), 14.

[22] R L Glass, *The Realities of Software Technology Payoffs*, Communications of the ACM **42** (1999), no. 2, 74–79.

[23] R H Glitho, F Khendek, and A De Marco, *Creating value added services in internet telephony: an overview and a case study on a high-level service creation environment*, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **33** (2003), no. 4, 446–457.

[24] T Goldschmidt, R Reussner, and J. Winzen, *A Case Study Evaluation of Maintainability and Performance of Persistency Techniques*, Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), May 10-18 (Leipzig (Germany)) (W Schafer, M B Dwyer, , and V Gruhn, eds.), 2008, pp. 401–410.

[25] M L Guerini, *Revisión de resultados experimentales en técnicas de prueba y de educción de conocimientos*, Tesis de magister en ingeniería de software, Universidad Politécnica de Madrid, España, 2007.

[26] J Hutchinson, M Rouncefield, and J Whittle, *Model-driven engineering practices in industry*, Proceeding of the 33rd International Conference on Software Engineering (ICSE), IEEE, 2011, pp. 633–642.

[27] J Hutchinson, J Whittle, M Rouncefield, and S Kristoffersen, *Empirical assessment of MDE in industry*, Proceeding of the 33rd International Conference on Software Engineering, ACM, ACM, 2011, pp. 471–480.

[28] IEEE Std. 1219, *IEEE Standard for Software Maintance*, 1998.

[29] ISO/IEC, *Iec 25000: Software engineering-software product quality requirements and evaluation (square)-guide to square*, International Organization for Standardization, Geneva, Switzerland (2005).

[30] ISO/IEC 14598 series, *Software Engineering - Product evaluation.(Part 1: General overview, Part 2: Planning and management, Part 3: Process for developers, Part 4: Process for acquirers, Part 5: Process for evaluators, Part 6: Evaluation module)*, 2001.

[31] ISO/IEC 25010, *Systems and software engineering - Systems and Software Quality Requirements and Evaluation(SQuaRE)- System and software quality models.*, 2011.

[32] ISO/IEC 9126-1, *Software engineering - Product quality - Part 1: Quality model*, 2001.

[33] I. Jolliffe, *Principal component analysis*, Wiley Online Library, 2005.

[34] A. Juristo and N. Moreno, *Basics of Software Engineering Experimentation*, Kluwer Academic Publisher, 2001.

[35] L Kish, *Some statisticals problems in research design*, American Sociological Review **24** (1959), 328–338.

[36] B Kitchenham, D Budgen, P Brereton, M Turner, S Charters, and S Linkman, *Large-scale software engineering questions-expert opinion or empirical evidence?*, Software, IET **1** (2007), no. 5, 161–171.

[37] B Kitchenham and S Charters, *Guidelines for performing systematic literature reviews in software engineering*, Technical report ebse-2007-01, School of Computer Science and Mathematics, Keele University, 2007.

[38] B Kitchenham, S Linkman, and D Law, *DESMET: a methodology for evaluating software engineering methods and tools*, Computing & Control Engineering Journal **8** (2002), no. 3, 120–126.

[39] C Lange and M Chaudron, *Interactive Views to Improve the Comprehension of UML Models - An Experimental Validation*, Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC '07), June 26-29. (Banff, Alberta (Canada)) (IEEE Computer Society Press, ed.), 2007, pp. pp. 221–230,.

[40] M Linaje, J Preciado, and F Sánchez-Figueroa, *A method for model based design of rich internet application interactive user interfaces*, Web Engineering (2007), 226–241.

[41] E.D. López, M González, M López, and E.L. Iduñate, *Proceso de Desarrollo de Software Mediante Herramientas MDA*, Revista Iberoamericana de Sistemas, Cibernética e Informática **3** (2006), no. 2, 6–10.

[42] A. MacDonald, D. Russell, and B. Atchison, *Model-driven development within a legacy system: an industry experience report*, Software Engineering Conference, 2005.

Proceedings. 2005 Australian, IEEE, 2005, pp. 14–22.

[43] Y Martínez, C Cachero, M Matera, S Abrahao, and S Luján, *Impact of MDE Approaches on the Maintainability of Web Applications: An Experimental Evaluation*, Conceptual Modeling-Er 2011: 30th International Conference on Conceptual Modeling, Brussels, Belgium, October 31-November 3, 2011. Proceedings, vol. 6998, Springer-Verlag New York Inc, Springer-Verlag New York Inc, 2011, pp. 233–246.

[44] Y Martínez, C Cachero, and S Meliá, *Evidencia empírica sobre mejoras en productividad y calidad mediante el uso de aproximaciones MDD: un mapeo sistemtico de la literatura.*, 2011, pp. 6–28.

[45] Martínez, Y and Cachero, C and Meliá, S, *MDD vs. traditional software development: A practitioner's subjective perspective*, Information and Software Technology, http://dx.doi.org/10.1016/j.infsoft.2012.07.004 (2012).

[46] Y Martínez, C Cachero, and S Meliá, *Experiment replication package*, http://www.dlsi.ua.es/ ccachero/labPackages/Maintainability2012.rar, 2012.

[47] J W Mauchly, *Significance test for sphericity of a normal n-variate distribution*, The Annals of Mathematical Statistics **11** (1940), no. 2, 204–209.

[48] S Meliá, J Gómez, S Pérez, and O Díaz, *A model-driven development for GWT-based Rich Internet Applications with OOH4RIA*, Web Engineering, 2008. ICWE'08. Eighth International Conference on, IEEE, 2008, pp. 13–23.

[49] S Meliá, J J Martinez, Á Pérez, and J Gómez, *OOH4RIA Tool: Una Herramienta basada en el Desarrollo Dirigido por Modelos para las RIAs*, Jornadas de Ingeniería del Software y Base de Datos, JISBD 2009, 2009.

[50] N Mellegå rd and M Staron, *Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment*, Product-Focused Software Process Improvement (2010), 336–350.

[51] E Mendes, N Mosley, and S Counsell, *The Need for Web Engineering: An Introduction*, Web Engineering (2006), 1–27.

[52] E M Méndez, M A Pérez, A.C. Grimán, L E Mendoza, and L Sistemas, *¿ cómo se relacionan la calidad sistémica y la productividad en el proceso de desarrollo de software?*, Iberoamericana de Sistemas, Cibernética e Informática **2** (2005), no. 2, 1–8.

[53] MODELWARE D5.3-1 Industrial ROI, *Assessment and Feedback-Master Document. Revision 2.2*, 2006.

[54] P Mohagheghi, *An Approach for Empirical Evaluation of Model-Driven Engineering in Multiple Dimensions*, C2M:EEMDD 2010 workshop- from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, CEA LIST Publication, 2010, pp. 6–17.

[55] P Mohagheghi and R Conradi, *An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes*, Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on, IEEE, 2004, pp. 7–16.

[56] P Mohagheghi and V Dehlen, *Where is the proof? - A review of experiences from applying MDE in industry*, European Conference on Model Driven Architecture–Foundations and Applications (ECMDA 2008), Springer, 2008, pp. 432–443.

[57] D L Moody, *Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models (PhD Thesis)*, Melbourne, Australia: Department Of Information Systems, University of Melbourne (2001).

[58] D L Moody, *The method evaluation model: A theoretical model for validating information systems design methods*, 11th European Conference on Information Systems (ECIS 2003), Naples, Italy, Citeseer, 2003, p. 79.

[59] G C Moore and I Benbasat, *Development of an instrument to measure the perceptions of adopting an information technology innovation*, Information systems research **2** (1991), no. 3, 192–222.

[60] N Moreno, J R Romero, and A Vallecillo, *An overview of model-driven web engineering and the mda*, Web Engineering: Modelling and Implementing Web Applications (2008), 353–382.

[61] G Norman, *Likert scales, levels of measurement and the laws of statistics*, Advances in health sciences education **15** (2010), no. 5, 625–632.

[62] M.J. Norusis et al., *Pasw statistics 18 guide to data analysis*, Prentice Hall Press, 2010.

[63] A Noruzi, *Google Scholar: The new generation of citation indexes*, Libri **55** (2005), no. 4, 170–180.

[64] R L Oliver, *A conceptual model of service quality and service satisfaction: compatible goals, different concepts*, Advances in Services Marketing and Management **2** (1993), 65–85.

[65] Object Management Group OMG, *MDA Guide Version 1.0.1*, 2003.

[66] D E Perry, A A Porter, and L G Votta, *Empirical studies of software engineering: a roadmap*, Proceedings of the conference on The future of Software engineering, ACM, 2000, pp. 345–355.

[67] K Petersen, R Feldt, S Mujtaba, and M Mattsson, *Systematic mapping studies in software engineering*, 12th International Conference on Evaluation and Assessment in Software Engineering, 2008, pp. 71–80.

[68] L M Pickard, *Combining Empirical Results in Software Engineering*, Tech. report, Keele University, England, 2004.

[69] R Picó, *Análisis funcional de la gestión de almacén y producción en openbravo ERP y su aplicación docente.*, Proyecto fin de carrera, Universidad Politécnica de Valencia, España, 2010.

[70] J C Preciado, M Linaje, R Morales-Chaparro, F Sanchez-Figueroa, G Zhang, C Kroiß, and N Koch, *Designing rich internet applications combining uwe and rux-method*, Web Engineering, 2008. ICWE'08. Eighth International Conference on, IEEE, 2008, pp. 148–154.

[71] R Pressman, *Ingeniería del software. un enfoque práctico*, 5ta edición ed., McGraw-Hill Interamericana,, Madrid, España, 2002.

[72] Real Academia Española, *Dicccionario de la Lengua Española - Vigésima segunda edición*, 2001.

[73] C K Riemenschneider, B C Hardgrave, and F D Davis, *Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING **28** (2002), no. 12, 1135–1145.

[74] F Ruiz and M Polo, *Mantenimiento del Software*, Master en ingenierÍa del software, 2007, p. 109.

[75] B Selic and I.B.M.D. Engineer, *An overview of uml 2.0*, INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, vol. 25, 2003, pp. 755–756.

[76] Bran Selic, *Model-Driven Development : Its Essence and Opportunities*, Engineering (2006).

[77] M.A Sicilia, *Métricas de Mantenibilidad Orientadas al Producto*, 2009.

[78] C U Smith and L G Williams, *Performance solutions: a practical guide to creating responsive, scalable software*, vol. 1, Addison-Wesley Boston, MA;, 2001.

[79] SUN Microsystems, *Java Pet Store Sample Application, Blueprints Online*, 2010.

[80] The Middleware Company, *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis (White Paper)*, 2003.

[81] R L Thompson, C A Higgins, and J M Howell, *Personal computing: toward a conceptual model of utilization*, MIS quarterly **15** (1991), no. 1, 125–143.

[82] M Urbieta, G Rossi, J Ginzburg, and D Schwabe, *Designing the interface of rich internet applications*, Web Conference, 2007. LA-WEB 2007. Latin American, IEEE, 2007, pp. 144–153.

[83] A Vallecillo, N Koch, C Cachero, S Comai, P Fraternali, I Garrigós, J Gómez, G Kappel, A Knapp, M Matera, and Others, *MDWEnet: A practical approach to achieving interoperability of model-driven Web engineering methods*, Workshop Proc. of 7th Int. Conf. on Web Engineering (ICWE'07), Italy, Citeseer, 2007.

[84] F Valverde and O Pastor, *Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach*, Web Information Systems Engineering-WISE

2009 (2009), 131–144.

[85] G Vanderburg, *Real Software Engineering, video/mp4*, 2010.

[86] V Venkatesh and F D Davis, *A theoretical extension of the technology acceptance model: Four longitudinal field studies*, Management science **46** (2000), no. 2, 186–204.

[87] S Walderhaug, M Mikalsen, I Benc, and S Erlend, *Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project*, From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI. Workshop at European Conference on Model-Driven Architecture, 2008.

[88] R Wieringa, N Maiden, N Mead, and C Rolland, *Requirements engineering paper classification and evaluation criteria: a proposal and a discussion*, Requirements Engineering **11** (2006), no. 1, 102–107.

[89] C Wohlin, P Runeson, and M Höst, *Experimentation in software engineering: an introduction*, Springer Netherlands, 2000.

Parte II

COMPENDIO DE ARTÍCULOS

.

CAPÍTULO 2
EVIDENCIA EMPÍRICA SOBRE MEJORAS EN PRODUCTIVIDAD Y CALIDAD EN ENFOQUES MDD: UN MAPEO SISTEMÁTICO

El contenido de este capítulo corresponde con el siguiente artículo: Martínez, Y., Cachero, C. and Meliá, S. - Evidencia empírica sobre mejoras en productividad y calidad en enfoques MDD: un mapeo sistemático. Revista Española de Innovación, Calidad e Ingeniería del Software. Volumen 7*, No.* 2*, octubre,* 2011*. ISSN:* 1885 − 4486*.*

La Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS) publica artículos que cubran cualquier aspecto de la innovación, investigación o aplicación práctica de conocimientos técnicos, ingenieriles y científicos en la ingeniería y la calidad del *software*. La revista pretende canalizar las contribuciones relevantes en los campos antes mencionados que permitan mejorar los resultados, la productividad, la eficiencia o la calidad en los proyectos y en la gestión de *software* o que permitan incrementar el conocimiento desde una perspectiva práctica.

La revista REICIS (www.ati.es/reicis) está indizada en numerosas bases de datos internacionales como DOAJ, e-revistas, ICYT-CSIC, Dialnet, Redalyc, Latindex, Ulrich y *Google Scholar*, lo que garantiza el cumplimiento de todos sus criterios de calidad editorial. Asimismo está incluida en la CIRC (Clasificación Integrada de Revistas Científicas) del CSIC (Consejo Superior de Investigaciones Científicas) como revista de categoría C.

Evidencia empírica sobre mejoras en productividad y calidad en enfoques MDD: un mapeo sistemático

Yulkeidi Martínez

Universidad de Ciego de Ávila, Cuba

yulkeidi@gmail.com

Cristina Cachero, Santiago Meliá

Universidad de Alicante, España.

{ccachero, santi}@dlsi.ua.es

Resumen

Para el avance del desarrollo de software dirigido por modelos, es esencial proporcionar evidencias empíricas que corroboren o refuten las promesas de mejora asociadas a este paradigma desde su concepción. El objetivo de este trabajo es clasificar la evidencia empírica existente respecto de la mejora en productividad y calidad de las aplicaciones. Para ello hemos aplicado el proceso de mapeo sistemático, un tipo de estudio secundario diseñado específicamente para abordar este tipo de objetivos. Como resultado de este trabajo, hemos identificado asunciones que carecen a día de hoy de evidencia empírica. Por tanto, constituyen líneas de trabajo que se deben abordar para una mayor rigurosidad y consistencia de la disciplina. También hemos identificado áreas donde un análisis más exhaustivo podría ser de utilidad. El mapa resultante facilita la entrada de nuevos investigadores a este campo.

Palabras clave: Mapeo sistemático, ingeniería dirigida por modelos, calidad, productividad

Empiric evidence on productivity and quality improvements with MDD approaches: a systematic mapping

Abstract

In order to consolidate the progress in the development of the Model-Driven Development paradigm, it is essential to provide empirical evidence that either corroborates or refutes the promises of improvement that attached to this paradigm since its inception. The purpose of this paper is to classify the existing empirical evidence referred to improvements in productivity and quality of the applications. In order to achieve this goal, we have applied the systematic mapping process, a type of secondary study specifically devoted to carry out this kind of studies. As a result of this work, we identified assumptions based on today's lack of empirical evidence, and therefore lines of work are to be addressed to bring more rigor and consistency in discipline. We have also identified areas where further analysis could be useful. The resulting map facilitates the entry of new researchers to the field.

Key words: Systematic Mapping, Model Driven Development, Quality, Productivity

1. Introducción

El Desarrollo Dirigido por Modelos (MDD, del inglés Model Driven Development) es una aproximación al desarrollo de software basado en (a) la creación de modelos del sistema software a distintos niveles de abstracción y (b) su uso como base de un proceso de generación automática de código [1]. Entre las reivindicaciones de este paradigma de desarrollo se encuentran [2] [3] [4]:

1. Mayor simplicidad del proceso. El desarrollador se puede aislar de la complejidad tecnológica y centrarse en la estructura y comportamiento deseado de la aplicación.
2. Mejora de la productividad del proceso de desarrollo. El uso de modelos independientes de cómputo (CIM, *Computation-Independent Model*), modelos independientes de plataforma (PIM, *Platform-Independent Model*) y modelos de plataforma específica (PSM,

Platform-Specific Model) permiten especificar el sistema a distintos niveles de abstracción y favorecen de este modo el reuso. La definición de transformaciones modelo a modelo y modelo a código automatiza gran parte del proceso de codificación.

3. Mejora de la calidad externa de la aplicación resultante (Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad) [5].

Sin embargo, y a pesar de las numerosas llamadas de atención por parte de la comunidad de Ingeniería del Software acerca de la necesidad de acompañar este tipo de afirmaciones con evidencias empíricas [6] [7], la gran mayoría de aportaciones en el campo del MDD sigue tomando la forma de nuevas metodologías, técnicas y herramientas que, aunque viables, no llegan a demostrar de una manera fiable su utilidad y ventajas respecto a sus predecesoras.

Con el fin de ayudar a que la comunidad proporcione estas evidencias empíricas, el área de experimentación en Ingeniería del Software ha desarrollado guías exhaustivas que ayudan a los investigadores en el proceso de obtención de datos fiables acerca de las ventajas o desventajas de los distintos métodos, técnicas o herramientas empleadas en la construcción de sistemas software [8] [9]. En ausencia de estos datos, se sigue corriendo el peligro de sostener conclusiones erróneas [10], perjudicando de esta manera tanto la toma de decisiones en el ámbito empresarial [11] como la propia imagen de la disciplina, tal y como ya ha sucedido en el pasado [12].

Aún más importante, es necesario proporcionar un acceso rápido, claro y conciso a las evidencias empíricas de las que se dispone, de manera que ese conocimiento llegue a los encargados de decidir acerca de su adopción en la práctica. En el caso de MDD, esta falta de organización de la evidencia empírica, así como, cuando existe, su falta de relación con metodologías bien definidas, puede estar perjudicando su adopción por parte de las empresas; es bien sabido que la decisión de adoptar una nueva aproximación o utilizar nuevas herramientas en un proceso de desarrollo de software debería venir avalada por un proceso fiable y repetible, de manera que se maximicen las probabilidades de éxito de la implantación en la industria [13].

Con el objetivo de (a) atraer la atención de la comunidad de Ingeniería del Software sobre la falta de un acervo empírico en el campo de MDD y (b) proporcionar un mapa conceptual que organice los datos que se han publicado hasta el momento, este trabajo presenta un mapeo sistemático [14] [15] de la evidencia empírica existente acerca de cómo MDD contribuye a mejorar la calidad de la aplicación resultante y la productividad del proceso de desarrollo. Esta evidencia se relaciona directamente con las reivindicaciones 2 y 3 presentadas al inicio de esta sección.

El trabajo está organizado como sigue: en la Sección 2 se presenta en detalles el proceso de del mapeo sistemático. En la sección 3 se presenta el análisis comparativo y se discuten los resultados del mapeo sistématico y sus limitaciones. Por último, en la sección 4 se presentan las conclusiones y trabajos futuros.

2. Mapeo Sistemático

La técnica de mapeo sistemático (*systematic mapping*) define un proceso y una estructura de informe que permite categorizar los resultados que han sido publicados hasta el momento en un área determinada [15]. El objetivo de un mapeo sistemático está en la clasificación, y está por tanto dirigido al análisis temático y a la identificación de los principales foros de publicación. Permite responder preguntas genéricas como ¿Qué es lo que se ha hecho hasta el momento en el campo X? Como limitación, este tipo de estudios no toma en consideración la calidad de los estudios incluidos. Una alternativa al mapeo sistemático es la revisión sistemática [14], cuya fase de revisión de trabajos, mucho más rigurosa, permite establecer el estado de evidencia a través de la exhaustiva extracción de datos cuantitativos y estudios de meta-análisis, y por tanto responder a preguntas de investigación mucho más específicas. Estos dos tipos de estudios son complementarios y tienen como objetivo identificar los huecos de la investigación, por lo que un mapeo sistemático es considerado por muchos como un paso previo imprescindible para decidir en qué áreas concretas del campo es interesante abordar una revisión sistemática más detallada [16]. El proceso de mapeo sistemático seguido en la presente investigación se presenta en la Figura 2–1.

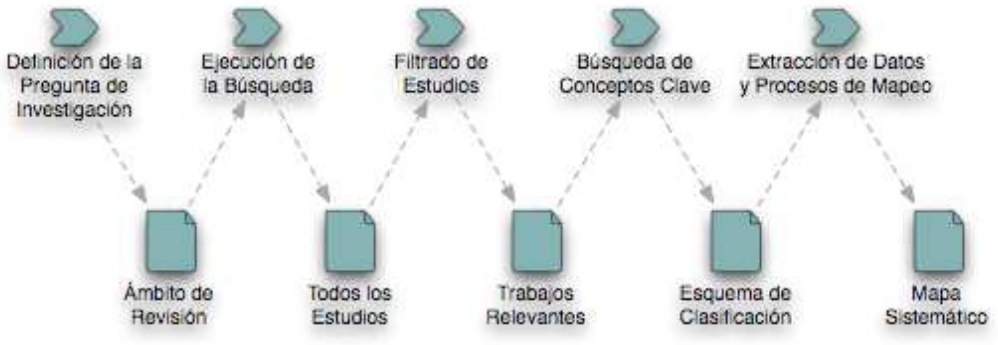


Figura 2–1: Proceso de mapeo sistemático.

2.1 Definición de la pregunta de investigación.

De acuerdo a [14], una pregunta de mapeo sistemático bien focalizada incluye cuatro partes:

Población: Profesionales interesados en migrar hacia procesos de desarrollo basados en el paradigma MDD.

Factor de estudio: Impacto de MDD sobre la calidad externa del software y la productividad del proceso.

Intervención en la comparación: Marcos de trabajo, metodologías, herramientas, etc. basadas en el paradigma MDD que han servido de base para comparar productividad del proceso y/o calidad externa de las aplicaciones resultantes con respecto a otros paradigmas.

Resultado: Grado de evidencia empírica existente en el campo.

Estos elementos nos han permitido definir las siguientes preguntas de investigación (PI):

PI1: ¿Qué tipos de publicación ofrecen más datos empíricos sobre el impacto de MDD sobre la calidad y la productividad, y cómo ha cambiado la tendencia a lo largo del tiempo?

PI2: ¿Qué características de calidad de producto (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad [5]) y proceso han sido más investigadas desde un punto de vista empírico?

PI3: ¿Qué tipo de estudio empírico (encuestas, casos de estudio, experimentos, meta-análisis) es el más usual a la hora de aseverar el impacto de MDD sobre características de calidad y productividad?

PI4: ¿Qué tipo de enfoque de investigación (validación en entornos controlados o evaluación en entornos reales) es el más utilizado en el campo?

En base a estas preguntas, el objetivo de la investigación presentada en este artículo se puede resumir como: "identificar los estudios empíricos que se han realizado durante el período 2000-2010 sobre la mejora de la calidad del producto y la productividad del proceso mediante el uso de aproximaciones MDD".

2.2. Ejecución de la Búsqueda.

La cadena de búsqueda utilizada como base para la obtención de los trabajos relevantes ha sido: empirical AND software AND (*quality or performance*) AND ("*model driven*" OR *model-driven* OR MDD OR MDE OR MDA) AND (*experiment* OR *survey* OR "*case study*" OR *meta-analysis*). La búsqueda se ha centrado en los años $2001 - 2010$. La elección del período temporal está motivada por el hecho de que el 2001 fue el año en que la OMG (*Object Management Group*) propuso la adopción de MDA (*Model Driven Architecture*) como estándar para las actividades involucradas en el MDD, aunque la guía oficial no fue publicada hasta junio 2003 [17].

Por otro lado, las fuentes de datos utilizadas han sido cuatro: *Google Scholar*, ACM, IEEE y *Springer*. Esta selección de fuentes es hasta cierto punto redundante, ya que *Google Scholar*, motor de búsqueda líder en el seno de la comunidad científica de investigadores académicos, indexa un gran número de fuentes de documentación técnica, entre las que se encuentran ACM, IEEE y *Springer*, que a su vez son los tres foros más significativos en la Ingeniería del *Software*. *Google Scholar* recupera además documentos que no aparecen en bibliotecas digitales organizadas, y que sin embargo forman parte del acervo científico de las distintas disciplinas, por lo que consideramos que es un complemento importante en la elaboración de mapeos y revisiones sistemáticas [18]. No menos importante, *Google Scholar* muestra los documentos ordenados en función de la importancia del foro de publicación en el que se encuentra el artículo, la relevancia de sus autores o la frecuencia con la que es citado el escrito. Por tanto su inclusión nos ayuda a garantizar que estamos incluyendo los trabajos más relevantes para la comunidad científica.

Los resultados de ejecutar nuestra cadena de búsqueda en *Google Scholar* fueron 896 artículos, que incluyen, como ya hemos comentado, tanto publicaciones indizadas en las librerías digitales más importantes en el área de Ingeniería del *Software* (incluidas ACM, IEEE y *Springer*) como literatura gris publicada on-line en las Webs de las universidades (informes técnicos, tesis de grado, maestrías, tesis doctorales, etc.). Una revisión preliminar de los resultados nos permitió constatar que las publicaciones relevantes para nuestro estudio se encontraban concentradas en los primeros puestos de la búsqueda, por lo que se decidió limitar el análisis más exhaustivo de las mismas a los 300 primeros trabajos.

Nuestro segundo paso de búsqueda consistió en adaptar la cadena de búsqueda a la idiosincrasia de los buscadores específicos de ACM, IEEE y *Springer*. La ejecución de la búsqueda en estos motores específicos arrojó 99 resultados (ACM), 3136 resultados (IEEE) y 317 resultados (Springer) respectivamente. Nuevamente, dado que los tres los buscadores ordenan por relevancia de la publicación, se analizaron los 100 resultados más relevantes de cada buscador, lo que nos da un total de 599 publicaciones analizadas. En la Tabla 2–1 se presenta un resumen de estos resultados.

Tabla 2–1: Resultados de la búsqueda antes y después de eliminar duplicados.

Buscador	***Google Scholar***	**ACM**	**IEEE**	***Springer***	**Total**
Resultados de la Consulta	896	99	3136	317	4448
Trabajos analizados	300	99	100	100	599
Trabajos candidatos	187	76	52	39	354
Trabajos Relevantes	40	12	6	6	64
Coincidencias con *Google Scholar*	–	9	1	4	14
Total Trabajos Relevantes	40	3	5	2	50

2.3. Filtrado de Estudios

El protocolo inicial de revisión definido para la selección de los estudios primarios se ha formulado basado en los siguientes criterios de inclusión/exclusión:

Inclusión: libros, documentos, informes técnicos y la literatura gris que describe los estudios empíricos sobre calidad o productividad en MDD, incluso aunque solo se tenga acceso al resumen del mismo.

Exclusión:

1. Artículos que no reportan estudios empíricos acerca de las mejoras en calidad o productividad cuando se utiliza una aproximación MDD. Esto implica dejar de lado cualquier trabajo centrado en justificar la mera viabilidad de la propuesta (sin comprobación empírica de las mejoras que introducen en cuanto a productividad y calidad externa del producto final).
2. Discusiones teóricas, revisiones y clasificaciones, así como propuestas de modelos de calidad que no vienen acompañados de un estudio empírico.
3. Estudios que se centran en evaluar/mejorar la calidad de los modelos y/o transformaciones que intervienen en las aproximaciones MDD, salvo que lo hagan en relación a su impacto sobre la calidad del producto final.
4. Estudios empíricos sobre aproximaciones MDD que estudian factores de contexto o datos subjetivos (e.g. los que evalúan los factores que influyen en su adopción exitosa, opinión subjetiva de desarrolladores, etc.).

2.3.1. Fiabilidad del criterio de inclusión

Para evaluar la fiabilidad de los criterios de inclusión/exclusión de los estudios relevantes, y por tanto incrementar las posibilidades de obtener resultados fiables e independientes del evaluador, se seleccionó una submuestra de la población, consistente en las 100 primeras referencias arrojadas por la búsqueda inicial en *Google Scholar*, lo que supone un 11, 16 % de la población total de los resultados arrojados por este buscador, y un 33 % de los resultados finalmente analizados del mismo. Tras el establecimiento de los criterios de revisión, el título y el resumen de dichas referencias fueron utilizados para clasificar los trabajos de manera independiente por dos revisores: E1 y E2. La fiabilidad inter-evaluador se calculó mediante el estadístico *Kappa* de *Cohen* [19]. El grado de fiabilidad arrojado por el estadístico fue satisfactorio (*Kappa*= 0,811; ver Tabla 2–2). Este grado de acuerdo indica la existencia de una base de criterios suficientemente clara y que no denota divergencias significativas entre los revisores [20].

Tabla 2–2: Tabla de Contingencia. 2^{da} fase del proceso de inclusión/exclusión (título + resumen)

		E1		**Total**
		0	**1**	
E2	**0**	86	2	**88**
	1	2	10	**12**

Durante la fase de conciliación de diferencias entre los evaluadores resultó especialmente interesante constatar la poca homogeneidad en cuanto al formato de informes de resultados empíricos, así como el uso tan libre que se sigue haciendo de palabras como "caso de estudio" o "experimento" en la literatura de Ingeniería del *Software*, a pesar de las numerosas llamadas de alerta al respecto [20]. En la práctica, esas palabras se refieren en muchas ocasiones a meros estudios de viabilidad de la propuesta, lo que complica notablemente la obtención de resultados relevantes en una búsqueda como la planteada en el presente artículo.

Los criterios validados fueron aplicados a los resultados devueltos por los cuatro motores de búsqueda utilizados como fuentes de datos para el mapeo sistemático. Finalmente

se incluyeron en el estudio un total de 50 trabajos: 40 de ellos aparecían en los resultados de *Google Scholar*, 3 en ACM, 5 en IEEE y 2 en *Springer*. En la Tabla 1 se presentan los datos pormenorizados. Analizando dicha tabla se aprecia que el grado de coincidencia de resultados entre el buscador genérico y los buscadores específicos apenas alcanza un 55 %, lo que demuestra la complementariedad de *Google Scholar* y el resto de motores de búsqueda [18]. Además, es interesante notar cómo IEEE aparece como la fuente de información peor cubierta por *Google Scholar* (ver Tabla 2–1).

2.4. Definición del esquema de clasificación.

Una vez seleccionados los trabajos relevantes se definieron, en base a los objetivos del estudio, tres tipos de clasificaciones (ver Figura 2–2):

- Medidas evaluadas: de calidad de proceso (productividad) y de calidad de producto (funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad). Esta última clasificación se basa en el modelo de calidad presentado en la norma ISO/IEEE 9126 [5].
- Tipo de estudio empírico realizado [21] [22]: caso de estudio, experimento, encuesta o meta-análisis.
- Enfoques de investigación: validación en entornos controlados o evaluación en entornos reales [23].

Además, para cada estudio relevante se recopiló información referente al año de publicación y tipo de publicación mediante el que fue diseminado.

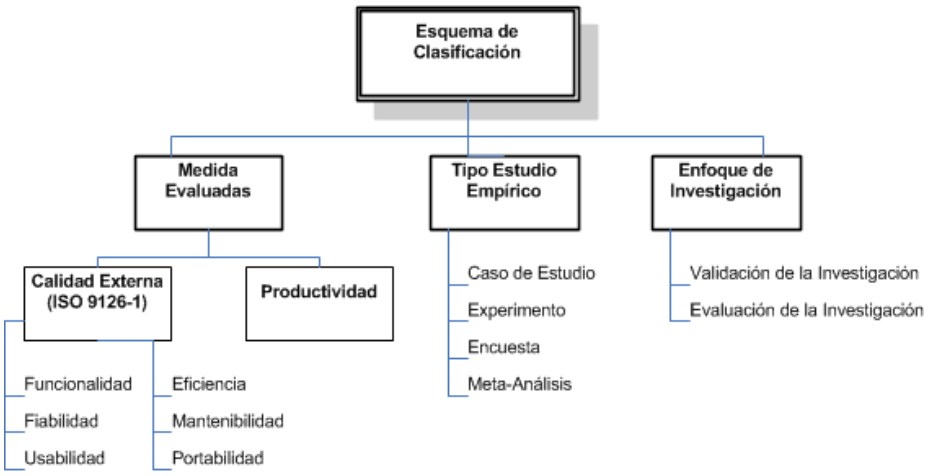


Figura 2–2: Esquema de clasificación.

2.5. Extracción de datos y Mapeo sistemático.

Tras definir el sistema de clasificación, el último paso del mapeo sistemático consiste en la extracción de datos y el proceso de mapeo de las distintas dimensiones. El resultado completo de esta actividad se muestra en el Apéndice 2.A. El resultado sintetizado de nuestro estudio se puede observar de manera gráfica en el diagrama de burbuja de la Figura 2–3, que visualiza: (1) la relación entre el tipo de estudio empírico y el estudio de medidas de calidad impactadas por el uso del paradigma MDD y (2) la frecuencia de estudios empíricos por años de publicación.

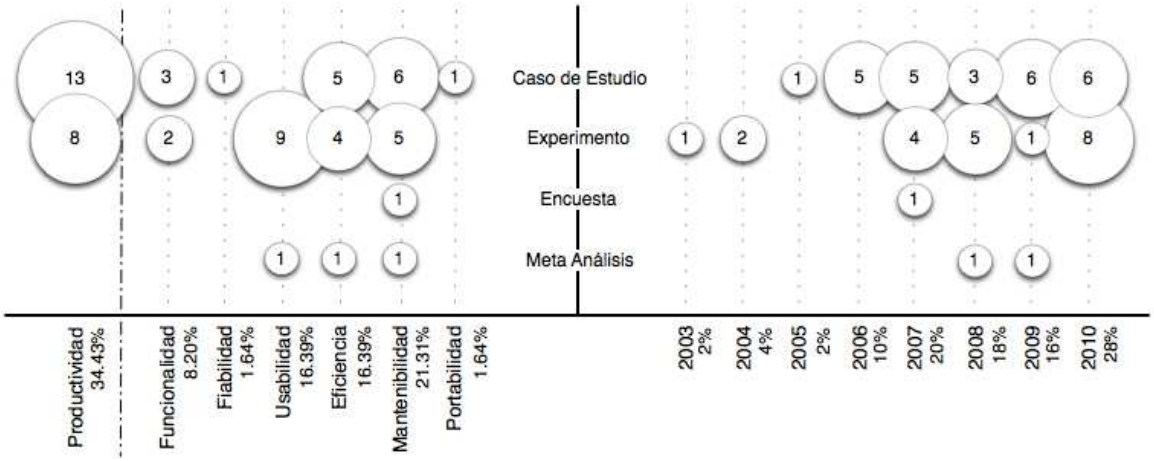


Figura 2–3: Diagrama de burbuja. Visualización del mapeo sistemático.

La Figura 2–3 ilustra básicamente dos diagramas de dispersión XY con burbujas en las intersecciones de categoría, que permite tener en cuenta varias categorías al mismo tiempo y da una visión general rápida de un campo de estudio, proporcionando un mapa visual [15]. En esta visualización de los resultados, el tamaño de una burbuja es proporcional al número de artículos que están en el par de categorías que correspondan a la burbuja de las coordenadas. Cuando un trabajo ha afectado más de una categoría (e.g. un meta-análisis de más de una característica de calidad, o presentación de más de un tipo de estudio empírico), el trabajo ha sido contabilizado en todas las características. De este modo, los porcentajes dan una visión más real sobre la cantidad de esfuerzo invertido en cada una de las dimensiones de calidad incluidas en el estudio.

De igual forma, en la Figura 2–4 se puede observar la distribución de trabajos por tipo de publicación y por enfoque de investigación. Del total de publicaciones incluidas en el mapeo (50), 21 pertenecen a revistas, 20 fueron presentadas en conferencias y 9 provienen de otros tipos de publicaciones como libros, tesis o informes técnicos. Por otro lado, 28 consistieron en validaciones en ambiente controlado, frente a 22 evaluaciones (en entornos de uso real) del paradigma.

(a) por tipo de publicación (b) por enfoque de investigación

Figura 2–4: Diagramas Circulares. Artículos por tipo de publicación y por enfoque de investigación.

3. Análisis comparativo y discusión

A continuación damos respuesta a las preguntas de investigación formuladas en la Sección 2.1. a través de los resultados obtenidos.

PI1: Tras un análisis pormenorizado de los resultados obtenidos, se puede apreciar que la publicación de estudios empíricos en el ámbito de MDD presenta una tendencia al alza, en consonancia con lo que está ocurriendo en otros campos de la Ingeniería del *Software*. Un indicador de esta tendencia es que aproximadamente 28 % de los estudios empíricos realizados en MDD se concentran en el año 2010. Los foros preferidos para este tipo de trabajos son revistas (42 %), lo que indica la relevancia otorgada por los investigadores a los resultados obtenidos. Otro dato interesante arrojado por el estudio es que, de las 599 publicaciones candidatas a ser incluidas en el estudio, solo 50 (un 8, 3 %) cumplieron los criterios para ser finalmente incluidas en el análisis, lo que, dado la especificidad de la búsqueda, por un lado demuestra el uso tan libre que se hace de los términos empíricos en el área, y por otro es un indicador de la falta de madurez empírica de MDD, lo que nos

lleva a pensar que puede ser pronto todavía para embarcarse en una revisión sistemática de dicha evidencia empírica. En caso de realizarse, los campos más prometedores serían el impacto de MDD en productividad y mantenibilidad.

PI2: La mantenibilidad (ver Figura 2–3) con un 21, 31 % de los resultados centrados en ella, es la característica de calidad externa que más ha sido investigada en el enfoque MDD. No lejos se encuentran la eficiencia y la usabilidad, con un 16, 39 % cada una. El resto de las medidas han sido estudiadas en menor escala, lo que plantea una oportunidad para futuras tesis y trabajos de investigación. Por otro lado, el estudio empírico de la productividad, como medida de calidad del proceso de desarrollo (eficiencia de proceso) ha tenido un auge representativo en los últimos 3 años, con un 34, 43 % de los resultados empíricos centrados en ella. Este dato parece sugerir que el trabajo empírico en MDD se orienta a demostrar la calidad de proceso más que la calidad de producto, lo que es consistente con la preponderancia de reivindicaciones de mejora del proceso entre las razones esgrimidas por los investigadores para la adopción del paradigma.

PI3: Como se puede observar en la Figura 2–3, los casos de estudio y los experimentos resultan ser los estudios empíricos con mayor predominio en las investigaciones actuales. Este es un dato prometedor, ya que estos estudios, donde se mide lo que hacen los sujetos, en lugar de medir lo que dicen que hacen, ofrecen un mayor grado de fiabilidad. Además, es interesante constatar cómo la usabilidad ha sido estudiada exclusivamente mediante experimentos, cuyos resultados son los más fiables a costa de un alcance necesariamente más limitado.

PI4: En cuanto a los enfoques de investigación más utilizados, los números (ver Figura 2–4) indican que la mayoría de las investigaciones actuales sobre el uso del enfoque MDD han demostrado la aplicabilidad de su propuesta a través de validaciones (estudios en entornos sintéticos). En concreto, 28 publicaciones de las 50 (56 %) han utilizado este enfoque, lo que aumenta la fiabilidad de los resultados pero no permite identificar con exactitud los beneficios e inconvenientes de su aplicación en entornos reales de desarrollo (al contrario de lo que ocurre con la evaluación de la investigación) [23]. Además, durante

el último año incluido en el estudio (2010) se ha incrementado la cantidad de estudios evaluados empíricamente tanto en entornos sintéticos como en la industria (ver Apéndice 2.A, Tabla 2.A.1), lo que nos permite afirmar que la comunidad investigadora está siendo cada vez más cuidadosa en propiciar investigaciones claras, reproducibles y con garantías de aplicación industrial.

3.1. Limitaciones del estudio

La principal limitación de este mapeo sistemático es haber analizado sólo 599 publicaciones del total de 4448 resultados obtenidos a las consultas realizadas en los diferentes buscadores (ver Tabla 2–1). Aunque esta decisión sin duda perjudica la cobertura del estudio, creemos que esta desventaja se ve paliada por el uso de los buscadores más relevantes del campo (*Google Scholar*, ACM, IEEE y *Springer*) y por la ordenación que hacen dichos buscadores de los resultados en función de su relevancia para la búsqueda.

La segunda limitación, debida a la propia filosofía del mapeo sistemático, es la calidad de los estudios incorporados. Esta limitación se podría haber evitado realizando una revisión sistemática de dichos trabajos. Sin embargo, el bajo número de publicaciones verdaderamente pertinentes (50), nos hace pensar que es todavía pronto para este tipo de evaluaciones de calidad.

Otra posible limitación de este tipo de estudios es la posibilidad de errar la clasificación por el uso ambiguo que hacen los autores de conceptos como, en nuestro caso, experimento o caso de estudio. Relacionado con esto, hemos ratificado lo que ya advertían otros autores [15] acerca de los resúmenes de los artículos, que a menudo son engañosos y carecen de información importante. En este estudio se ha limitado el impacto de estos dos riesgos mediante una aproximación conservadora al proceso de inclusión/exclusión de estudios, que ha implicado la lectura de cuantas partes del artículo hayan sido necesarias hasta poder resolver la duda de si incluir o excluir el estudio.

4. Conclusiones y trabajos futuros

Es un hecho que los estudios empíricos en el área del MDD se han incrementado sustancialmente en los últimos años, lo cual permite ya contar con ciertas evidencias del impacto de este paradigma sobre la calidad y productividad de los productos de *software*, más allá de resultados anecdóticos.

Sin embargo, el porcentaje de estudios que realmente proporcionan evidencia empírica acerca de mejoras de calidad y productividad en MDD sigue siendo muy bajo (solo la productividad del paradigma ofrece un número de estudios comparativamente significativo), lo que contrasta con otras disciplinas e incluso con otras áreas de Ingeniería del *Software* [24]. Por tanto, sigue siendo relevante enfatizar la necesidad de que la comunidad de MDD dedique una cantidad sustancial de esfuerzo a la comprobación empírica de sus aseveraciones que permita la posterior realización de revisiones sistemáticas y meta-análisis rigurosos.

De manera general, los resultados más relevantes reportados por los estudios empíricos relacionados con la productividad de MDD se pueden resumir como sigue:

- Los estudios que validan el uso de MDD en entornos académicos reportan una productividad de 2 a 9 veces superior a la obtenida con otros paradigmas de desarrollo [25]. La productividad puede llegar a ser hasta 20 veces superior según se va aumentando el tamaño del proyecto de desarrollo. Estos resultados contrastan con los reportados por experimentos en entornos industriales [26], donde los resultados son mucho más heterogéneos, y van desde los que directamente reportan una pérdida de productividad de un 10 % [27] a los que coinciden con los estudios académicos y reportan ganancias que oscilan entre un 20 % y un 35 % de productividad [27] [28] [29] [30]. Como principal limitación a estos resultados, los informes en entornos industriales se basan en estudios a pequeña escala. Las principales razones esgrimidas para justificar las pérdidas de productividad se relacionaban directamente con el uso de herramientas inmaduras y altos

costos de modelado, que puede llegar a ser tan complejo como programar directamente con un lenguaje tradicional de tercera generación.

- Por otro lado, se ha constatado cómo el uso de herramientas, librerías especializadas, etc. de soporte a MDD reduce la curva de aprendizaje y el esfuerzo de entrenamiento con este paradigma, lo que a su vez incide en la productividad del paradigma [31].
- Existen evidencias de que el enfoque MDD presenta deficiencias a la hora de expresar las reglas de diseño arquitectónicas de la aplicación, lo que actúa en detrimento de la productividad y la calidad de la aplicación final [32].

Con respecto a la mantenibilidad, los artículos incluidos en este estudio reportan que el tiempo necesario para evaluar el impacto de un cambio es substancialmente más corto (con un decremento que, en los estudios revisados, ronda el 37 %) si se usa una visualización gráfica (base del MDD) en comparación con una textual (no MDD) [33]. Además, usando aproximaciones MDD completas para el proceso de mantenimiento se mejora aún más la mantenibilidad [34]. Estos datos contrastan en cierta medida con lo reportado en [35], donde se expone que el uso de UML como lenguaje de modelado para tareas de mantenimiento no tiene un impacto significativo en el tiempo necesario para realizar un cambio, pues también se debe contar el tiempo para poner al día la documentación de UML. Sin embargo, por lo que se refiere a la exactitud funcional de los cambios (introducción de errores en el *software* durante el cambio), UML ha demostrado un impacto positivo a la hora de mejorar la calidad del código, incluso si los desarrolladores no están muy familiarizados con su uso. Otros estudios [36] [37] [38] presentan experimentos para medir el efecto del reuso, la complejidad y el acoplamiento de objetos en la mantenibilidad del *software*, aunque sin comparar distintas aproximaciones:

- La reusabilidad del *software* aumenta en detrimento de la simplicidad y mantenibilidad de algunos artefactos [36].
- Existe correlación entre la complejidad de un diagrama y las facilidades de comprensión y modificación del mismo [37].

- De igual forma, el acoplamiento entre objetos está fuertemente correlacionado con las facilidades de comprensión y modificación de expresiones OCL que pueden actuar como aserciones en el código final [38].

Si nos centramos en dominios concretos, una comparación del Modelado Orientado a Aspectos (MOA) vs la Codificación Orientada a Aspectos, los estudios experimentales y cuantitativos realizados en [39] reportan que las aproximaciones MOA generan una aplicación más pequeña, menos compleja y más modular. Además, en algunos casos el desarrollo basado en MOA acorta el ciclo de mantenimiento.

En cuanto al impacto de MDD sobre la eficiencia, funcionalidad, fiabilidad, usabilidad y portabilidad, la literatura provee resultados en su mayor parte anecdóticos, poco contrastados, lo que hace muy difícil su generalización. Los estudios se centran principalmente en presentar experiencias donde prácticas específicas (una determinada herramienta, aproximación, técnica, modelo, etc.) han mostrado sus bondades a la hora de mejorar la calidad del producto final.

A pesar de que nuestro estudio abarca el período comprendido entre los años 2001 y 2010, el trabajo empírico en el campo sigue dando frutos, y en los últimos meses se han seguido publicando estudios empíricos relevantes para nuestra investigación. En concreto, en [40] [41] se ha estudiado el impacto de los factores sociales, técnicos y orgánicos en el éxito o fracaso de la aproximación MDD en la industria. Particularmente se han encontrado una correlación positiva entre un entorno con mejores conocimientos organizativos y mejor comunicación en el equipo de desarrollo y las variables de productividad y mantenibilidad del proceso MDD, con un incremento de $66,7\,\%$ y $73,7\,\%$ respectivamente. También se han recopilado datos que parecen sugerir que el uso de modelos en la comprensión de problemas con alto nivel de abstracción incide en el aumento tanto de la productividad (hasta un $72,2\,\%$) como en la mantenibilidad (hasta un $73,4\,\%$). Otras conclusiones de estos estudios inciden en la importancia del entrenamiento y la educación para alcanzar las promesas de este paradigma.

Con el mapeo sistemático presentado en este trabajo se han conseguido identificar los principales trabajos de investigación empírica publicados hasta el momento en los principales foros científicos. Este tipo de estudios es básico de cara a facilitar la apertura del campo a nuevos investigadores. Además, este estudio ha puesto de manifiesto las enormes posibilidades de trabajo en este campo para la comunidad de Ingeniería del *Software* empírica con respecto al impacto del uso de aproximaciones basadas en el paradigma MDD sobre la calidad tanto de producto como de proceso.

De especial relevancia resulta constatar los resultados tan dispares encontrados en función de si los estudios han sido realizados en un entorno académico o industrial y, sobre todo, en función del tamaño del proyecto abordado. Otras variables que, hasta lo que alcanza nuestro conocimiento, no han sido suficientemente consideradas y que podrían influir en los resultados son los tipos de aplicaciones abordadas (e.g. si se tratan de aplicaciones intensivas en datos, transaccionales, etc.) o la experiencia de los desarrolladores. Por último, sorprende no encontrar ningún meta-análisis de productividad del paradigma, pese a que sí existen meta-análisis de otras variables con menos contribuciones. Pensamos que esto es debido, por un lado, a la disparidad en cuanto a la calidad de los informes publicados, que a veces presentan carencias que hacen imposible evaluar la calidad del estudio y agregar los datos, y por otro a la heterogeneidad de los contextos y premisas sobre las que se han construido los estudios, lo que igualmente complica su comparación y la agregación de sus resultados. El uso de plantillas y paquetes experimentales estandarizados ayudaría sin duda a paliar estos problemas y a agilizar la obtención de datos de calidad.

Agradecimientos

A Jesús Pardillo, por sus útiles comentarios durante la elaboración de este trabajo.

Referencias

[1] Muñoz, J. y Pelechano, V., "MDA vs Factorías de Software". En: Estévez, A., Pelechano V. y Vallecillo, A. (eds.), Actas del II Taller sobre Desarrollo de Software Dirigido por

Modelos, MDA y Aplicaciones (DSDM 2005). Granada (Spain), 13 de Septiembre, pp. 1, 2005.

[2] Macías, M., "Modelamiento basado en el Dominio: Estado del Arte", Revista de las I Jornadas de Ingeniería de Software 2004, vol. 1, nº 1, pp. 33-41, 2004.

[3] Selic, B., "Model-Driven Development: Its Essence and Opportunities". En: IEEE Computer Society Press (eds.), Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06). Gyeongju (Korea), April 24-26, pp. 313-319, 2006.

[4] López, E., González, M., López, M. y Iduñate, E.L., "Proceso de Desarrollo de Software Mediante Herramientas MDA", Revista Iberoamericana de Sistemas, Cibernética e Informática, vol. 3, nº 2, pp. 6-10, 2006.

[5] ISO/IEC 9126-1, Software engineering - Product quality Part 1: Quality model, ISO, 2001.

[6] Glass, R., "The Realities of Software Technology Payoffs", Communications of the ACM , vol. 42, nº 2, pp. 74-79, 1999.

[7] Pickard, M., Combining Empirical Results in Software Engineering. Technical Report vol. 1, Keele University, England, 2004.

[8] Kitchenham, B., Linkman, S. y Law, D., "DESMET: a methodology for evaluating software engineering methods and tools", Computing and Control Engineering Journal, vol. 8, nº 3, pp. 120-126, 1997.

[9] Guerini, M. L., Revisión de resultados experimentales en técnicas de prueba y de educción de conocimientos. Tesis de Magister en Ingeniería de Software, Universidad Politécnica de Madrid, España, 2007.

[10] Juristo N. y Moreno A., Basics of Software Engineering Experimentation, Kluwer Academic Publisher, 2001.

[11] Picó, R., Análisis funcional de la gestión de almacén y producción en openbravo ERP y su aplicación docente. Proyecto fin de Carrera, Escuela Técnica Superior de Informática Aplicada, Universidad Politécnica de Valencia, España, 2010.

[12] Vanderburg, G., Real Software Engineering, video/mp4 presentations of Lone Star Ruby Conference 2010, 2010. Disponible en: http://confreaks.net/videos/282-lsrc2010-realsoftware-engineering) [Consulta 17/10/2010].

[13] Kriter Software SL., Los diez factores que garantizan el éxito de la implantación ERP, 2007. Disponible en: http://www.kriter.net/item/es/software-empresa-erp-kriter-identifica-los-diez-factores-que-garantizan-el-exito-de-la-implantacion/18/25/ [Consulta 30/09/2010].

[14] Kitchenham, B. y Charters, S., Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University, Inglaterra, 2007.

[15] Petersen, K., Feldt, R., Mujtaba, S. y Mattsson, M., "Systematic mapping studies in software engineering". En: Visaggio, G., Baldassarre, M.T., Linkman, S. y Turner, M. BCS eWIC (eds.), Proceedings of 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). University of Bari (Italy), June 26 - 27, pp. 71-80, 2008.

[16] Mujtaba, S., Petersen, K., Feldt, R. y Mattsson, M., "Software product line variability: A systematic mapping study", School of Engineering, Blekinge Institute of Technology, Sweden, 2008. Disponible en: http://sites.google.com/site/drfeldt/mujtab-apsec08-sysmap-spl-variabili.pdf [Consulta 14/10/2010].

[17] Object Managament Group, Overview and guide to OMG's architecture (MDA Guide v1.0.1) OMG, 2003. Disponible en: www.omg.org/cgi-bin/doc?omg/03-06-01 [Consulta 15/12/2009].

[18] Noruzi, A., "Google Scholar: The new generation of citation indexes", Libri/Copenhagen Journal, vol. 55, nº 4, pp. 170-180, 2005.

[19] Gwet, K., "Inter-Rater Reliability: Dependency on Trait Prevalence and Marginal Homogeneity", Series of Statistical Methods for Inter-Rater Reliability Assessment, vol. 2, nº 2, pp. 1-9, 2002.

[20] Fleiss, J.L., Statistical methods for rates and proportions, John Wiley, 1981.

[21] Kish, L., "Some statisticals problems in research design", American Sociological Review, vol. 24, nº 3, pp. 328-338, 1959. [22] Mendes, E. y Mosley, N., Web Engineering, Springer-Verlag New York Inc, 2006.

[23] Wieringa, R., Maiden, N., Mead, N. y Rolland, C., "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion", Journal Requirements Engineering, vol. 11, nº 1, pp. 102-107, 2006.

[24] Zelkowitz, M., "An update to experimental models for validating computer technology", The Journal of Systems and Software, vol. 82, nº 3, pp. 373-376, 2009.

[25] Diaz, O. Villoria F.M., "Generating blogs out of product catalogues: An MDE approach", The Journal of Systems and Software, vol. 83, nº 10, pp. 1970-1982, 2010.

[26] Mohagheghi, P. y Dehlen, V., "Where is the proof? - A review of experiences from applying MDE in industry". En: Schieferdecker, I. y Hartman, A. (eds.), Proceedings of Model Driven Architecture: Foundations and Applications: 4th European Conference, (ECMDA-FA 2008), Berlin (Germany), June 09-12, pp. 432-443, 2008.

[27] MODELWARE D5.3-1 Industrial ROI, Assessment and Feedback-Master Document. Revision 2.2, 2006. Disponible en: www.modelware-ist.org [Consulta 28/09/2010].

[28] The Middleware Company, Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach: Productivity Analysis (White Paper), Report by the Middleware Company on behalf of Compuware, 2003. Disponible en: www.omg.org/mda-/mda-files/MDA-Comparison-TMC-final.pdf [Consulta 28/09/2010].

[29] MacDonald, A., Russell, D. y Atchison, B., "Model-Driven Development within a Legacy System: an Industry Experience Report". En: IEEE Computer Society Press (eds.),

Proceedings of Australian Software Engineering Conference (ASWEC 2005). Brisbane (Australia), 29 March-1 April, 2005, pp. 14-22, 2005.

[30] Lange, C. y Chaudron, M., Ïnteractive Views to Improve the Comprehension of UML Models - An Experimental Validation". En: IEEE Computer Society Press (eds.), Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC '07). Banff, Alberta (Canada), June 26-29, pp. 221-230, 2007.

[31] Zhu, L., Gorton I., Liu Y. y Bao Bui N., "Model driven benchmark generation for web services". En: Di Nitto, E., Hall, R.J., Han, J., Han, Y., Polini, A., Sandkuhl, K. y Zisman A. (eds.), Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering (SOSE 2006). Shanghai (China), May 27-28, pp. 33-39, 2006.

[32] Mattsson, A., Lundell, B., Lings, B y Fitzgerald, B., "Linking Model-Driven Development and Software Architecture: A Case Study", IEEE Transactions on Software Engineering, vol. 35 nº 1, pp. 83-93, 2009.

[33] Mellegård, N. y Staron, M., "Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment". En: Babar, M.A., Vierimaa, M. y Oivo, M. (eds.), Proceedings of Product-Focused Software Process Improvement 11th International Conference (PROFES 2010). Limerick (Ireland), June 21-23, pp. 336-350, 2010.

[34] Goldschmidt, T., Reussner, R. y Winzen, J., "A Case Study Evaluation of Maintainability and Performance of Persistency Techniques". En: Schäfer, W., Dwyer, M.B. y Gruhn, V. (eds.), Proceedings of the 30th International Conference on Software Engineering (ICSE 2008). Leipzig (Germany), May 10-18, pp. 401-410, 2008.

[35] Dzidek W.J., Arisholm, E. y Briand, L.C.A, "Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance", IEEE Transactions on Software Engineering, vol. 34, nº 3, pp. 407-432, 2008.

[36] Lucrédio, D., "Evaluating the impact of MDD in software reuse", 2009, Disponible en: www.les.inf.puc-rio.br/ [Consulta 03/11/2010].

[37] Manso, M. E., Cruz-Lemus, J. A. Genero, M. y Piattini, M., "Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study". En: Chaudron, M.R.V. (ed.), Proceedings of the Workshop in Model Software Engineering: Workshops and Symposia in MODELS 2008). Toulouse (France), September 28-October 3, pp. 303-313, 2008.

[38] Reynoso, L., Genero, M. and Manso, E., "Measuring Object Coupling in OCL Expressions: A Cognitive Theory-Based Approach". En: Daponte, P. (ed.), Proceedings of the 23rd IEEE Instrumentation and Measurement Technology Conference (IMTC 2006). Sorrento (Italy), April 24-27, pp. 1087-1092, 2007.

[39] Hovsepyan, A., Scandariato, R. Van Baelen, S., Berbers, Y. y Joosen, W., "From Aspect-Oriented Models to Aspect-Oriented Code? The Maintenance Perspective". En: Jézéquel, J. M. y Südholt, M. (eds.), Proceedings of the 9th International Conference on Aspect-Oriented Software Development (AOSD 2010). Rennes and Saint Malo (France), March 15 -19, pp. 85-96, 2010.

[40] Hutchinson, J., Rouncefield, M. y Whittle, J., "Model-driven engineering practices in industry". En: Taylor, R.N., Gall, H. y Medvidovic, N. (eds.), Proceeding of the 33rd International Conference on Software Engineering (ICSE 2011). Waikiki, Honolulu (Hawaii), May 21-28, pp. 633-642, 2011.

[41] Hutchinson, J., Whittle, J., Rouncefield, M. y Kristoffersen, S., "Empirical assessment of MDE in industry". En: Taylor, R.N., Gall, H. y Medvidovic, N. (eds.), Proceeding of the 33rd International Conference on Software Engineering (ICSE 2011). Waikiki, Honolulu (Hawaii), May 21-28, pp. 471-480, 2011.

Apéndice 2.A: Publicaciones relevantes clasificadas

Tabla 2.A.1. Mapa sistemático

Publicación	Año	Fuente	Tipo de estudio	Medida de Calidad	Enfoque de Investig.
The value of conceptual modeling in database development: An experimental investigation	2003	GS	E	Productividad	V
The Impact of Software Reuse and Incremental Development on the Quality of Large Systems	2004	GS	E	Productividad, Mantenibilidad	Ev
Configuring real-time aspects in component middleware	2004	GS	E	Eficiencia	Ev
Alfresco Content Display	2005	GS	CE	Productividad	Ev
From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company	2006	GS	CE	Productividad	Ev
Applying model-driven development to distributed real-time and embedded avionics systems	2006	GS	CE	Productividad	V
Analysis of crosscutting in model transformations	2006	GS	CE	Mantenibilidad	V
Applying system execution modeling tools to evaluate enterprise distributed real-time and embedded system QoS	2006	GS	CE	Eficiencia	V
Model driven benchmark generation for web services	2006	ACM	CE	Productividad	Ev
A model-driven architecture approach using explicit stakeholder quality requirement models for building dependable information systems	2007	GS	CE	Fiabilidad	V
Quicker: A model-driven QoS mapping tool for QoS-enabled component middleware	2007	GS	CE	Funcionalidad	V
An activity-based quality model for maintainability	2007	GS	CE	Mantenibilidad	V
A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor	2007	GS	CE	Productividad	Ev
mTurnpike: a Model-driven Framework for Domain Specific Software Development	2007	GS	E	Eficiencia	V
Building measure-based prediction models for UML class diagram maintainability	2007	GS	E	Usabilidad, Mantenibilidad	V
Measuring Object Coupling in OCL Expressions: A Cognitive Theory-Based Approach	2007	GS	E	Usabilidad, Mantenibilidad	V
Reliable Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems	2007	GS	CE	Eficiencia	Ev
Interactive views to improve the comprehension of UML models-an experimental validation	2007	IEEE	E	Productividad	V
Survey of traceability approaches in model-driven engineering	2007	IEEE	S	Mantenibilidad	Ev
Where is the proof?-a review of experiences from applying MDE in industry	2008	GS	CE	Productividad	V
Transitioning from code-centric to model-driven industrial projects–empirical studies in industry and academia	2008	GS	E	Productividad	V
Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project	2008	GS	E	Funcionalidad, Usabilidad, Productividad	Ev
Usability evaluation of user interfaces generated with a model-driven architecture tool	2008	GS	E	Usabilidad	V
Analyzing the Influence of Certain Factors on the Acceptance of a Model-based Measurement Procedure in Practice: An Empirical Study	2008	GS	E	Usabilidad	V
A realistic empirical evaluation of the costs and benefits of uml in software maintenance	2008	IEEE	E	Mantenibilidad	Ev
A case study evaluation of maintainability and performance of persistency techniques	2008	IEEE	CE	Productividad Mantenibilidad	Ev
A Model-Based Framework for Security Policy Specification, Deployment and Testing	2008	Springer	CE	Eficiencia	Ev

Model-Driven Performance Evaluation for Service Engineering	2008	Springer	MA	Eficiencia	Ev
Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process	2009	GS	CE	Productividad	V
Embedded System Construction–Evaluation of Model-Driven and Component-Based Development Approaches	2009	GS	CE	Mantenibilidad	Ev
Evaluating the Correctness and Effectiveness of a Middleware QoS Configuration Process in Distributed Real-time and Embedded Systems	2009	GS	CE	Portabilidad	Ev
Level of detail in UML models and its impact on model comprehension: A controlled experiment	2009	GS	E	Usabilidad	V
Empirical Validation of Measures for UML Class Diagrams: A Meta-Analysis Study	2009	GS	MA	Usabilidad, Mantenibilidad	V
MDA-based tool chain for web services development	2009	GS	CE	Productividad	V
A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare	2009	GS	CE	Productividad	V
Linking model-driven development and software architecture: A case study	2009	IEEE	CE	Productividad	V
Model-driven development for early aspects, Volume 52, Issue 3	2010	GS	CE	Mantenibilidad , Eficiencia	Ev
Distribution of Effort Among Software Development Artefacts: An Initial Case Study	2010	GS	CE	Funcionalidad	V
The impact of structural complexity on the understandability of UML statechart diagrams	2010	GS	E	Usabilidad	V
MOOGLE: A model search engine	2010	GS	E	Productividad	Ev
An Empirical Investigation of the Utility of 'pre-CIM'models	2010	GS	E	Productividad	Ev
Towards to the validation of a usability evaluation method for model-driven web development	2010	GS	E	Eficiencia, Usabilidad	Ev
Productivity Analysis of the Distributed QoS Modeling Language	2010	GS	CE	Funcionalidad, Productividad	V
Usability evaluation of multi-device/platform user interfaces generated by model-driven engineering	2010	GS	E	Usabilidad	V
Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment	2010	GS	E	Funcionalidad, Usabilidad,	V
Moppet: A Model-Driven Performance Engineering Framework for Wireless Sensor Networks	2010	GS	CE	Mantenibilidad	Ev
2010 36th EUROMICRO Conference on Sofware Engineering and Advanced Applications (The Impact of Model Driven Development on the Software Architecture Process)	2010	GS	CE	Eficiencia	V
Enterprise systems development: Impact of various software development methodologies	2010	GS	CE	Productividad	V
From aspect-oriented models to aspect-oriented code?: the maintenance perspective	2010	ACM	E	Mantenibilidad	Ev
Generating blogs out of product catalogues: An MDE approach	2010	ACM	E	Productividad	Ev

Leyenda:

GS: *Google Scholar*	**CE**: Caso de Estudio **E**: Experimento **MA**: Meta análisis **S**: Encuesta (*Survey*)	**Ev**: Evaluación **V**: Validación

.

CAPÍTULO 3
IMPACT OF MDE APPROACHES ON THE MAINTAINABILITY OF WEB APPLICATIONS: AN EXPERIMENTAL EVALUATION

El contenido de este capítulo corresponde con el siguiente artículo: *Martínez, Y., Cachero, C. and Matera, M. and Abrahao, S. and Luján, S. - Impact of MDE approaches on the maintainability of Web applications: an experimental evaluation. Proceeding of Conceptual Modeling ER* 2011 *: 30th International Conference on Conceptual Modeling, Brussels, Belgium, October* 31*-November* 3*,* 2011*, pp* 233 − 246.

El contenido de este capítulo fue divulgado en la conferencia internacional sobre modelado conceptual (ER). Esta es una de las conferencias más importantes en los datos y el modelado de proceso, la tecnología de base de datos, y las aplicaciones de base de datos. Esta conferencia es un amplio foro de debate para investigadores y expertos industriales interesado en todos aspectos de base de datos y diseño de sistemas de información.

ER es considerado como una de las conferencias mejor rankeadas, está indizado en el *ranking* ERA (*The Excellence in Research for Australia*) 2010 (categoría A) y tiene un impacto aproximadamente de 0.90 de acuerdo con *The Conference Ranking,* 29/07/2011 (http://www.cs-conference-ranking.org). El índice de aceptación de esta conferencia ronda aproximadamente en un 20 %.

Impact of MDE approaches on the maintainability of Web applications: an experimental evaluation

Yulkeidi Martínez[1], Cristina Cachero[2], Maristella Matera[3], Silvia Abrahao[4] and Sergio Luján[2]

[1] Universidad Máximo Gómez Báez de Ciego de Ávila, Cuba

[2] Universidad de Alicante, Spain.

[3] Politecnico di Milano, Italy

[4] Universidad Politécnica de Valencia, Spain

Abstract. Model-driven Engineering (MDE) approaches are often recognized as a solution to palliate the complexity of software maintainability tasks. However, there is no empirical evidence of their benefits and limitations with respect to code-based maintainability practices. To fill this gap, this paper illustrates the results of an empirical study, involving 44 subjects, in which we compared an MDE methodology, WebML, and a code-based methodology, based on PHP, with respect to the performance and satisfaction of junior software developers while executing *analysability*, *corrective* and *perfective* maintainability tasks on Web applications. Results show that the involved subjects performed better with WebML than with PHP, although they showed a slight preference towards tackling maintainability tasks directly on the source code. Our study also aims at providing a replicable laboratory package that can be used to assess the maintainability of different development methods.

1. Introduction

It is well known that maintenance is the most expensive phase in the software life cycle, absorbing between 45 % and 60 % [1] of total management costs. This situation has led many researchers to focus on maintainability from different perspectives. On the one

hand, *unstructured maintenance* postulates wading straight into the code of the target application in order to make the necessary changes, normally with the aid of specialized tools. Sometimes this process relies on -automatic- maintainability measures that operate over the code of the target application [2]. On the other hand, *structured maintenance* examines and modifies the design, and then -either manually or automatically- reworks the code to match it. Many authors claim that structured maintenance is a more reliable and efficient process than unstructured maintenance [3].

The Model Driven Engineering (MDE) paradigm goes one step further in the lane of structured maintenance, and advocates the use of models and model transformations to speed up and simplify the maintenance process. The MDE community claims several advantages over code-based maintenance processes, such as short and long term productivity gains, improved project communication and defect and rework reduction [4, 5]. Unfortunately, despite all these claims and the fact that MDE practices are maturing by the day, practitioners still lack a body of practical evidence that soundly backs the purported maintainability gains due to the use of this paradigm [5, 6]. As Glass says, "We (practitioners) need your help. We need some better advice on how and when to use methodologies" [7]. Such evidence can be provided in the shape of empirical studies - such as surveys, experiments, case studies or postmortem analysis studies - that directly measure the effect of the chosen paradigm on the developer performance and satisfaction [8]. Many authors have written about the importance of providing empirical evidence in Software Engineering (SE) [9, 10]. Unfortunately, the percentage of empirical studies on maintainability in MDE approaches is still very low, contrasting with other disciplines and even other areas of SE [11]. This paper tries to fill this gap, and presents the results of a quasi-experiment in which two groups of junior developers performed a set of maintainability tasks on two Web applications.

The paper is structured as follows: Section 2 presents the rationale behind the selection of WebML and PHP as typical examples of the two current paradigms (model-driven and code-based) in Web applications development, as well as the definition of maintainability

that we are using all along the paper, and how it has been assessed in the past in MDE methodologies. This background sets the context for the definition of hypotheses and variables in Section 3, together with the experimental design (subjects, instrumentation, operation and data collection mechanisms). Section 4 presents the data analysis and an interpretation of results that takes into account the identified threats to validity. Last, Section 5 concludes the paper and presents some further lines of research.

2. Background

The last years have witnessed a continuous evolution in the scenario of Web application development. A number of technologies have been proposed, fostering the adoption of server-side and client-side languages for the Web. Within this plethora of proposals, some technologies have been largely adopted by practitioners. Server-side scripting is one such technology, through which the generation of HTML markup is achieved by means of languages like JSP, PHP, and ASP.NET. Among them, numbers point at PHP as the most popular server-side scripting language today [12]. Given such diffusion, in our experiment we have chosen it as representative of the programming languages for the Web.

In parallel to the technology innovation line, the Web Engineering community has devoted several efforts to the definition of MDE approaches [13], strongly characterized by the adoption of *conceptual models*. Such approaches offer high-level abstractions, capturing the most salient characteristics of Web applications, which can be used for the definition of application models abstracting from implementation details. MDE also stresses the importance of model transformations, leading to the automatic generation of the application code starting from the high-level conceptual models. One of the arguments most commonly brought forward in favor of MDE approaches is the "ease of maintenance". The commonly accepted (but scarcely proved) claim is that the adoption of model-driven design techniques in the development of a Web application implicitly enhances maintainability: requests for changes can be turned into changes at the conceptual level and then propagated systematically down to the implementation code.

Among the MDE Web development methodologies so far proposed, WebML (Web Modeling Language) [14] outstands, due to its broad adoption both in academia and industry. WebML is a visual language and development method for specifying the content structure of a Web application and the organization and presentation of contents in the form of hypertext. The main contribution of WebML is the proposal of a mix of concepts, notations, and techniques for the construction of data-intensive Web applications, which blends traditional ingredients well known to developers, such as conceptual data design with the Entity-Relationship model, with new concepts and methods for the design of hypertext, which are central to Web development. Therefore, the value of WebML lies not in the individual ingredients, but in the definition of a systematic framework, also equipped with a CASE tool[5] offering support for all the design activities and for the automatic code generation.

2.1 The maintainability concept

Although many definitions for maintainability exist, perhaps the most commonly used is the one provided by the ISO [15], that is, "the capability of the software product to be modified". Maintainability can be assessed through measures associated to the following sub-characteristics:

- *Analysability*: Capability of the software product to be diagnosed for deficiencies or causes of failure in the software, or for the parts to be modified to be identified.
- *Changeability*: Capability of the software product to enable the application of a specified modification.
- *Stability*: Capability of the software product to avoid unexpected effects from modifications of the software.
- *Testability*: Capability of the software product to enable modified software to be validated.

[5] WebRatio - http://www.webratio.com

- *Compliance*: Capability of the software product to meet the standards or conventions relating to maintainability.

In MDE environments not all these sub-characteristics are equally important, though. Provided that the chosen MDE approach is mature enough (such as is the case of WebML), we can safely assume testability and compliance: in an automated code generation environment these issues are taken good care of during the code generation process. Therefore, in this paper, we have centered on the first three sub-characteristics: analysability, changeability and stability.

In relation to the changes that can take place during the life cycle of a system, some authors also distinguish among four different modification types [16]:

- *Corrections*: Corrective maintainability refers to the capability to detect errors, diagnose the problems and fix them.
- *Improvements*: Perfective maintainability refers to the capability to extend the software according to new requirements or enhancements.
- *Adaptations*: Adaptive maintainability refers to the capability to modify the software in order to cope with the effects of environmental changes.
- *Preventions*: Preventive maintainability refers to the software capability to support internal reengineering processes without adverse impact maintainability.

Some authors use a more coarse classification, and merely distinguish between *corrective tasks* (corrections) and *evolution tasks* (which encompass improvements, adaptations and preventions).

Based on these two characterizations of maintainability, our experiment includes three types of tasks: *analysability tasks*, which cover the analysability subcharacteristic, and *corrective* and *perfective tasks*, which cover the two most common changeability types. These tasks have been assessed both objectively and subjectively. The perceived stability of changes has also been measured.

2.2 Maintainability assessment

The empirical assessment of maintainability activities in MDE methodologies has been gaining momentum during the last years. This claim is supported by a systematic mapping [17] that gathers the empirical evidence on software productivity and quality gains due to the use of MDE approaches during the period from 2001 to 2010 [18]. In this study, out of the 300 publications related to the impact of the MDE paradigm on the product quality and process productivity, only 9 (that is, barely a 3 %) focused on maintainability. One of the conclusions of this systematic mapping is that the bulk of the work on model-driven maintainability lies in the definition of prediction models of the actual maintainability of the applications that are derived from the conceptual artifacts. However, there is little evidence on the impact of model-driven development methodologies on the performance and satisfaction of maintainability tasks. According to our systematic mapping, out of the 9 maintainability papers, only [19] tackled this issue, and concluded that the time to evaluate the impact of a change to the software was significantly shorter (around 37 % shorter) in MDE than in code-based development environments[6].

3. Description of the Experiment

Following the GQM template [20], our empirical study is aimed at *analyzing* WebML and PHP *for the purpose of* evaluating model-driven against code-based maintenance practices *with respect* to their performance and satisfaction *from the point of view of* young developers and prospective adopters. The context of the study is two advanced SE courses at the University of Alicante and the Politecnico di Milano, where both a code-based Web development process, based on the PHP language, and a model-driven Web development process, based on the WebML methodology, are taught.

The design of the experiment was based on the framework for experimentation in SE suggested by [8]. The study was conducted as a laboratory blocked subject-object quasi-experiment. The broad research questions addressed are:

[6] Due to space constraints, for a more detailed discussion about the results of the systematic mapping, readers are referred to [18]

RQ1: Is the maintainability performance (actual efficiency and actual effectiveness) of the WebML methodology higher than the actual maintainability performance of a traditional, code-based, methodology using PHP?

RQ2: Is the maintainability satisfaction (perceived ease of use and perceived usefulness) of the WebML methodology higher than the maintainability satisfaction of a traditional, code-based methodology using PHP?

These research questions are further explained and decomposed into research hypotheses next.

3.1 Variables and operational hypotheses

The independent variable of our study is the methodology used to perform the maintainability tasks: model-driven (WebML methodology) or code-based (tasks performed directly over PHP code). Hence, the experiment defines two treatments: maintainability tasks over WebML models and maintainability tasks over PHP code. The collected experimental data allow comparing the effects of both treatments.

The theoretical model underlying the definition of dependent variables and the formulation of hypotheses of our experiment is presented in Fig. 3–1 [21]. According to this model, the actual performance of the developers influences their satisfaction, and this in turn contributes to their intention to use the method and, eventually, to its actual usage. The actual performance is made up of two subcomponents: the *actual effectiveness*, which in our experiment is measured through the precision and recall while performing tasks with the methodology, and the *actual efficiency*, which is measured through the time that it takes to fulfill the maintainability tasks. On the other hand, the satisfaction is also made up of two subcomponents: the *perceived ease of use*, which in our experiment is measured through a global satisfaction scale, and the *perceived usefulness*, which is measured through the perceived subjective complexity, certainty and stability experienced by subjects while using the methodology to perform the maintainability tasks. All these measures are further explained below.

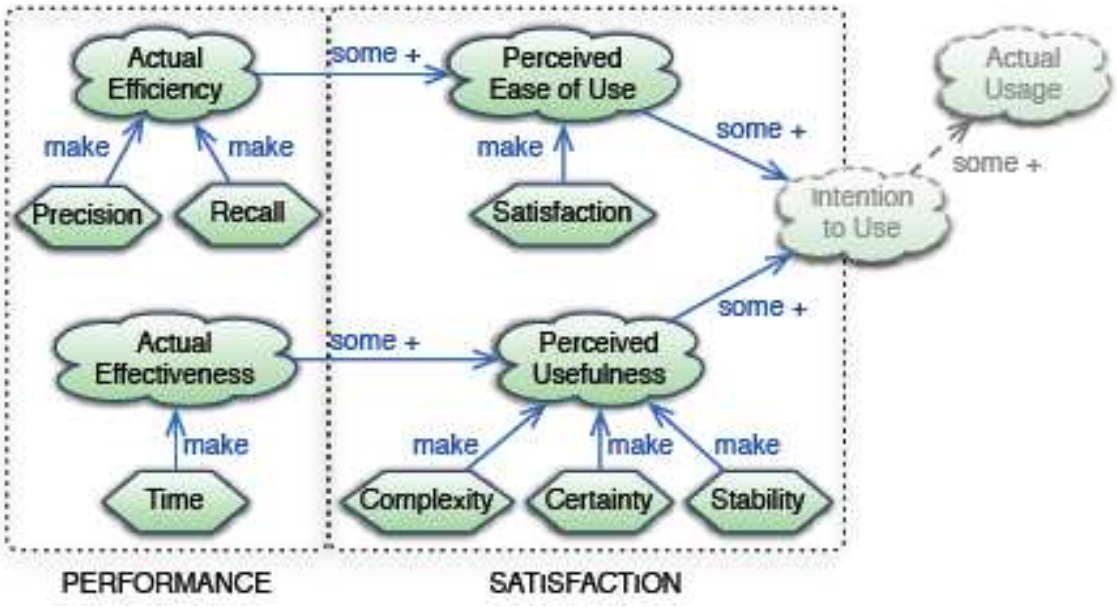


Figura 3–1: Theoretical model components and their associated experimental variables (adapted from [21]).

It results that in our experiment there are two types of dependent variables for comparing the treatments: *performance-based* (precision, recall and time) and *satisfaction-related* (complexity, certainty and, in the case of corrective and perfective tasks, perceived stability).

Maintainability tasks involve a set of actions on models or code. Some of these actions may be correct while others may be wrong, not contributing to the maintenance problem at hand. Still, the subject may miss some actions that would be needed to completely solve the task. For this reason, inside the group of performance-based variables, we have distinguished between *maintainability precision* (MP) and *maintainability recall* (MR). Precision is defined as the percentage of actions correctly identified by the subjects with respect to the total number of actions reported, while recall refers to the ratio between the percentage of correct actions performed by the subject and the total number of actions needed to completely solve the task. The third performance variable is *maintainability time* (MT), which is the time (in seconds) that it takes a subject to complete each maintainability task.

As perception variables, we have used the *maintainability perceived complexity* (MPCx), *the maintainability perceived certainty* (MPCt), *the maintainability perceived stability* (MPSt) and a *maintainability* (global) *satisfaction* scale (MS). Also, we have defined a global *perceived learnability* (PL).

The hypotheses tested in the experiment, which have been defined based on the existing empirical evidence found during the systematic mapping presented in Section 2, and which cover the performance and satisfaction components of the theoretical model presented in Fig. 3–1, are:

- **HAA** (Actual Analysability): Analyzing maintainability issues over WebML models allows for a better actual performance than analyzing them over PHP code.
- **HACC** (Actual Corrective Changeability): Correcting errors over WebML models allows for a better actual performance than correcting them over PHP code.
- **HAPC** (Actual Perfective Changeability): Improving a Web application over WebML models allows for a better actual performance than improving it over PHP code.
- **HPA** (Perceived Analysability): Subjects feel that finding errors overWebML models is less complex; they also feel more certain about the results.
- **HPCC** (Perceived Corrective Changeability): Subjects feel that correcting errors over WebML models is less complex and more stable than doing so over PHP code; they also feel more certain about the results.
- **HPPC** (Perceived Perfective Changeability): Subjects feel that introducing improvements over WebML models is less complex and more stable than doing so over PHP code; they also feel more certain about the results.
- **HS** (Satisfaction): Generally speaking, subjects feel more satisfied when performing maintainability tasks with WebML than with PHP.
- **HP** (Learnability): Generally speaking, subjects believe that PHP is easier to learn than WebML.

3.2 Subjects

The experimental subjects of our study were two groups of Computer Science students. The first group included twelve students enrolled in the "Internet Programming" course in Alicante. The second group included thirty-two students enrolled in the "Web Technologies" course in Milano, Italy. All the students aged between 22 and 24 years, and they all had a similar background. The subjects were chosen for convenience, i.e., they were students enrolled in the corresponding courses during the second term of the year 2009-2010. The necessary preparation for the experiment and the experimental tasks themselves fitted well into the scope of the courses, as one of the course objectives was to teach students how to maintain applications in PHP and WebML respectively.

3.3 Design and instrumentation

The geographical dissemination of students made necessary to design the experiment as a between-subject experiment. Subjects in Milano used the WebML approach to perform the maintainability tasks and subjects in Alicante used PHP to perform the same maintainability tasks over the same applications. For both parts of the experiment we used two different sample applications to test the hypotheses: (1) a Concert application (adaptation of an example used in [22]) and (2) a Bookstore application (used in [23]). Subjects inside each group were randomly assigned to either of these applications. To make the results obtained in Alicante (with PHP) and in Milano (with WebML) comparable, the PHP and the WebML experiment instruments were designed to be as similar as possible in terms of both the layout of the information and the information content. Also, the applications were chosen to be of similar complexity, which was controlled by assuring that both applications had a similar number of lines of code in PHP and, at the same time, a similar number of conceptual constructs in WebML.

The instrumentation of the experiment consisted of:

- WebML Booklet (Instrument 1): for each application, a domain model (ER diagram) and corresponding WebML hypertext schema, describing the Web interface.

- PHP Booklet (Instrument 2): for each application, a domain model (Class diagram), an application storyboard (to understand the file structure of the application) and the PHP source files of the application.

These instruments are included in the replication package available at http://www.dlsi.ua.es/∼ccachero/labPackages/er2011/labPackage.rar. Prior to the experiment, all the instruments were tested in two pilot studies, one in Alicante and one in Milano, and the obtained feedback was used to improve the understandability of the experimental materials. We also want to stress that the PHP applications exactly corresponded with the applications generated from the hypertext schemas in WebML. Therefore, the results obtained with both procedures can be compared, as both groups received the same applications, albeit in different formats.

The experiment had the following structure:

1. Subject instruction sheet.
2. Pre-experiment questionnaire: it included demographic questions as well as questions about subjects' previous experience with Web application development, Web programming and application modelling.
3. Experimental tasks: For each treatment, two modalities (A and B), each one featuring a different application. Each modality included an analysability, a corrective changeability and a perfective changeability task. Each task was accompanied by a post-task questionnaire which assessed the perceived usefulness of the methodology (see Fig. 3–1) while performing the task (usersatisfaction with the results achieved) through three measures (complexity, certainty and stability), each measured on a 5-point Likert scale. The tasks were exactly the same across treatments (that is, Modality A of the WebML treatment was exactly the same as Modality A of the PHP treatment, except for the treatment).
4. Post-experiment questionnaire: it included a semantic-differential scale that required developers to judge the application on 11 pairs of adjectives describing satisfaction with the development methodology. This scale assessed the perceived ease of use of the methodology (see Fig. 3–1), and showed a high degree of reliability ($\alpha = 0{,}861$).

Based on this scale, the ease of use index was computed averaging scorings for the items. The post-experiment questionnaire also included an item gathering the subjects' impression on the learnability of the treatment they had applied to perform the tasks.

3.4 Operation and data collection procedures

In both groups, all the subjects performed the experiment during the last session of the course. In this way we controlled that they had received the same amount of training on the treatment they were going to apply. We are conscious that 30 hours of course (many of which are devoted to theoretical issues, not directly related with the treatment) is not enough to master a development process, although it was equal for both groups. Also, most of the students assigned to the PHP treatment in Alicante had worked with PHP before the training course, which was not the case with WebML students in Milano. Therefore, our group of Milano students approaches a sample of novice software modelers, while the group of Alicante students approaches a sample of intermediate programmers. This is an internal threat to the validity to the study that has conditioned our interpretation of the results, as we will see in Section 4.

The operation phase for each group of the experiment was defined as follows. First, the students filled in the pretest questionnaire. They chose a nickname that was used for the remaining parts of the experiment, so that they were sure that the results were not going to be used to grade them in any way. Then, half of the students received the Modality A, where the presented application was the concert application, and half of them received the Modality B, where the presented application was the bookstore application. In both modalities the user had to perform a set of analysability, corrective and perfective changeability tasks. Also, they had to subjectively rate the complexity of each task, as well as their perceived certainty and stability of their solution. Last, they received a post-test questionnaire, where they expressed their satisfaction with the treatment and their general perception of the methodology learnability (based on the training sessions).

To maintain the comparability of the data collected during the experiment, no feedback was given to the subjects on their performance with the tasks. We also controlled that no interaction whatsoever between participants occurred. The experiment had a time limit (2 hours). Although the time needed to complete the experiment was checked with the pilot tests, we are conscious that this time limitation may introduce a ceiling effect that we have also taken into account when interpreting the results.

The performance-based variables were collected using data collection forms. These forms recorded the errors found (for analysability tasks) and the corrective and perfective actions needed (for corrective and perfective changeability). Also, forms included the time spent on each task (controlled by the experiment instructor) and the subjective opinion of the students, both regarding each task and regarding the global methodology.

4 Data analysis and interpretation of results

Due to space constraints, the tables containing the descriptive statistics for the set of measures taken for the different maintainability task types (analysability, corrective changeability and perfective changeability) are available at http://www.dlsi.ua.es/∼ ccachero/labPackages/er2011/descrStats.pdf. In the remainder of the paper particular values from these tables are commented when necessary.

4.1 RQ1: Actual efficacy of treatments

For the sake of the reliability of the statistical analysis results, and due to the relatively low number of subjects on the PHP group, we chose to apply the Mann-Whitney U non parametric test, which makes no assumptions about normality of distributions. All the analyses were performed with the PASW Statistics application, v18[7]. It is important to note that not all the tasks were filled in by all the subjects, which is the reason why the different hypotheses testing procedures may present different degrees of freedom.

We first tested the HAA hypothesis, which is related to the precision, recall and time of subjects while detecting errors in the systems using each of the two treatments.

[7]PASW - http://www.spss.com/software/statistics/

The result of the test show that both groups differ significantly in precision ($U(41) = 80{,}5; Z = -3{,}011; p = 0{,}003$) and time ($U(41) = 345{,}5; Z = 4{,}609; p < 0{,}001$), and that in both cases the WebML group showed a better performance than the PHP group. However, the analysability recall (that is, the number of actual errors found out of the total number of errors) was not significantly different ($U(41) = 170; Z = -0{,}286; p = 0{,}775$). Otherwise stated, performing analysability tasks on WebML seems to speed up the identification of errors, and to avoid misclassifying a feature in the application as an error. However, it does not seem to significantly help to detect the errors in the application.

We then tested the hypothesis HACC, which is related to the precision, recall and time of subjects while correcting actual errors in the systems using each of the two treatments. The result of the test show that, again, both groups differ significantly in precision ($U(40) = 104; Z = -2{,}163; p = 0{,}031$) and time ($U(39) = 252; Z = 2{,}484; p = 0{,}013$). The WebML group shows again a better performance than the PHP group. However, the corrective changeability recall (that is, the number of actual errors corrected out of the total number of errors) was not significantly different ($U(40) = 132; Z = -1{,}31; p = 0{,}19$). Otherwise stated, performing corrective changeability tasks on WebML seems to speed up the correction of errors, and to avoid proposing corrections that do not actually correct anything. However, it does not seem to significantly help to correct the actual errors in the application.

The last hypothesis we tested was HAPC, which related to the precision, recall and time that took subjects to introduce perfective modifications in the applications. There were not significant differences between the mean WebML changeability precision and the mean PHP changeability precision ($U(40) = 137{,}5; Z = -1{,}22; p = 0{,}222$). Similarly, the perfective recall (percentage of actions that actually contributed to successfully implement the change) was not significantly higher in WebML than in PHP ($U(40) = 165; Z = -0{,}283; p = 0{,}777$). However, subjects were again significantly quicker performing the improvements in WebML than in PHP ($U(40) = 336; Z = 4{,}647; p < 0{,}001$).

Since, as we have aforementioned, WebML subjects were novice developers, while PHP subjects had a higher level of experience, we can rely more heavily on the found significant differences in performance (all pointing at WebML having a higher performance than PHP for maintainability tasks): had WebML subjects had more experience with the methodology, the only natural evolution of the measures would have been towards making differences larger in favor of WebML.

4.2 RQ2: Perceived efficacy and satisfaction of treatments

For all the statistical analysis, we again performed a non-parametric Mann-Whitney U test.

Regarding HPA, the result of the test shows that both groups differ significantly in subjective complexity assessment ($U(37) = 84; Z = -2{,}195; p = 0{,}028$) but not in subjective certainty assessment ($U(37) = 92; Z = -1{,}871; p = 0{,}061$). Otherwise stated, the subjects regard working on models as simpler than working on code, but they feel equally (un)secure about the errors they identify.

Regarding HPCC, users felt that changes were slightly less complex when using PHP ($U(36) = 140; Z = -0{,}179; p = 0{,}858$), although this difference was not significant. Also, users felt slightly more certain about their changes in WebML ($U(36) = 125; Z = -0{,}355; p = 0{,}723$). Last but not least, since in this case users were actually changing the application, we could ask about their feeling of stability. Users felt the stability of the system to be significantly better preserved when using PHP ($U(36) = 224; Z = 3{,}152; p = 0{,}002$). Otherwise stated, users seem to feel more sure about what needs to be done when using WebML, but they seem to feel more 'in control' of the changes when they directly work on code.

Regarding HPPC, WebML was considered slightly simpler to introduce perfective changes than PHP, although the statistical test showed no significance ($U(37) = 105; Z = -1{,}55; p = 0{,}121$). On the contrary, subjects using PHP felt slightly more confident about

the correctness of the changes introduced, although, again, the difference was not significant with respect toWebML subjects ($U(37) = 187{,}5; Z = 1{,}31; p = 0{,}19$). Regarding stability, both groups showed very similar levels of confidence on the stability of the solution ($U(36) = 136{,}5; Z = 0{,}053; p = 0{,}958$).

Regarding HS, as we have aforementioned, the post-tests questionnaire included a scale of satisfaction with the technique. The satisfaction indexes with both treatments were compared by a t-test. Results showed a non-significant effect of the methodology ($t(38) = -0{,}208; p = 0{,}836$). On average, evaluations were moderately positive for both methodologies, with a mean difference of 0.06 slightly favoring the PHP group.

Last but not least, we tested HP; the global satisfaction index (measured through the satisfaction scale) is highly correlated to the measure (a direct item) assessing learnability of the methodology ($r = 0{,}663; p < 0{,}001$). This suggests that the easier the methodology was perceived to be, the better it was evaluated.

4.3 Threats to Validity

The analysis of the threats to validity evaluates under which conditions our experiment is applicable and offers benefits, and under which circumstances it might fail [24].

Threats to Conclusion Validity refer to the relationship between the treatment and the outcome. In our experiment we have used the number of lines of code to control the application size. This measure is generally preferred over functional points due to its higher reliability [8]. Additionally, statistical tests have been chosen conservatively, without making any kind of assumption on variable distributions. However, given the limited amount of time that students had to fulfil the experiment questionnaire, it is possible that the subjects have felt a pressure that may have affected the data. This notwithstanding, since both treatment groups suffered from the same time restrictions, we can assume that such effect, if present, has affected all the levels of the treatment equally.

Threats to Internal Validity are concerned with the possibility of hidden factors that may compromise the conclusion that it is indeed the treatment what causes the

differences in outcome. In our case, the students belonged to different countries, which may bias the results. We limited the effect of the lack of randomization of subjects assigned to treatments by assuring that the mean age for the WebML group of students ($M = 22{,}47$) was not significantly different from the mean age for the PHP group of students ($M = 23{,}17$). Also, we assured that they had been enrolled in a similar number of courses on related modelling/programming topics, and had developed a similar number of WebApps ($M(WebML) = 3{,}93$; $M(PHP) = 4{,}25$; $t(27) = -0{,}183$; $p = 0{,}18$). Only the experience with code could not be controlled ($M(WebML) = 7{,}27$; $M(PHP) = 69{,}27$).

This difference can be explained by the structure of the degrees in universities, which still today mainly focus on programming more than modelling skills. We tried to diminish the effect of different facilitators running the experiment in the two locations by compiling an instruction sheet that avoided unintended differences in the conditions of the experiment. Another threat to the internal validity of this study is related with the instrumentation. Since precision and recall measures had to be manually computed by different evaluators (each one an expert in WebML and PHP respectively), there is a risk of the criteria being applied differently to each group. We have tried to avoid this risk by defining a clear taxonomy of errors that were agreed upon before the evaluation.

Threats to Construct Validity refer to the relationship between theory and observation. In this sense, both the treatments and the measures used to assess the maintainability have been previously widely used in literature. This notwithstanding, there are still some risks we need to be aware of. First, we have used similar size applications, all belonging to the same domain (monomethod bias). Therefore we cannot generalize the results to applications of different sizes or different domains. Second, we have observed a positive outcome between maintainability and WebML, but we cannot assure that using WebML does not hamper other quality characteristics (restricted generalizability across constructs).

Last but not least, **Threads to External Validity** are concerned with generalization of the results. In our case, the fact that we have used undergraduate students, that is, subjects without experience in business, as well as the particular methodologies and languages we have used, constitute a limited environment. The latter is an unavoidable risk for this kind of experiments, in which subjects need to use a predefined approach to perform the tasks. The easiest way to minimize this threat is through replication of the experiment with different languages and MDE approaches. For this purpose, the replication package of this experiment can be found at http://www.dlsi.ua.es/∼ ccachero/labPackages/er2011.rar.

5 Conclusions

Generally speaking, the use of MDE engineering approaches such as WebML improves the precision and time it takes to perform analysability, corrective and perfective maintainability tasks. Also, junior developers, with a stronger experience in coding rather than on modeling, perceive maintainability tasks over models as simpler. This notwithstanding, subjects still rely more on their maintainability outcomes (certainty and perceived stability of the solution achieved) when they perform the maintainability tasks directly over PHP code. One reason for this result can be still the major confidence about code hold by the users involved in our experiment. This certainty and perceived stability however slightly tips the satisfaction balance in favor of code-based approaches.

Our results augment the repository of empirical data comparing maintainability performance and satisfaction of MDE methodologies with respect to traditional code-based approaches. The laboratory package that accompanies our research allows for replications with the same or different methodologies, languages, applications and subject profiles. Such replication is at the base of our future research.

Acknowledgements

The authors wish to thank the students who kindly agreed to participate in this empirical study.

Referencias

[1] Ruiz, F., Polo, M.: Mantenimiento del Software. Grupo Alarcos, Departamento de Informática de la Universidad de Castilla-La Mancha. (2007)

[2] Coleman, D., Ash, D., Lowther, B., Oman, P.: Using metrics to evaluate software system maintainability. Computer 27(8) (2002) 44-49

[3] Ameller, D., Gutiérrez, F., Cabot, J.: Dealing with non-functional requirements in model-driven development. (2010)

[4] López, E., González, M., López, M. y Iduñate, E.L., "Proceso de Desarrollo de Software Mediante Herramientas MDA", CISCI: Conferencia Iberoamericana en Sistemas, Cibernética e Informática (2007)

[5] Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, IEEE (2009) 113-120

[6] Mohagheghi, P.: An Approach for Empirical Evaluation of Model-Driven Engineering in Multiple Dimensions. In: From Code Centric to Model Centric: Evaluating the Effectiveness of MDD (C2M:EEMDD), CEA LIST Publication (2010) 6-17

[7] Glass, R.L.: Matching methodology to problem domain. Communications of the ACM 47(5) (2004) 19-21

[8] Wohlin, C., Runeson, P., Host, M.: Experimentation in software engineering: an introduction. Springer Netherlands (2000)

[9] Dyba, T., Kitchenham, B.A., Jorgensen, M.: Evidence-based software engineering for practitioners. Software, IEEE 22(1) (2005) 58-65

[10] Kitchenham, B., Budgen, D., Brereton, P., Turner, M., Charters, S., Linkman, S.: Large-scale software engineering questions-expert opinion or empirical evidence? Software, IET 1(5) (2007) 161-171

[11] Zelkowitz, M.V.: An update to experimental models for validating computer technology. Journal of Systems and Software 82(3) (March 2009) 373-376

[12] Wikipedia. http://en.wikipedia.org/wiki/PHP

[13] Vallecillo, A., Koch, N., Cachero, C., Comai, S., Fraternali, P., Garrigós, I., Gómez, J., Kappel, G., Knapp, A., Matera, M., Others: MDWEnet: A practical approach to achieving interoperability of model-drivenWeb engineering methods. In:Workshop Proc. of 7th Int. Conf. on Web Engineering (ICWE07), Italy, Citeseer (2007)

[14] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Morgan Kaufmann series in data management systems: Designing data-intensive Web applications. Morgan Kaufmann Pub (2003)

[15] ISO/IEC FCD 25010: Systems and software engineering - Software product. Requirements and Evaluation(SQuaRE) - Quality models for software product quality and system quality in use. (2009)

[16] Chapin, N., Hale, J.E., Khan, K.M., Ramil, J.F., Tan, W.G.: Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution: Research and Practice 13(1) (2001) 3-30

[17] Kitchenham, B., Mendes, E., Travassos, G.H.: Cross versus Within-Company Cost Estimation Studies: A Systematic Review. IEEE Transactions on Software Engineering 33(5) (May 2007) 316-329

[18] Martínez, Y., Cachero, C., Melia, S.: Evidencia empírica sobre mejoras en productividad y calidad mediante el uso de aproximaciones MDD: un mapeo sistemático de la literatura. REICIS (submitted) (2011)

[19] Mellegå rd, N., Staron, M.: Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment. Product-Focused Software Process Improvement (2010) 336-350

[20] Perry, D.E., Porter, A.A., Votta, L.G.: Empirical studies of software engineering: a roadmap. In: The Future of Software Engineering, ACM (2000) 345-355

[21] Moody, D.L.: Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models (PhD Thesis). Melbourne, Australia: Department of Information Systems, University of Melbourne (2001)

[22] Abrahão, S., Mendes, E., Gomez, J., Insfran, E.: A model-driven measurement procedure for sizing Web applications: design, automation and validation. Model Driven Engineering Languages and Systems (2007) 467-481

[23] Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6) (2000) 137-157

[24] Cook, T.D., Campbell, D.T., Day, A.: Quasi-experimentation: Design & analysis issues for field settings. Houghton Mifflin Boston (1979)

.

CAPÍTULO 4
EVALUATING THE IMPACT OF A MODEL-DRIVEN WEB ENGINEERING APPROACH ON THE PRODUCTIVITY AND THE SATISFACTION OF SOFTWARE DEVELOPMENT TEAMS

El contenido de este capítulo corresponde con el siguiente artículo: Martínez, Y., Cachero, C. and Meliá, S. - Evaluating the impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams. Proceeding of 12*th International Conference on Web Engineering ICWE* 2012*: Berlin, Germany, july* 23-27*,* 2012*, pp* 223-237*.*

La conferencia internacional ICWE 2012 (del inglés *International Conference on Web Engineering*) tiene por objetivo promocionar la excelencia científica y práctica sobre ingeniería Web, y al traer juntos a investigadores y profesionales que trabajaban en tecnologías, metodologías, herramientas, y técnica usadas para desarrollar y mantener aplicaciones Web que resulten en mejores sistemas, y por lo tanto, a permitir y mejorar la diseminación y el uso del contenido y servicios a través de la Web. Este artículo, además fue publicado en el libro *Web Engineering*, M. Brambilla *et al.* (Eds.), LNCS 7387, pp. $223 - 237$, *Springer-Verlag Berlin Heidelberg* 2012. DOI: $10{,}1007/978 - 3 - 642 - 31753 - 8 - 17$, ISBN: $978 - 3 - 642 - 31752 - 1$.

Este congreso está indizado en el *ranking* ERA 2010 (categoría C).

Evaluating the impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams

Yulkeidi Martínez[1], Cristina Cachero[2], and Santiago Meliá[2]

[1] Universidad Máximo Gómez Báez de Ciego de Ávila, Cuba

[2] DLSI. Universidad de Alicante, Spain.

Abstract. BACKGROUND: Model-Driven Engineering claims a positive impact on software productivity and satisfaction. However, few efforts have been made to collect evidences that assess its true benefits and limitations.

OBJECTIVE: To compare the productivity and satisfaction of junior Web developers during the development of the business layer of a Web 2,0 Application when using either a code-centric, a model-based (UML) or a Model-Driven Engineering approach (OOH4RIA).

RESEARCH METHOD: We designed a full factorial, intra-subject experiment in which 26 subjects, divided into five groups, were asked to develop the same three modules of a Web application, each one using a different method. We measured their productivity and satisfaction with each approach.

RESULTS: The use of Model-Driven Engineering practices seems to significantly increase both productivity and satisfaction of junior Web developers, regardless of the particular application. However, modeling activities that are not accompanied by a strong generation environment make productivity and satisfaction decrease below code-centric practices. Further experimentation is needed to be able to generalize the results to a different population, different languages and tools, different domains and different application sizes.

1. Introduction

It is a well known fact that the Web Engineering community advocates the use of models in order to improve software development processes for Web applications. However, there are many issues around modeling that are, as of today, cause of controversy and heated debates: to which extent should practitioners model? Which should be the level of detail of these models? Should practitioners strive to maintain the models current, or should these models be disposable? These and others are open questions whose answer currently partly depends on the development culture of the person asked, and partly on the context in which such practices are being adopted. In this respect we claim that, instead, the decision about which is the adequate application of software modeling practices should be answerable based on objective data regarding its impact on well-known process and product quality dimensions. From these dimensions, productivity, defined as a ratio of what is produced to what is required to produce it, outstands, due to its impact during the selection of a development process in industry [1]. Also, satisfaction is an important aspect of quality, since, being software development a human process, the developer's satisfaction is a key factor for the successful adoption of such practices [2].

One quite well-known way of classifying modeling practices in industry is according to the extent to which modeling is used to support the development process. Fowler [3] describes three different modes in which modeling languages (and the UML in particular) can be used: sketch, blueprint and programming language.

- Sketches are informal diagrams used to communicate ideas. They usually focus on a particular aspect of the system and are not intended to show every detail of it. It is the most common use of the UML, and the recommended practice in agile, code-centric frameworks like Scrum. When models are used as sketches, tools are rarely used, the modeling activity being mostly performed in front of blackboards where designers join to discuss complex or unclear aspects of the system. They are most useful in code-centric approaches, where the objective is to develop self-explaining code.

- Blueprints are diagrams that show most of the details of a system in order to foster its understanding or to provide views of the code in a graphical form. Blueprints are widely used in Model-Based Development (MBD) practices, such as the ones promoted by frameworks such as the Rational Unified Process (RUP).
- Last but not least, models can be used to fully characterize the application. If such is the case, the diagrams replace the code, and they are compiled directly into executable binaries. This is the modeling use that lies at the core of Web Engineering Model-Driven Development (MDD) approaches.

This classification has led some authors to characterize the modeling maturity level of organizations based on the role of modeling in their software development process, from manual, code-centric, to full, Model-Driven [4]. While code-centric development methods require - at most - an informal use of modeling techniques and languages, both MBD and MDD require a formal use of models, which mainly relies on the use of Computer Aided Software Engineering (CASE) tools. These tools may offer not only modeling environments - including model checkers that may assure syntactical correctness, semantic accurateness, consistency or completion of models, to name a few desirable model characteristics - but also partial or even complete software generation environments that, in the case of MDD CASE tools, are based on model transformations.

Both MBD and MDD tools work under the assumption that designing models that can generate partial or complete code is much simpler and quicker than actually writing such code. This same view is sustained in MBD and MDD related scientific literature. Such literature claims that the two most outstanding advantages of MDD over code-centric or even MBD approaches are (a) short and long term process productivity gains [5] and (b) a significant external software product quality improvement [6-8]. These advantages are justified in literature by the higher level of compatibility between systems, the simplified design process, and the better communication between individuals and teams working on the system that the MDD paradigm fosters [9]. However, neither the MBD nor the MDD research community have yet been able to provide practitioners with a sufficient

body of practical evidence that soundly backs the purported gains of their recommended modeling practices with respect to code-centric approaches [10, 11]. Many authors have written about the importance of providing empirical evidence in software engineering [12] but, unfortunately, the percentage of empirical studies -be them surveys, experiments, case studies or postmortem analysis [13]- that provide data to illustrate the impact of MBD and MDD approaches over different quality characteristics (such as productivity or satisfaction) is still very low, which hampers the generalizability of the results. In order to guarantee such generalizability, we also need to take into explicit consideration many factors that may affect these characteristics, such as tool usage, adaptation of the development methodology to the idiosyncrasy of the particular development team, type, complexity and size of the project, and so on. This situation contrasts with other disciplines and even other areas of Software Engineering [14], and it is often mentioned as one of the causes that explain the low adoption level of modeling practices by the practitioner's mainstream [15].

Given this situation, the aim of this paper is to augment the repository of empirical data that contributes to giving a scientific answer to the following research question: *What is the impact of the development method (be it rooted in a code-centric, an MBD, or an MDD paradigm) on the productivity and satisfaction of junior developers while developing Web* 2,0 *applications?*

In order to answer this question, Section 2 describes some previous studies that center on the impact of MBD and MDD practices on process productivity and satisfaction with respect to traditional code-centric practices. Section 3 outlines our experiment design, and analyzes its results and threats to validity. Finally, Section 4 presents the conclusions and further lines of research.

2. Background

Although still scarce in number and not always systematically performed [16], in the last years we have witnessed an increase in the number of empirical studies that provide

empirical data regarding the impact of MBD and MDD practices on the productivity and satisfaction of software development teams.

Regarding MBD, in [5] initial evidence is provided about the use of models and tools as an effective way to reduce the time of development and to improve software development variables such as productivity and product quality.

Regarding MDD, there is a number of experiments where productivity of developers using different methods -some model-driven, others code-centered- was measured [10, 17-21]. The conclusion in all these studies is that, as projects grow larger, the use of MDD development practices significantly increases productivity (results ranging from two up to nine or even twenty times, depending on the study). The only evidence contradicting these findings is an industrial experience presented in [15]. In this paper, the authors reported a set of studies that showed contradictory results, with both software productivity gains and losses, depending on the particular study. They explained the found productivity losses by pointing at the use of immature tools and high start-up costs. Also, these studies showed that modeling was considered to be an activity at least as complex as programming with a traditional third generation language.

Last but not least, although most of the aforementioned studies include some kind of global satisfaction score, there are few studies that center on the developers' subjective perceptions while applying different methods. In [22], the author empirically assessed the satisfaction of an MDD method (called MIMAT) that includes Functional Usability Features (FUFs) in an MDD software development process. The study concluded that the users' satisfaction improves after including FUFs in the software development process. Our experiment does not center on the impact of a method enrichment, but compares different methods with respect to the developer's satisfaction and productivity.

Next, we present the quasi-experiment that we have performed to test the impact of three methods, each one an example of a code-centric, an MBD and an MDD approach respectively, on the productivity and satisfaction of junior software developers.

3. Description of the Experiment

During the months of January and February 2011, a quasi-experiment was conducted at the University of Alicante. A quasi-experiment differs from a true experiment in that subjects are not randomly chosen. Quasi-experiments, although suffering from a lower internal validity, are widely used and deemed useful in the Empirical SE field, since they allow investigations of cause-effect relations in settings such as ours, in which randomization is too costly [23].

3.1 Goals and context definition

Following the GQM template [24], our empirical study is aimed at *analyzing* three methods, one representative of the code-centric paradigm, one representative of the MBD paradigm and one representative of the MDD paradigm, *for the purpose of* comparison *with respect to* their productivity and satisfaction *from the point of view of* junior software developers. The context of the study was a set of M.Sc. students developing the business layer of a Web 2.0 application.

The design of the experiment was based on the framework for experimentation in SE suggested by [13]. The whole data set is included in the replication package available at http://www.dlsi.ua.es/∼ccachero/labPackages/Productivity.v1.rar.

The research questions addressed in this study were formulated as follows:

RQ1: Is the team's productivity significantly different among methods, regardless of the particular module being developed?

RQ2: Is the developer's satisfaction significantly different among methods, regardless of the particular module being developed?

These research questions were devised to be answerable by quantitative means.

Subjects and application The initial group of subjects were 30 students of the Web Applications Developer Master at the University of Alicante. These students were divided into six teams of 4 to 6 people. From them, the data corresponding to Team 6 had to be

dropped due to some of their components abandoning the Master for work reasons soon after the experiment had started. Therefore, the final set of observations corresponds to the observations of the remaining five groups (26 subjects). Since the abandonment of the group had nothing to do with the application being developed, the treatments that the group were applying to his project nor the particular order in which they were applying them, we can assume that the results of the experiments have not been compromised. The final sample comprised 25 men and 1 women, of whom 75 % had more than 2 years of professional experience as developers of web applications. The mean age of the participants was 25, 6 years old and all of them were Computer Engineering graduates of the University of Alicante.

Regarding the subjects' level of knowledge with respect to the different technologies and methods used during the experiment, a questionnaire showed that 81 % knew UML, and that another 12 % considered that they had a high-level of knowledge of UML. It should also be noted that 76 % of the subjects had previously programmed with VS and .NET during their degree courses, although only 12 % had applied them in industry. Finally, the subjects acknowledged no previous practical knowledge of MDD, although 56 % of them were aware of the existence of the paradigm. By the time the experiment took place, the subjects had received additional training in all three methods. Such training consisted in 30 hours of training in programming in *C#* using Visual Studio 2010, 20 hours of training in UML modelling with RSM and 10 hours of training in modelling with the OOH4RIA tool.

Each of the five groups developed a social media application for a different domain:

- *Trips*: the social network for this domain is focused on establishing relationships between people who want to reduce travel costs by sharing their own cars.
- *Events*: the social network for this domain is centered on organized social events.
- *Hospitals*: the social network for this domain aims at improving the communication between physicians and patients.

- *Academics*: the social network for this domain focuses on connecting and sharing teaching contents among teachers and students.
- *Facework*: the social network for this domain helps workers to share information about different tasks, resources and goals of the company.

All the applications shared the same complexity, which was controlled by defining a set of functional features that all the applications had to support, regardless of the domain. From them, the three functional features that were included in our experiment were:

- Support for the establishment of a community of users (from now on Group) to create contents and relationships among people of different environments (professional, personal, etc., depending on the particular application being developed).
- Support for the organization of events (from now on Events) where people can invite their friends or colleagues to attend to a place where the event is realized (the particular event being a celebration, a work meeting, etc. depending on the particular application being developed).
- Support for an organizational community (from now on Organization) where subjects (be them companies, celebrities, etc., depending on the particular application) can publish content, photos, etc. in a unidirectional way to the social network.

Each one of these functional features was designed as a module. In order to further control the complexity of each module, we strictly defined their architecture, which was based on four main layers: the Business Objects layer (BO), the Data Access Objects layer (DAO), the Data Transfer Objects layer (DTO) and the Database layer (DB). In this way, it was possible to standardize to a certain point the code that had to be developed and facilitate its measurement. Although we are conscious that such strict architecture may hamper the external validity of the experiment, this factor was kept constant across the three treatments, in order to preserve the comparability of the results. The subjects were asked to implement each module following a different method. The time assigned for the implementation of each module was two weeks.

In order to develop the different projects, the students had to follow the Agile Unified Process (Agile UP) methodology [25], a streamlined approach to software development that is based on the IBM's Rational Unified Process (RUP) [26]. The Agile UP lifecycle is serial in the large, iterative in the small, and delivers incremental releases over time. Specifically, our experiment was situated in the construction phase of Agile UP, which is focused on developing the system to the point where it is ready for pre-production testing. The construction phase is made up of a set of disciplines or workflows that groups different tasks of this process. These disciplines, together with the impact of modelling practices on each of them depending on the paradigm, are presented in Table 4–1. All the students had previously developed at least one application with Agile UP, and they had an additional 10-hour training period to refresh the main concepts.

Tabla 4–1: Degree of automation of Agile UP disciplines by development paradigm.

Discipline	Code-Centric(.NET)	Model-Based(RSM)	Model-Driven(OOH4RIA)
Model	Sketch or absent	Blueprint	Fully-fledged (DSL)
Implementation	Manual	Semi-automatic	Automatic
Test	Manual	Manual	Semiautomatic

Implementation Language and Case Tools

The development environment for the experiment was set up as follows:

- Code Development Environment: NET framework and NHibernate (Object-Relational Mapping).
- IDE (Integrated Development Environment) Development Tool: Visual Studio 2010.
- Languages: *C#* and the Extensible Application Markup Language (XAML).
- MBD Modeling Environment: Rational Software Modeler (RSM).
- MDD Modeling Environment: OOH4RIA IDE.
- Other tools: Subversion 1.6 (SVN), Jira (Issue Tracking) and Survey Monkey (for questionnaires).

The code-centric treatment relied solely on the development enviroment provided by the Visual Studio 2010 and the use of external tools that permit to manage the collaborative work (Subversion). On the other hand, the MBD treatment required the students to work with the UML class diagram of the RSM tool. Last but not least, for the MDD treatment the students worked with the OOH4RIA approach [27].

Students were scheduled to work on these three modules during six weeks along the months of January and February 2011. By this time of the year, the students had already gone through most of the topics of the master, and had gathered a substantial amount of experience with the different tools and the development environments. The experiment defined a tight timetable of deliverables, one every two weeks. Each deliverable consisted of a set of source files and a domain model. The source code had to contain four specific file types: the Business Object files (BO), the Data Access Object files (DAO), the Data Transfer Object files (DTO) and the Database files (DB). The teams were continuously monitored by a Master instructor whose role in the experiment was to look after the quality of the data gathered, both in class and off-line through the Jira and the SVN report systems.

3.2 Experiment Planning

Given the low number of development teams available, and in order to facilitate the detection of method impact by controlling the variability among subjects, the experiment was conceived as an intra-subject design. The combination team-module- approach was defined using a Factorial Design [28, 29] (see Table 4–2). This kind of design avoids order effects and can provide some unique and relevant information about how variables interact or combine in the effect they have on the dependent variables. Also, this design eliminates any possible order effect. Teams were randomly assigned to each treatment order.

In order to answer the research questions presented in Section 3.1, we have defined the following independent (experimentally manipulated) variables (IV) or factors:

Tabla 4–2: Diseño experimental: diseño factorial e intra-sujetos.

Equipo-Módulo	Aplicación	Grupo	Eventos	Organización
1	*Trips*	*code-centric*	MBD	MDE
2	*Events*	*code-centric*	MDE	MBD
3	*Hospital*	MBD	MDE	*code-centric*
4	*Academics*	MDE	MBD	*code-centric*
5	*Facework*	MBD	*code-centric*	MDE
6*	*Automobile*	MDE	*code-centric*	MBD
Group marked with * did not finish the experiment.				

- Meth: Method, a categorical variable with three levels: code-centric, MBD, MDD. It is important to note that, in this experiment, when we refer to method we are in fact talking about a compound variable (method*tool), due to the coupling of these two variables in our experimental settings.
- Mod: Module, a categorical variable with three possible values: Groups, Events, Organization.

The Dependent (measurable) variables (DV) are:

- P(Meth, Mod), a ratio variable that measures the productivity of the team with each method and module.
- S(Meth,Mod), an interval variable, based on a 7-point Likert scale, that measures the satisfaction of the developers with each method and module.

The DV have been measured through the following collection procedures:

1. To measure Productivity we measured both the development time and the size of the modules developed by each team.
   - Development time: The student had to document the time of each development activity through the JIRA tool.
   - Module size: We measure the size of the code produced by students in source lines of code (SLOC). SLOCs come in handly to express the size of software among programmers with low levels of experience [30]. We automated the obtention of this measure through the Line Counter [31] tool.

2. To measure Satisfaction we defined a satisfaction scale (SS) made up of eleven items, where each one was based on a 7-point Likert rating scale.

These measures have been used to test the following testable hypotheses, which are based on the research questions and the existing empirical evidence presented in Section 2:

- Productivity Hypothesis (PH): Prod(MDD) > Prod(MBD) > Prod(code-centric). Developer teams are significantly more productive with the MDD method, followed by the MBD method, followed by the code-centric method.
- Productivity-Module Interaction Hypothesis (PMIH): P(Module*Meth)< 0,05. The effect on P of the particular module to which the method is applied is insignificant compared to the effect of the method.
- Satisfaction Hypothesis (SH): Satisf(MDD) > S(MBD) > S(code-centric). Developers are significantly more satisfied with the MDD method, followed by the MBD method, followed by the code-centric method.
- Satisfaction-Module Interaction Hypothesis (SMIH): S(Module*Meth) < 0,05. The effect on S of the particular module to which the method is applied is insignificant compared to the effect of the method.

3.3 Instrumentation

Besides the instructional materials, all the students received three booklets:

- Modules Description Booklet: a requirements document describing the functional and non functional requirements of the three modules included in the experiment. This booklet was divided in three parts, and it was the same regardless of the order in which the treatments were to be applied.
- Jira Time Reporting Booklet: a document explaining the time reporting conventions that were to be used during the experiment.
- Subject Instruction sheet: a set of instructions to the students to correctly perform the experiment.

These instruments are included in the replication package available at http: //www.dlsi.-ua.es/∼ccachero/labPackages/Productivity.v1.rar.

The experiment had the following structure:

1. Subject instruction sheet.
2. Pre-experiment questionnaire: it included demographic questions as well as questions about subjects' previous experience with Web application development, Web programming and application modeling, etc.
3. Project work: For each treatment, the students spent two weeks working on the corresponding module with the assigned methodology.
4. Post-experiment questionnaire: it included a semantic-differential scale that required developers to judge each method on 11 pairs of adjectives describing the developer's overall satisfaction with such method.

At the end of each module, the students delivered both the Domain Models and the Source Code (BO, DAO, DTO and DB files).

3.4 Data analysis and interpretation of results

The statistical analysis was carried out with PASW (Predictive Analytics SoftWare) Statistics [32].

Prior to the assessment of the hypotheses, we checked the reliability of the Satisfaction scale in the context of our experimental settings. For the satisfaction scale, all the items showed a correlation higher than 0,3, while the global Cronbach alpha was 0,892, giving proof of a high internal consistency among the scale items. This high correlation has led us to calculate the scale mean for each method, and consider this mean as a global rating of satisfaction with each one of the three treatments (code-centric, MBD, MDD).

RQ1: Impact of method on team productivity. The data gathered to accept/reject the PH and PMIH hypotheses (see section 3.2) are graphically presented in Fig. 4–1

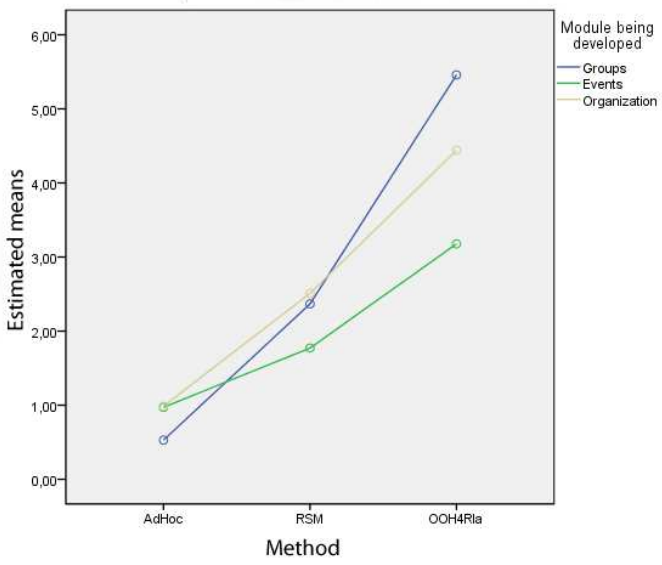


Figura 4–1: Productivity: SLOC/Hours.

To test the PH and PMIH hypotheses, we applied a 3*5 Mixed Design ANOVA , in which the module (Groups, Events, Organization) was the between subjects variable, and the calculated P ratings for each method were the within subjects variables. In order to assure that applying this statistical method made sense, we verified that the principle of spherity was not violated by applying the W Mauchly's test ($W = 0{,}005; p > 0{,}05$) [33]. The results showed that MDD produced the highest P ($M = 4,60; SD = 1,17$), followed by MBD ($M = 2,30, SD = 1,10$) and then Code-centric ($M = 0{,}80; SD = 0{,}29$), and that these differences were significant ($F = 25{,}395; p = 0{,}001$).

The results also showed that the interaction Mod*Meth was not significant ($F = 3{,}009; p > 0{,}05$). We can then safely examine the main effects of the two independent variables (Mod and Meth) on these means without needing to qualify the results by the existence of a significant interaction. The main effect of module did not attain significance ($F = 0{,}538; p > 0{,}05$), while the main effect of method did reach significance, ($F = 25{,}39; p < 0{,}01$), that is, the differences in P are significantly affected by the method used, regardless of the particular module being developed. The last step of the analysis consisted on studying the pairwise differences among methods through a one-way RM Anova with pairwise comparisons. In order not to augment the risk of a type−1 error, a Bonferroni

adjustment was applied. This means reducing the significance threshold to 0,0167 ($p = 0{,}05/3 = 0;0167$). With this adjustment, the differences in productivity between the Code centric and the MBD method did not attain significance ($t = -2{,}69; p = 0{,}054$) but the differences between Code-centric and MDD ($t = -6{,}029; p = 0,004$) and MBD and MDD ($t = -6{,}031; p = 0{,}004$) did.

RQ2: Impact of method on developer's satisfaction. The data gathered to accept/reject the SH and SMIH hypotheses (see section 3.2) are graphically presented in Fig. 4–2

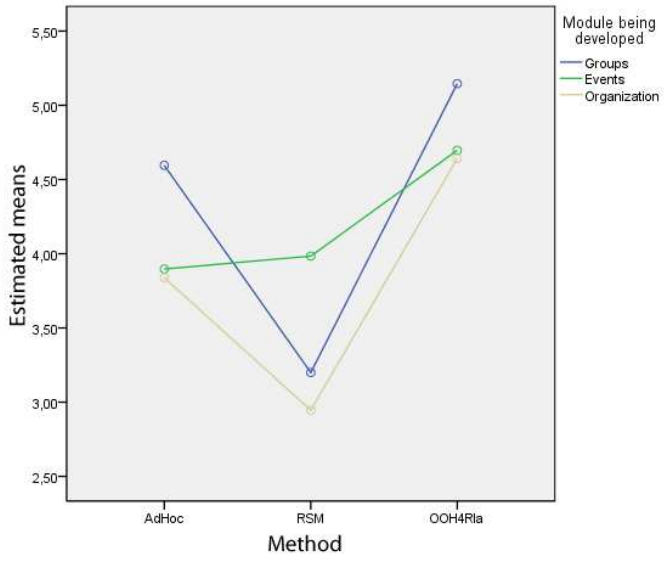


Figura 4–2: Satisfaction: Likert Scale.

To test the SH and SMIH hypotheses we applied a 3*5 Mixed Design ANOVA, in which the module (Groups, Events, Organization) was the between-subjects variable, and the S ratings for each method were the within-subjects variables. In order to assure that applying this statistical method made sense, we first checked that the principle of sphericity was not violated by applying the W Mauchly's test ($W = 0{,}838; p = 0{,}142$) [33]. The results showed that MDD produced the highest S ($M = 4{,}76; SD = 0{,}73$), followed by code-centric ($M = 4{,}17; SD = 0,72$) and then MBD ($M = 3{,}48; SD = 0{,}96$). The results also showed that the interaction Mod*Meth is not significant ($F = 1{,}768; p > 0{,}05$). If we examine the effects of the two independent variables (module and method) we can observe

how the Mod inter-subject influence did not attain significance ($F = 0{,}167; p > 0{,}05$), while the main effect of method did reach significance, ($F = 18{,}04; p < 0{,}01$), that is, the differences in S are significantly affected by the method used, regardless of the particular module being developed. The last step of the analysis consisted on studying the pairwise differences among methodologies through a one-way Anova with pairwise comparisons. In order not to augment the risk of a type-1 error, a Bonferroni adjustment was applied. This means reducing the significance threshold to 0,0167 ($p = 0{,}05/3 = 0,0167$). Even with this conservative adjustment, all the pairwise S differences were significant ($p < 0{,}05$), which means that, in our experiment, subjects rated significantly differently the three approaches, being MDD the best method rated and MBD the worst.

3.5 Threats to Validity

The analysis of the threats to validity evaluates under which conditions our experiment is applicable and offers benefits, and under which circumstances it might fail. For the classification of these threats, we have followed the classification proposed by Cook and Cambell [34]: internal, external, construction and conclusion.

Threats to conclusion validity refer to the relationship between the treatment and the outcome. In order to minimize the threats, we have strived to automatically capture as many measures as possible, with the help of well-known tracking systems such as JIRA or SVN. Additionally, statistical tests have been chosen conservatively, without making any kind of assumption on variable distributions. However, the fact that the students self-reported the measures, together with the duration of the experiment (six weeks) and the low number of subjects hamper the conclusion validity.

Threats to internal validity are concerned with the possibility of hidden factors that may compromise the conclusion that it is indeed the treatment what causes the differences in outcome. All groups applied all the treatments to different modules at different times, what minimizes many internal threats such as selection, history, maturation or social

threads such as compensatory rivalry or resentful demoralization. However, being an intra-subject design, carry-over effects may have occurred.

Threats to construct validity refer to the relationship between theory and observation. In this sense, both the treatments and the measures used to assess productivity and satisfaction have been previously widely used in literature. This notwithstanding, there remains the possibility of an interaction of testing and treatments: the need to self-report certain measures may have changed the behavior of the students. We believe that the fact that the experiment took over six weeks minimizes this risk, since it is very difficult to maintain a "potentially abnormal" behavior over such a long period of time without it being detected.

Also, the hypothesis of the experiment (that is, a higher productivity of MDD environments) was quite easy to guess, so students may have felt bound to report less time when using MBD or MDD. Anyway, the experiment observers took special care not to disclose this hypothesis to the students. Additionally, the experiment suffers from a restricted generalizability across constructs: we have checked a positive outcome between productivity and MDD, but we cannot assure that this does not come at the expense of other characteristic of the developed software, such as modularity, reusability, or any other quality attribute.

Last but not least, external validity is concerned with generalization of the results. In this group of threats we have identified a lack of sample representativeness (M.Sc. students), academic environment, a strict architecture and a restricted domain and complexity. Also, we are conscious of the existing coupling between method and tool: all the methods were accompanied by tools. Although we tried to choose well-known development environment and -when possible- use standards (e.g. UML for the modeling activity in MBD and MDD), we are conscious that the different tools add a different "flavor" to the methods. Therefore, this experiment needs to be replicated in order to make sure that it is the method used and not the tool what causes the observed differences.

4 Conclusions

During the last years the Web Engineering community has claimed how the use of modeling practices in MDD and MDB approaches significantly improves the productivity and satisfaction of Web applications with respect to code-centric development approaches. However, up to now, the quality and quantity of the empirical analysis that demonstrate the true impact of these modeling techniques over the final developer's productivity or satisfaction are still very low. In this paper, we have presented a rigorous analysis of a quasi-experiment carried out in a controlled environment. The data gathered shows that the productivity and the satisfaction of junior Web developers are significantly affected by the development method but they are independent from the particular module being developed. The main conclusions of our study (that still need to be corroborated with further replications) are:

1. The MDD approach seems to significantly increase the productivity of developers with respect to both the MDB and the code-centric approach.
2. The MDD approach satisfies the most the expectations of juniors developers, followed by code-centric and, in third position, the MDB approach.

These results are well aligned with with the assumption that model-driven techniques improve the productivity and also the satisfaction among developers when they are accompanied by a generation environment. However, the productivity and the satisfaction can decrease even below code-centric practices when the modeling activities are used exclusively as blueprints to improve the understanding, and the developers must implement manually almost all the final code.

Our study of the impact of MDD on the productivity and satisfaction is just the beginning of a family of experiments in which we want to replicate the same analysis with practitioners in industry and also with more complex Web application client-side models. Moreover, further experimentation is needed to separate the effect of methods from the effect of their accompanying development environments (Visual Studio 2010, RSM or the

OOH4RIA tool) and to be able to generalize the results to a different population, different methods and languages, different application types or different application sizes.

Acknowledgements

This paper has been co-supported by the DLSI, the Spanish Ministry of Education, and the University of Alicante under contracts TIN2010-15789 (SONRIA) and GRE10-23 (DISEMRIA). The authors also wish to thank their students to take the time to participate in this empirical study. Besides we would like to thank to Jose Javier Martinez and Juan Antonio Osuna, who contributed to the development of the OOH4RIA Tool.

References

1. CMU/SEI: CMMI Product Development Team, CMMI for Development version 1.2 (2006)

2. Moore, G.C., Benbasat, I.: Development of an instrument to measure the perceptions of adopting an information technology innovation. Information Systems Research 2(3) (1991) 192-222

3. Fowler, M.: UML distilled: a brief guide to the standard object modeling language. 3rd editio edn. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2004)

4. Kleppe, A.G., Warmer, J., Bast, W.: MDA explained: the model driven architecture: practice and promise. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2003)

5. Bruckhaus, T., Madhavii, N.H., Janssen, I., Henshaw, J.: The impact of tools on software productivity. Software, IEEE 13(5) (2002) 29-38

6. Genero, M., Manso, M.E., Visaggio, A., Canfora, G., Piattini, M.: Building measure-based prediction models for UML class diagram maintainability. Empirical Software Engineering 12(5) (2007) 517-549

7. Abrahão, S., Iborra, E., Vanderdonckt, J.: Usability evaluation of user interfaces generated with a model-driven architecture tool. Maturing Usability (2008) 3-32

8. Mellegård, N., Staron, M.: Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment. Product-Focused Software Process Improvement (2010) 336-350

9. Mellor, S.J., Clark, T., Futagami, T.: Model-driven development: guest editors' introduction. IEEE software 20(5) (2003) 14-18

10. Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on, IEEE (2009) 113-120

11. Mohagheghi, P.: An Approach for Empirical Evaluation of Model-Driven Engineering in Multiple Dimensions. In: C2M:EEMDD 2010 workshop- from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, CEA LIST Publication (2010) 6-17

12. Kitchenham, B., Budgen, D., Brereton, P., Turner, M., Charters, S., Linkman, S.: Large-scale software engineering questions-expert opinion or empirical evidence? Software, IET 1(5) (2007) 161-171

13. Wohlin, C., Runeson, P., Höst, M.: Experimentation in software engineering: an introduction. Springer Netherlands (2000) 14. Zelkowitz, M.V.: An update to experimental models for validating computer technology. Journal of Systems and Software 82(3) (2009) 373-376

15. Mohagheghi, P., Dehlen, V.: Where is the proof? - A review of experiences from applying MDE in industry. In: European Conference on Model Driven Architecture-Foundations and Applications (ECMDA 2008), Springer (2008) 432-443

16. Abrahão, S., Poels, G.: A family of experiments to evaluate a functional size measurement procedure for Web applications. Journal of Systems and Software 82(2) (2009) 253-269

17. Afonso, M., Vogel, R., Teixeira, J.: From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company. (2006)

18. Krogmann, K., Becker, S.: A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor. Software Engineering (2007) 169-176

19. Staron, M.: Transitioning from code-centric to model-driven industrial projects-empirical studies in industry and academia. Model Driven Software Development: Integrating Quality Assurance (2008)

20. Kapteijns, T., Jansen, S., Brinkkemper, S., Houët, H., Barendse, R.: A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. From code centric to model centric software engineering: Practices, Implications and ROI (2009) 22

21. Mellegård, N., Staron, M.: Distribution of Effort Among Software Development Artefacts: An Initial Case Study. Enterprise, Business-Process and Information Systems Modeling (2010) 234-246

22. Panach, J.: Incorporación de mecanismos de usabilidad en un entorno de producción de software dirigido por modelos. Tesis doctotal, Universidad Politécnica de Valencia (2010)

23. Kampenes, V., Dyba, T., Hannay, J., Ksjoberg, D.: A systematic review of quasiexperiments in software engineering. Information and Software Technology 51(1) (Jan 2009) 71-82

24. Perry, D.E., Porter, A.A., Votta, L.G.: Empirical studies of software engineering: a roadmap. In: Proceedings of the conference on The future of Software engineering, ACM (2000) 345-355

25. Ambler, S.: Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley (2002) 26. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)

27. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: Architectural and technological variability in rich internet applications. IEEE Internet Computing 14(3) (2010) 24-32

28. Montgomery, D.C.: Design and analysis of experiments. John Wiley & Sons Inc (2008)

29. Plonsky, M.: Psychological Statistics (2009)

30. Gollapudi, K.: Function points or lines of code?-an insight. Global Microsoft Business Unit, Wipro Technologies (2004)

31. Seato: Counting Lines of Code in *C#* (2004)

32. SPSS Inc. an IBM Company Headquarters: PASW Statistics 18 - Content Guide (2009)

33. Mauchly, J.W.: Significance test for sphericity of a normal n-variate distribution. The Annals of Mathematical Statistics 11(2) (1940) 204-209

34. Cook, T.D., Campbell, D.T., Day, A.: Quasi-experimentation: Design & analysis issues for field settings. Houghton Mifflin Boston (1979)

.

CAPÍTULO 5
MDD VS. TRADITIONAL SOFTWARE DEVELOPMENT: A PRACTITIONER'S SUBJECTIVE PERSPECTIVE

El contenido de este capítulo corresponde con el siguiente artículo: Martínez, Y., Cachero, C. and Meliá, S. - MDD vs.Traditional Software Development: a practitioner's subjective perspective. Information and Software Technology. Published by Elsevier B.V. http://dx.doi.org/10.1016/j.infsof.2012.07.004

Information and Software Technology es la revista de archivo internacional que se concentra en investigación y la experiencia que contribuye a la mejora práctica del desarrollo de *software*. El alcance de la revista incluye los métodos y las técnica para mejorar la ingeniería de *software* y dirigir su desarrollo. La revista apoya y da la bienvenida propuestas de evaluaciones sistemáticas dentro del alcance de la revista. *Information and Software Technology* tienen un editor que se especializa en evaluaciones sistemáticas.

Esta revista está indizada en el JCR (*Journal Citation Reports*) 2011, con un factor de impacto de 1.250, de acuerdo con el índice de cita de *Thomson's Science Citation Index* (http://www.isiwebofknowledge.com/) -*ranking* 29/104, Cuartil 2, categoría *Computer Science/Software Engineering*-. Además, está incluida en la CIRC del CSIC como revista de categoría EX.

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

MDD vs. traditional software development: A practitioner's subjective perspective

Yulkeidi Martínez[a,*], Cristina Cachero[b], and Santiago Meliá[b]

[a] Universidad Máximo Gómez Báez de Ciego de Ávila, Cuba

[b] DLSI. Universidad de Alicante, Spain.

ARTICLE INFO

Article history:
Received 9 November 2011
Received in revised form 5 July 2012
Accepted 9 July 2012
Available online xxxx

Keywords:
MDD
MBD
Code-centric development
Experiment
Usefulness
Ease of use

ABSTRACT

Context: Today's project managers have a myriad of methods to choose from for the development of software applications. However, they lack empirical data about the character of these methods in terms of usefulness, ease of use or compatibility, all of these being relevant variables to assess the developer's intention to use them.
Objective: To compare three methods, each following a different paradigm (Model-Driven, Model-Based and Code-Centric) with respect to their adoption potential by junior software developers engaged in the development of the business layer of a Web 2,0 application.
Method: We have conducted a quasi-experiment with 26 graduate students of the University of Alicante. The application developed was a Social Network, which was organized around a fixed set of modules. Three of them, similar in complexity, were used for the experiment. Subjects were asked to use a different method for each module, and then to answer a questionnaire that gathered their perceptions during such use.
Results: The results show that the Model-Driven method is regarded as the most useful, although it is also considered the least compatible with previous developers' experiences. They also show that junior software developers feel comfortable with the use of models, and that they are likely to use them if the models are accompanied by a Model-Driven development environment.
Conclusions: Despite their relatively low level of compatibility, Model-Driven development methods seem to show a great potential for adoption. That said, however, further experimentation is needed to make it possible to generalize the results to a different population, different methods, other languages and tools, different domains or different application sizes.

*Corresponding author. Address: Universidad Máximo Gómez Baje de Ciego de Ávila, Carretera a Moron km 9 1/2, C.P. 67800, Ciego de Ávila, Cuba. Tel.: +53 33 217010; fax: +53 33 266365.
E-mail addresses: yulkeidi@gmail.com (Y. Martinez), ccachero@dlsi.ua.es (C. Cachero), santi@dlsi.ua.es (S. Meliá).
0950-5849/$ - see front matter © 2012 Published by Elsevier B.V. http://dx.doi.org/10.1016/j.infsof.2012.07.004

1. Introduction

It is a well-known fact that the Software Engineering (SE) community has for a long time been advocating the use of models in the quest to improve software development practices [1]. The reason for this support is twofold: First of all, models increase the level of abstraction at which problems are tackled, and therefore help software developers to manage software complexity [2]. In addition, the bandwidth of information visual representations is potentially higher than that of information presented in media reaching any of the other senses [3]. Among the SE community, the Unified Modeling Language (UML) is well established as the standard modelling language. UML does not promote any particular development process, however, and modelling practices may differ greatly from organization to organization. One well-known way of classifying such modelling practices is according to the extent to which modelling is used to support the development process. In this sense, Fowler [4] describes three different modes in which modelling languages (and UML in particular) can be used: sketch, blueprint and programming language.

- Sketches are informal diagrams used to communicate ideas. They usually focus on a particular aspect of the system and are not intended to show every detail of it. This is the most common use of UML, and is the recommended practice in agile, code-centric frameworks like Scrum [5]. When models are used as sketches, tools are rarely used, as the modelling activity is performed mostly in front of whiteboards, where designers join to discuss complex or unclear aspects of the system.
- Blueprints are diagrams that show most of the details of a system, in order to foster an understanding of it, or to provide views of the code in a graphical form. Blueprints are widely used in Model-Based Development (MBD), which is at the core of standard development practices like the ones promoted by the Rational Unified Process (RUP) framework [6].
- Last but not least, fully-fledged models can be used to characterize the application completely. If such is the case, the diagrams may even replace the code, and be transformed

automatically into executable binaries. This is the modelling use that lies at the core of the Model-Driven Development (MDD) paradigm.

This classification has led some authors to characterize the modelling maturity level of organizations on the basis of the role of modelling in their software development process [7]. While code-centric development approaches do not ask for much more than an informal use of modelling techniques and languages, both MBD and MDD approaches require a more formal use of models. Furthermore, MDD approaches rely on models that need to be syntactically correct, but also semantically accurate, consistent with each other and complete, so that they can be used as input for model transformations [8]. Table 5–1 goes into the main differences between these three ways of conceiving the development cycle, and compares the three paradigms on a discipline (workflow) basis (Modeling, Implementation and Testing), with respect to their reliance on models to help carry out the activities involved. The three disciplines used for comparison are the ones that typically appear in an Agile UP development process during the construction phase [9].

Tabla 5–1: Correspondence between Agile UP disciplines and development paradigms.

Discipline	Code-centric	Model-based	Model-driven
Modeling	Sketch or absent	Blueprint	Fully-fledged (DSL)
Implementation	Manual	Semiautomatic	Automatic or semiautomatic
Testing	Manual	Manual	Semiautomatic or manual

As mentioned previously, code-centric approaches do not usually rely on Computer Aided Software Engineering (CASE) tools to represent the modelling discipline. It is rather the case that they usually promote the use of, at most, whiteboard sketching that allows the main ideas among different team members to be communicated speedily. The implementation and testing disciplines are, moreover, usually tackled manually.

When the Model-Based paradigm is followed, its modelling discipline generates a blueprint, usually with the help of a UML tool. These UML tools normally enable a partial implementation (usually plain classes with attributes and empty methods) to be obtained. However, code testing is usually carried out manually. Finally, the Model-Driven

paradigm typically relies on fully-fledged models that are usually supported by a Domain-Specific Language (DSL). This paradigm requires modelling to be carried out by means of a Model-Driven CASE tool that makes it possible to generate the final implementation, either partially (semiautomatically) or in full (automatically). The corresponding tests can be partially inferred from models or manually implemented, depending on the development environment.

As for the CASE tools available for MBD and MDD methods, and depending on the particular paradigm for which they were devised, they may offer not only modelling environments, which assure the syntax correctness of the models, but also model checking and partial or complete software generation capabilities. Well-known MBD environments include MagicDraw [10], Visual Paradigm [11], or Rational Software Architect (RSA) [12] and its precursor, Rational Software Modeller (RSM). Although all of them offer certain code generation capabilities, they are not able to generate fully-fledged applications, and most MBD approaches do not rely on them for the coding phase. As regards MDD, some of the best-known tools which are currently available are TrueView Domain Modeller [13], Borland Together [14], OOH4RIA IDE [15], and WebRatio [16], although more tools claiming to provide powerful modelling environments and code-generation capabilities are appearing by the day.

Behind all these efforts, there is the underlying assumption that using development methods that rely on models and tools with code generation capabilities improves the overall experience of the developer for when he/she is developing applications. This, in turn, improves the developer's intention to adopt the method, particularly as systems become larger and more complex [17, 18]. The first assumption has been reliably supported by different studies that show the benefits of MDD in comparison both to MBD approaches and traditional, code-centric approaches [19,20]. Among these advantages we can cite lower entire product life cycle and maintenance costs, shorter time-to-market and fewer human resources, short and long term productivity gains, software quality improvements or defect and rework reduction [21-23]. In addition, these advantages are justified in literature by

the higher level of compatibility between systems, the simplified design process, and the better communication between individuals and teams working on the system that the MDD paradigm fosters [24, 25].

The second assumption, that is, that the benefits of MDD and the intention to adopt the paradigm are strongly and positively correlated, is justified by well-known theoretical models of technology adoption. Many of these models have been tailored to match the idiosyncrasy of software development methods [26]. The vast majority of these models assume that objective measures significantly and strongly affect subjective perceptions, which in turn shape the developer's adoption intentions [27]. This assumption runs counter to some facts that have been repeatedly demonstrated in other research areas; i.e., that subjective measures are not only not strongly correlated but often in disagreement with their objective counterparts [28-30]. If this was also the case for SE method adoption, that would explain why, in spite of the proven cost and quality advantages of MDD, the purported paradigm shift from pure code-centric approaches to MDD, which has been expected in the industry for years, is still to come [31, 32].

In the quest for a confirmation/refutation of this hypothesis, theories from social psychology can help. In this discipline, a number of empirical studies show that actual future behaviour (adoption in practice) may be reliably predicted from intentions to engage in the behaviour [33]. This means that the assessment of the intention to use a given methodology is a good predictor of the future actual usage of such methodology. They also suggest that the intention to engage in a behaviour (such as adopting a software development method) is better assessed through perceived, subjective characteristics (e.g. perceived usefulness) than through primary characteristics of the method (e.g. actual efficacy) [34]. The reason is that, since different adopters may perceive those primary characteristics differently, their eventual behaviour (including their intention to use the innovation) might differ. This being the case, given any primary characteristic of a method (such as the mean performance of the developer when using it), behaviour of individuals (developers) is better predicated by how they perceive those primary characteristics (e.g.

how fast they feel they can produce code with the method) than by the objective measure of the variable (their actual productivity).

In a nutshell, we agree with [32] that the low level of adoption of MDD approaches may be partly due to the fact that method assessment efforts still revolve mostly around method technological features (such as separation of concerns, the availability of tools or artefact traceability, to name just a few) while paying little attention to the developers' attitudes and perceptions of the method. It is therefore of paramount importance to the discipline to go into greater depth in the study of such perceptions and the variables that actually impact them (beyond the primary characteristics of the method). By doing so, we would be able to define methods and techniques, that showed a greater likelihood to make an impact on software development practices in industry.

To advance on this path, in this paper we present an empirical study based on a tailored method adoption model, which studies the variables that may impact the actual use of methods. It does this using three representatives of the three aforementioned paradigms (code-centric, MBD and MDD). In contrast to other studies, the intra-subject design used in this study favours the detection of perceived advantages-disadvantages of each approach with respect to the others. The fact that, after the experiment, the users had to choose one of the three methods to work with during the rest of the project imitates the decision process that developers and project managers take in the workplace on a daily basis.

The paper is structured according to the reporting guidelines for controlled experiments in SE [35], as follows: Section 2 presents a set of theories that may help to explain the software method adoption process. These theories constitute the theoretical framework of our experiment, and they set the context for the definition of goals, hypotheses and variables in Section 3, together with the experimental design (subjects, instrumentation, operation and data collection mechanisms) and the threats to validity of the experiment. Section 4 presents research that is complementary to our work. Finally, Section 5 concludes the paper and outlines some further lines of work.

2. Theoretical model

As mentioned previously, there exist a number of models that establish the determinants of user technology acceptance. Among them, TAM [36], TAM2 [37], PCI [34], TPB [38], and MPCU [39] stand out, due to their theoretical foundations.

In [40], the authors present a comparative study of how well these models predict the intention to adopt a given method. This comparison proves that some of the dimensions defined in those models are useful to predict method adoption (namely Compatibility, Subjective norm and Voluntariness), while others, given the differences between adopting a method and a tool, can be dropped (e.g. Career Consequences, Perceived behavioural control, etc.). This study also dismisses the impact of Ease of Use on method adoption. That is surprising, since Ease of Use is one of the main components of the TAM model and is regarded as being closely correlated with intention of use in other well-known theoretical models of method adoption [27,32,41,42]. Given the controversy of this dimension, we have decided to keep Ease of Use as a potential explainer of variability in method adoption, and to test it with the empirical data. Another controversial dimension is tool performance (also called Perceived Tool Maturity), which is included in [32,41,42] as a potentially relevant variable to explain both actual use of MDD methods and future intention to use them. This dimension has not been found significant in any of these studies, however, nor has it been considered relevant in any of the seminal theoretical models studied. For this reason, we have left it out of our theoretical model.

In short, our theoretical model includes the following explanatory variables:

- Perceived Usefulness (PU): the extent to which a developer believes that using the method will enhance his or her job performance. The more useful a method is regarded by developers, the more likely they are to form intentions to use it.
- Perceived Ease of Use (PEU): the degree to which a developer believes that using a particular method will be free of effort. The easier developers consider the method to be, the more likely they are to adopt it.

- Compatibility (PC): the degree to which the method is perceived as being consistent with the existing values, needs and past experiences of potential adopters. The more compatible a method is with respect to how developers perform their work, the more likely they are to form intentions to use it.
- Subjective Norm (SN): the degree to which a developer believes that others who are important to them are of the opinion that they should perform the behavior. The more developers believe that others who are important to them consider that they should use the method, the more likely they are to form intentions to use it.
- Voluntariness (V): the extent to which potential adopters perceive the adoption decision to be non-mandatory. Since adopting a new development method requires a huge mental effort, the theoretical model establishes that the more voluntary the adoption decision is seen to be by developers, the less likely they are to adopt it.

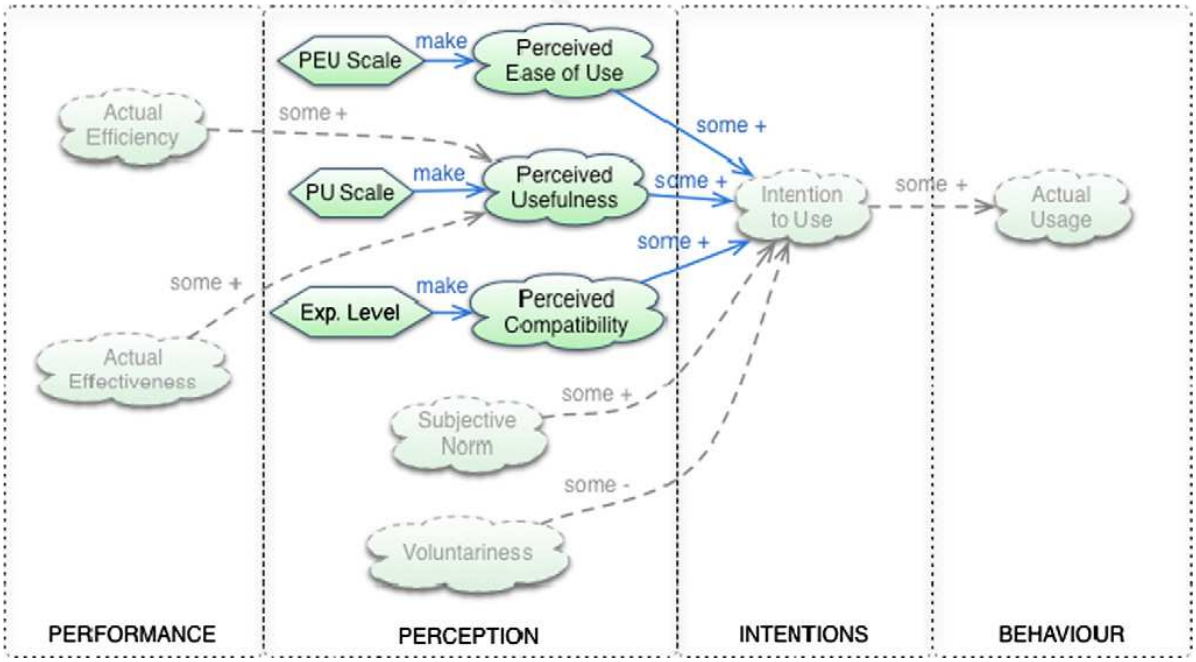


Figura 5–1: Synthesized theoretical model of method adoption.

Fig. 5–1 depicts the components of the resulting theoretical model. In this figure, the model components (clouds), the measures (hexagons) and influences (arrows) actually put to test by our experiment (PEU, PU and PC) have been stressed with solid colours and lines, while the parts of the model that have been left out of our experiment are marked with shaded colours and dashed lines. In particular, SN and V have remained constant

throughout the methods and tools used. We will go into this issue further in Section 3. Fig. 5–1 also reflects the direction of the effect: PEU, PU, PC and SN are assumed to affect (some +) the intention to use a method positively, while Voluntariness is assumed to affect (some -) such adoption intention negatively.

3. Description of the experiment

During the months of January and February of 2011, a controlled experiment was conducted at the University of Alicante. A controlled experiment in SE is a randomized or quasi-experiment, in which individuals or teams (the study units) engage in one or more tasks for the sake of comparing different processes, methods, techniques, languages or tools (the treatments) [43].

The goal of our experiment was to evaluate the method adoption intentions of subjects after developing the business layer of a Web application using three different approaches: *code-centric* (based on *C#* and the .NET framework), MBD (based on UML and supported by the Rational Software Modeler tool) and MDD (based on the OOH4RIA approach and supported by the OOH4RIA IDE tool). We are conscious that the choice of specific tools for each method diminishes the external validity, that is, the generalizability of the results. Such a selection was necessary, nevertheless in order to replicate the actual environment in which each method is usually applied and thereby increase the construct validity of the experiment. This issue will be further discussed in Section 3.4.

In order to fulfil our research goal better, the tools and environments were chosen on the basis of their widespread adoption among practitioners, whenever possible. To be specific, the coding activities were performed with Visual Studio 2010, while the MBD modeling activities were performed with Rational Software Modeler. Both IDE's are currently being used by a large community of software practitioners. On the one hand, the Microsoft Visual Studio 2010 Professional IDE makes it possible to program, debug and test multi-device applications in the Microsoft .NET platform. On the other hand, the IBM Rational Software for modeling applications following the UML 2.0 standard [44]; it also

introduces some extensions to generate some parts of applications for different platforms such as Java and .NET.

With respect to the MDD paradigm, there are two tool mainstreams:

- Some proposals opt to apply the UML profiling mechanism to extend the different UML elements by defining constraints and adding tag values, so as to introduce domain-specific semantics. In this category, there is a large set of tools that enables code to be generated. However, only two of these tools can generate an NHibernate-based object-oriented business logic, as well as persistence layers. These are Modelio [45] and Visual Paradigm [45], both of them are commercial, and they do not provide academic licenses.
- A second set of proposals, which has been gaining momentum over recent years, defines Domain-Specific Languages (DSLs), that is, relatively simple languages (graphical or textual) that are created to model some problem domain. To do that, each DSL requires the definition of a MOF metamodel (including OCL constraints) and a graphical or textual language to represent its concrete syntax. Although there are several DSL approaches capable of generating NHibernate-based business logic and persistence layers, (e.g., Integranova, RadarC, etc.), here also it is the case that they do not provide any kind of academic license.

Given the fact that none of the aforementioned proposals is standard or even widely adopted, and due to the lack of academic licenses for any of the commercial tools, we opted to use OOH4RIA. OOH4RIA belongs to the second group of proposals defining a set of DSL's for modelling different aspects of Rich Web Applications. OOH4RIA is free of charge for universities, and it is taught as part of the Master's course in which the experiment took place.

3.1. Goals and context definition

Following the GQM template, our empirical study is aimed at *Analysing* a code-centric, an MBD and an MDD approach *for the purpose* of evaluating *with respect* to its

adoption intentions *from the point of view* of junior software developers. The context of the study was a set of M.Sc. students developing the business layer of a Web application.

The design of this experiment is based on the framework for experimentation in SE research, as suggested in [46]. Also, a laboratory package has been compiled for the sake of replicability [47]. This study is based on the theoretical method adoption model presented in Section 2, and compares three methods (each one supported by specific tools) one representative of the *code-centric* paradigm, one representative of the MBD paradigm and one representative of the MDD paradigm.

The research questions addressed in this study were formulated as follows:

RQ1: Is the developer's perceived usefulness of the method significantly different between the methods, regardless of the particular application?

RQ2: Is the developer's perceived ease of use of the method significantly different between the methods, regardless of the particular application?

RQ3: Is the developer's compatibility with the method significantly different between the methods?

RQ4: Are the perceived usefulness, perceived ease of use and compatibility measures correlated?

RQ5: Do the developers' adoption intentions significantly differ between the three approaches (*Code-centric*, Model-based, Model-driven), regardless of the particular application being developed?

RQ6: What are the main perceived advantages/drawbacks of each approach?

The first four research questions were devised to be answered by quantitative means, while the fifth research question was exploratory (no formal scale for measuring the Intention to Adopt (IA) was devised) and the sixth one, qualitative in nature, was aimed at gathering some possible explanations for the quantitative results, giving additional evidence of the degree of completeness of the theoretical model.

3.1.1. Subjects and application

The subjects were 30 students of the "Web Applications Developer" Master at the University of Alicante. These students were divided into six groups of 4-6 people. From these, one group dropped out of the experiment, due to two of their components abandoning the Master's course for work reasons. The final set of observations thus corresponds to the observations of the remaining five groups (26 subjects). Since this abandoning of the experiment had nothing to do with the treatments that the group was applying to their project, nor had it to do with the particular order in which the team was applying them, we can assume that the results of the experiments have not been compromised.

The final sample was made up of 25 men and 1 woman, of whom 75 % had more than 2 years of experience in developing web applications. The mean age of the participants was 25,6 years old and all of them were Computer Engineering graduates of the University of Alicante.

Regarding the subjects' level of knowledge with respect to the different technologies and methods used during the experiment, a questionnaire showed that 81 % knew UML, and that another 12 % considered that they had a high-level of knowledge of UML. It should also be noted that 76 % of the subjects had previously programmed with Visual Studio (VS) and .NET during their degree course, although only 12 % had applied it in industry. Finally, the subjects acknowledged no previous practical knowledge of MDD, although 56 % of them were aware of the existence of the paradigm.

By the time the experiment took place, the subjects had received additional training in all three methods. Such training consisted in 30 h of training in programming in *C#* using Visual Studio 2010, 20 h of training in UML modelling with RSM and 10 h of training in modelling with the OOH4RIA tool. The reason for these training divergences is that each method training relies on concepts introduced previously, and so students could grasp the basics more quickly.

Each group developed a social network for a different domain:

- *Trips*: the social network for this domain is focused on establishing relationships between people who want to reduce travel costs by sharing their own cars.
- *Events*: the social network for this domain is centered on organized social events.
- *Hospitals*: the social network for this domain aims at improving the communication between physicians and patients.
- *Academics*: the social network for this domain focuses on connecting and sharing teaching contents among teachers and students.
- *Facework*: the social network for this domain helps workers to share information about different tasks, resources and goals of the company.

All the applications shared the application type (a social network) and the complexity (which was controlled by dividing the applications into modules and, for each one, defining a range of functional requirements that all the applications had to support, regardless of the domain). From these, the three modules whose development was monitored in our experiment were:

- Support for the establishment of a community of users (Groups), to create contents and relationships among people of different environments (professional, personal, etc., depending on the particular application being developed).
- Support for the organization of events (Events) where people can invite their friends or colleagues to attend a place where the event is held (the particular event being a celebration, a work meeting, etc. depending on the specific application being developed)
- Support for an organizational section community (Organization) where subjects (be they companies, celebrities, etc., depending on the particular application), can publish content, photos, etc., in a unidirectional way, to the social network.

For six consecutive weeks, the teams were asked to implement each module following a different method. The time assigned for the implementation of each module was 2 weeks. The order in which teams applied each method was randomized to avoid order effects.

Such randomization can be seen in Table 5–2. All the groups were co-located in the Master classroom for 30 h (10 h per method), and performed the rest of the work at home. Since the applications were all different, there was no need for interaction between the groups. After the experiment, the subjects were asked to give their individual opinions regarding each methodology. They were requested to choose the approach they preferred most, out of the three that they had used during the experiment, so they could use that one to develop the remaining modules of the project.

Tabla 5–2: Diseño experimental: diseño factorial e intra-sujetos.

Dominio	**Module**		
	Group	Events	Organization
Trips	*code-centric*	MBD	MDE
Events	*code-centric*	MDE	MBD
Hospital	MBD	MDE	*code-centric*
Academics	MDE	MBD	*code-centric*
Facework	MBD	*code-centric*	MDE

As mentioned previously, to develop the different projects the students had to follow the Agile Unified Process (Agile UP) methodology [9], a streamlined approach to software development that is based on the IBM's Rational Unified Process (RUP). The Agile UP lifecycle is serial in the large, iterative in the small, and delivers incremental releases over time. Our particular experiment was situated in the construction phase of Agile UP, which focuses on developing the system to the point where it is ready for pre-production testing. The construction phase is made up of a set of disciplines or workflows that groups different tasks of this process. These disciplines were presented in Table 5–1, together with the impact of modeling practices on each of them, depending on the particular paradigm.

3.1.2. Implementation language and CASE tools

As commented above, the development environment for the experiment was set up as follows:

- Development framework: .NET framework, Silverlight 4.0 and NHibernate (Object-Relational Mapping).

- Coding IDE: Visual Studio 2010
- Modeling tools: RSM and OOH4RIA
- Code Generation tool: OOH4RIA
- Languages: *C#* and XMLMapping (ORM mapping of NHibernate).
- Other tools: The set of questionnaires filled-in by each developer was published online.

The *code-centric* treatment relied solely on the coding tools provided by the Visual Studio 2010 environment. The MBD treatment also required the students to work with RSM. Lastly, for the MDD treatment, the students worked with the OOH4RIA IDE. Both RSM and the OOH4RIA IDE are based on the Eclipse Modeling Project [48]. Eclipse is an open source software, whose main purpose is to provide highly-integrated platform tools [49]. According to [50], Eclipse has contributed to the successful implementation of Model Driven Architecture (MDA, which is the OMG standard for MDD [51]), by providing an open source platform and a whole implementation of the MDA specifications.

To standardize the code that was to be developed, the subjects had to implement/generate four specific files for each module (Group, Events, Organization): the Business Objects file (BO), the Data Access Object file (DAO), the Data Transfer Object file (DTO) and the Database file (DB).

3.2. Experiment planning

As shown in Fig. 5–1, the idiosyncrasy of the experiment made some of the dimensions of our initial theoretical model non-relevant; namely, Voluntariness and Subjective Norm, which did not apply to our course environment. The three paradigms were equally valued and no obligation whatsoever was made about the method that the students had to use once the experiment was finished. This lack of relevance has been outlined in Fig. 5–1, by showing the corresponding dimensions in a lighter shadow of grey.

Also, given the fact that the subjects were not randomly chosen but selected rather on the basis of their attendance to a Master's degree, our study belongs to the quasi-experiment category. Quasi-experiments, although suffering from a lower internal validity

than true experiments, are widely used and deemed useful in the Empirical SE field, since they allow investigations of cause-effect relations in settings such as ours, in which randomization is too costly [43].

Given the evidence gathered by our theoretical model and the research questions presented in Section 2, we have defined the following Independent (experimentally manipulated) Variables (IV) or factors:

- Meth: Method, a categorical variable with three levels: code-centric, MBD, MDD. It is important to note that, in this experiment, when we refer to method, we are in fact talking about a compound variable (method*tool), due to the coupling of these two variables in our experimental settings.
- App: Application domain, a categorical variable with five possible values: Trips, Hospitals, Events, Academics, Facework.

The dependent (measurable) variables (DV) are:

- PU: Perceived usefulness of each method: an interval measure based on a 7−point Likert scale.
- PEU: Perceived ease of use of each method: an interval measure based on a 7−point Likert scale.
- PC: Compatibility of each method: an interval measure based on a 3−point Likert scale.
- IA: Intention to Adopt a given method: a nominal measure with three possible values: *code-centric*, MBD, MDD.

We defined the following testable hypotheses, also based on the research questions:

- Perceived Usefulness (RQ1):
  - HPU_0: PU(MDD) = PU(MBD) = PU(code-centric). Perceived usefulness of code-centric, MBD and MDD methods do not significantly differ. This fact holds, regardless of the actual application being developed.
  - HPU_A: PU(MDD) > PU(MBD) > PU(code-centric).

- Perceived Ease of Use (RQ2):
  - $HPEU_0$: PEU(MDD) = PEU(MBD) = PEU(code-centric). Perceived ease of use of code-centric, MBD and MDD methods do not significantly differ. This fact holds, regardless of the actual application being developed.
  - $HPEU_A$: PEU(MDD) < PEU(MBD) < PEU(code-centric).
- Perceived Compatibility (RQ3):
  - HPC_0: PC(MDD) = PC(MBD) = PC(code-centric). Perceived Compatibility of code-centric, MBD and MDD methods do not significantly differ.
  - HPC_A: PC(MDD) < PC(MBD) < PC(code-centric).

In order to test the hypotheses, we defined three questionnaires to measure the PU, PEU and PC dependent variables, respectively.

- The perceived usefulness PU(Meth) was assessed through a 7-point Likert scale that consisted of four items: subjective developer's throughput (with respect to an expert), subjective developer's efficiency, subjective utility of the method and subjective reliability of the results, obtained from applying the method.
- The perceived ease of use PEU(Meth) was assessed through a semantic-differential scale that required developers to judge the development method on seven pairs of adjectives describing their experience. Four adjectives were formulated in positive and four in negative to control a possible acquiescence bias. Developers could modulate their evaluation on 7-points (after re-coding of reversed items 1 = very negative, 7 = very positive).
- To measure PC (Meth), we defined a two-item scale, made up of a 3-point rating of familiarity (1 = low, 2 = medium, 3 = high), and a 3-point level of previous experience (1 = low, 2 = medium, 3 = high) with the techniques and tools involved in code-based, MBD and MDD development, respectively.

Since ANOVA presents a high degree of robustness with respect to both ordinality and non-normality of the scale [52], all the PEU, PU and PC questionnaires have been treated as interval measures for the refutation of the hypotheses involving the comparison of methods. This is also consistent with the general opinion that Likert scales that are defined

by means of sufficient (typically 7 or 9−point) symmetric and equidistant Likert items approximate an interval-level measurement. Furthermore, treating the scale as interval can be beneficial, because otherwise some valuable information (the "distance" between opinions) could be lost [52]. However, since the robustness of correlations with respect to violations of the non-normality of distributions is not that straight-forward, they have been treated as ordinal for the analysis of correlations among dependent variables.

As far as the perceived advantages/disadvantages of the different methods are concerned, these were gathered through two open questions, one asking about the three main advantages perceived, and one asking about the three main disadvantages perceived. The questionnaire containing all these questions is included in the replication package [47].

Last of all, the IA DV was measured indirectly through a decision, made by the teams, on which method to use for the rest of the project, once the experiment was finished.

3.3. Data analysis and interpretation of results

The statistical analysis was carried out with the PASW (Predictive Analytics SoftWare) Statistics software [53].

Prior to the assessment of the hypotheses, we checked the reliability of the PU, PEU and PC scales in the context of our experimental settings. We applied the Alpha of Cronbach's Alpha test, which revealed the following results:

- For the PU scale, all the items showed a correlation higher than 0,3, while the global Cronbach's alpha was 0,817, giving proof of sufficient internal consistency among the PU items. It is therefore possible to calculate the mean, considering it to be a global rating of PU with each one of the three treatments (*code-centric*, MBD, MDD).
- For the PEU scale, all the items showed a correlation higher than 0,3, while the global Cronbach's alpha was 0,896, giving proof of high reliability of the scale. Again, this means that we can calculate the mean and consider this as a global rating of PEU with each one of the three treatments (*code-centric*, MBD, MDD).

- For the PC scale, we found very low levels of correlation (although significant) between the developers' perception of a method as "odd" and their level of experience reported with such a method. This being so, we opted to rely solely on the level of experience reported when measuring the PC in subsequent analysis. This is because, from our point of view, it reflects more clearly the definition of Compatibility given in our theoretical model.

Table 5–3 summarizes the means and standard deviations for the answers given to each question item. The whole questionnaire is presented in Appendix A.

Tabla 5–3: Descriptive statistics

Variables	***Code-centric***		**MDB**		**MDD**	
	Mean	SD	Mean	SD	Mean	SD
PU1	4,92	1,230	3,69	1,619	5,04	1,280
PU2	4,08	1,383	3,31	1,463	5,12	1,107
PU3	4,54	1,421	3,46	1,476	4,62	0,983
PU4	4,62	1,169	3,58	1,58	5,00	1,041
PEU1-R	4,35	1,056	4,00	1,233	5,15	1,047
PEU2	3,31	1,490	3,65	1,325	4,92	1,256
PEU3	4,40	1,225	4,00	1,354	5,12	1,211
PEU4-R	3,38	1,444	3,23	1,366	4,46	1,174
PEU5	3,73	0,827	3,00	1,118	4,62	1,134
PEU6-R	3,73	1,002	3,08	1,093	4,54	1,265
PEU7	4,65	0,936	3,6	1,359	5,04	1,113

3.3.1. *RQ*1*: Perceived usefulness of the development approaches*

To test the HPU hypothesis (concerning the existence of significant differences in the perceived usefulness of the different methods), we applied a $3*5$ Mixed Design ANOVA, in which the application domain (Trips, Events, Hospitals, Academics and Facework) was the between-subjects variable, and the PU ratings for each method was the within-subjects variable.

In order to assure that applying this statistical method made sense, we first checked the normal distribution of the dependent variables with the Levene's contrast. All the variables showed a significance parameter greater than 0,05, which allows us to assume normality. We also checked the homogeneity of covariance among groups with the Box's M

test ($F = 1{,}137; p = 0{,}295$). Last but not least, we verified that the principle of sphericity was not violated by applying Mauchly's W test ($W = 0{,}909; p = 0{,}387$).

The results showed that MDD produced the highest PU ($M = 4{,}90; SD = 0{,}795$), followed by code-centric ($M = 4{,}51; SD = 0{,}973$) and then MBD ($M = 3{,}48, SD = 1{,}191$). The results also showed that the interaction Meth∗App was not significant ($F(8, 42) = 1{,}753; MSE = 1{,}311; p = 0{,}114$). We can then safely examine the main effects of the two independent variables (application and method) on these means without needing to qualify the results by the existence of a significant interaction. The main effect of application did not attain significance ($F(4, 21) = 1{,}126; MSE = 1{,}459; p = 0{,}371$), but the main effect of method did reach significance, ($F(2, 42) = 19{,}411; MSE = 14{,}516; p \leqslant 0{,}001$). This means that *the differences in PU are significantly affected by the method used, regardless of the particular application being developed.*

Given the significance of the method variable, the last step of the analysis consisted in studying the pair-wise differences among methods through a matched T-test. In an effort not to increase the risk of a type-1 error, a Bonferroni adjustment was applied. This means reducing the significance threshold to 0,0167 ($p = 0{,}05/6933 = 0{,}0167$). With this adjustment, the PU differences between MBD and both code-centric and MDD approaches were significant, while the difference between code-centric and MDD PU scores was not.

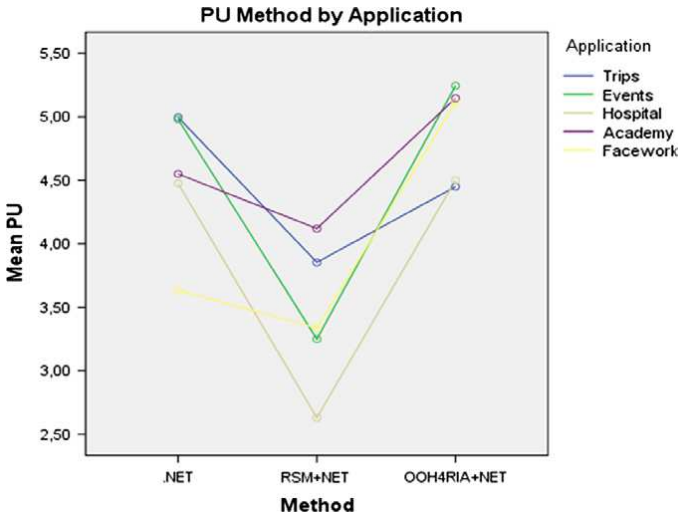


Figura 5–2: PU means by Method. Each line corresponds to one of the five application domains for which a social network was developed.

We can observe these results graphically in Fig. 5–2. The fact that the particular application was not significant is reflected in the five lines that are more or less overlapping. The Method variable influence is reflected in the acute ups and downs of the lines. Finally, the lack of significance of the Meth∗App interaction is reflected in the lines being more or less parallel (all of them showing the same tendency with each method). The same graphical clues hold for the rest of the graphics.

3.3.2. *RQ2: Perceived ease of use of the development approaches*

We then tested the HPEU hypothesis related to the perceived ease of use of the different methods. The chosen procedure, once again, was a 3∗5 Mixed Design ANOVA, in which the application (*Trips*, *Events*, *Hospitals*, *Academics*, *Facework*) was the between-subjects variable, and the PEU rating for each method was the within-subjects variable.

In order to assure that applying this statistical method made sense, we first checked the normal distribution of the dependent variables with the Levene's contrast. All the variables showed a significance greater than 0,05, which allows us to assume normality. We also checked the homogeneity of covariance among groups with the Box's M test ($F = 0{,}768; p = 0{,}78$). Last but not least, we verified that the principle of sphericity was not violated by applying Mauchly's W test ($W = 0{,}877; p = 0{,}269$).

The results showed that MDD produced the highest PEU ($M = 4{,}87; SD = 0{,}832$), followed by code-centric ($M = 3{,}98; SD = 0{,}757$) and then MBD ($M = 3{,}55; SD = 0{,}927$). The results also showed that the interaction Meth∗App is significant ($F(8, 42) = 2{,}801; MSE = 1{,}436; p < 0{,}014$). This means that the results vary according to the particular application domain for which the social network was being developed.

Another finding is that, while the main effect of application did not attain significance ($F(4, 21) = 1{,}033; MSE = 0{,}826; p = 0{,}414$), the main effect of method did reach significance, ($F(2, 42) = 24,704; MSE = 25{,}329; p < 0{,}001$). This means that *the differences in PU are significantly affected by the method used, regardless of the particular application*

being developed. This result, however, must be interpreted with caution, because the effect of the method is different in the different application domains.

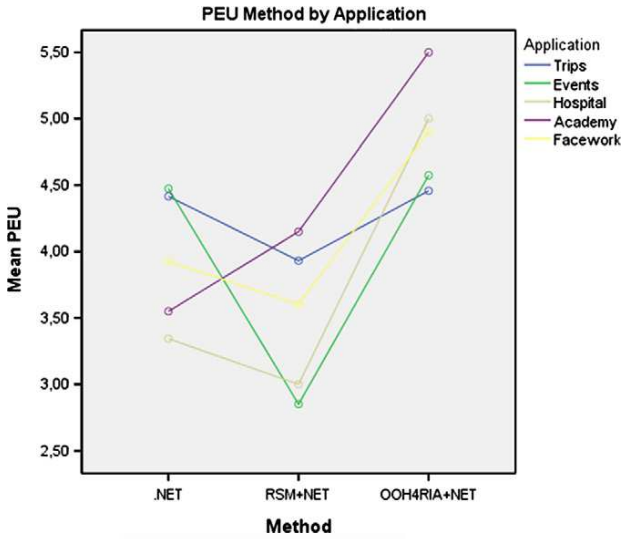


Figura 5–3: PEU means by method. Each line corresponds to one of the five applications that were developed as part of the experiment.

Although the main variable was found significant, the existence of a Meth*App interaction effect makes any further analysis of pair-wise comparisons unadvisable. We can observe these results in Fig. 5–3 in the form of a graph. It can be observed here that the significance of the Meth*App interaction is mostly due to the Academic application, whose MBD PEU follows a completely different trend from the remaining applications.

3.3.3. *RQ3: Perceived compatibility of the development approaches*

To test the HC hypothesis (concerning the existence of significant differences in the perceived compatibility of the different methods), we dropped the inter-subject factor and applied a one-way RM ANOVA. Our premise was that, as all the applications have the same set of functional requirements and belong to the same domain, the method compatibility (the degree to which the method is perceived as being consistent with the existing values, needs and past experiences of developers) did not depend on the particular application being developed.

On a 3-point scale (1 = low, 2 = medium, 3 = high), developers rated the method based on MBD as the most compatible ($M = 2{,}12; SD = 0{,}326$) by a slight margin, followed by the code based method ($M = 2{,}0; SD = 0{,}40$) and, finally, the MDD method

($M = 1,54; SD = 0{,}508$). This compatibility was based exclusively on their previous experience with the methods.

A check of the sphericity showed a significant W ($W = 0{,}471; p < 0{,}001$), which made us test the significance of the differences with a conservative Greenhouse-Geisser F. The results showed that the main effect of method did reach significance ($F(1{,}308, 32{,}706) = 15{,}492; MSE = 3{,}704; p < 0{,}001$); that is, the differences in PC are significantly affected by the method used.

A follow-up pair-wise analysis using a T-test with a Bonferroni adjustment showed that MDD is perceived as significantly less compatible than code-based and MBD, while differences between code-based and MBD are not significant.

3.3.4. *RQ4: Correlation among conceptual model dimensions*

As we have mentioned previously, ANOVA presents a high degree of robustness with respect to both ordinality and non-normality of the scale [52]. However, the robustness of correlations with respect to violations of the non-normality of distributions is not that straightforward. For this reason, in order to study the correlation between the three conceptual model dimensions included in our study, a non-parametric Spearman's correlation analysis was performed. Previous to this analysis, all the scales had been transformed into a 3-point scale (low, medium, high), for compatibility reasons. The results of this analysis can be seen in Table 5–4.

Tabla 5–4: Correlations between theoretical model variables (OEffc, OEffv, PU, PEU)

		PU	**PEU**	**PC**
PU	Rho Spearman	1	$0,663^{**}$	$-0,278^{*}$
	p		0,000	0,014
	N	78	78	78
PEU	Rho Spearman	$0,663^{**}$	1	$-0,288^{*}$
	p	0,000		0,011
	N	78	78	78
PC	Rho Spearman	$-0,278^{*}$	$-0,288^{*}$	1
	p	0,014	0,011	
	N	78	78	78

** The correlation is significant at the level 0,01 (bilateral).

* The correlation is significant at the level 0,05 (bilateral).

The results show how PC is negatively correlated with both PEU and PU, which means that, in this experiment, *developers regarded the development methods with which they were less familiar as more useful and easy to use*. These results are hard to explain, and contradict the theoretical model; they need further investigation with a more elaborate PC measurement scale.

In addition we can observe how, according to our data, PU and PEU are strongly correlated: the more useful the method is perceived to be, the easier it is to use (and vice versa). Since the two variables are dependent variables in our experiment, we cannot ascertain whether there is causality, nor tell in which direction that may be. More specific research needs to be carried out to find out more about these aspects of the theory, and, ideally, to come up with more elaborate constructs that are independent from each other.

3.3.5. *RQ*5*: Intention to adopt analysis*

When, after the experiment, our subjects were asked to choose one specific method to go on with the project, 20 subjects out of 26 (that is, 76,9 % of the sample) decided to use the MDD approach.

3.3.6. *RQ*6*: Perceived advantages and disadvantages of methods*

For a better understanding of previous objective results, we complemented our study with an opinion survey, in which developers reported in a free-form format the main advantages they perceived, along with the disadvantages they saw in each method. After gathering the results, and in order to organize the natural language responses, we classified each advantage/disadvantage under a common epigraph (see the first column of Table 5–5). Moreover, for each method, these features were reported as either an advantage or a drawback.

Starting with the traditional code-centric approach (see second column of Table 5–5), we can observe that 50 % of subjects perceived the feeling of control that they experienced when working with code as one important advantage of the code-centric approach, while none regarded that as a disadvantage. In addition, subjects reported as advantages a lower

learning curve (5), higher personalization (4) and higher compatibility, when compared to their previous experience (3). At the other end of the spectrum, according to subjects, the main drawback of code-centric is that it requires a high development effort (with 20 subjects giving responses that fell in that category) and it has a low maintainability (5 subjects).

Tabla 5–5: Reported advantages/drawbacks of each method.

Reported Feature		***Code-centric***	**MBD**	**MDD**
Development Effort	Advantage	1	10	24
	Drawback	20	5	1
Feeling in control	Advantage	13	–	–
	Drawback	–	–	3
Learning Curve	Advantage	5	5	2
	Drawback	1	3	6
Compatibility	Advantage	3	–	–
	Drawback	–	2	5
Maintainability	Advantage	–	6	3
	Drawback	5	–	–
Personalization	Advantage	4	–	–
	Drawback	–	1	–
Reliability	Advantage	–	–	–
	Drawback	–	12	7

With respect to the MBD approach using RSM (see third column of Table 5–5), the advantages of the method varied greatly. The most important ones were reduced development effort (10), better maintainability (6) and a reduced learning curve (5). However, these advantages were not appreciated universally; development effort and learning curve, for example, were pointed out as disadvantages by 5 and 3 subjects, respectively. We should remark, too that subjects commented that the code generation capabilities provided by RSM were clearly insufficient (5 subjects). This noted, the main MBD drawback reported was a significant lack of perceived reliability of the approach (12 responses), due to some errors detected in the RSM code-generation process.

Finally, we asked the same question regarding the OOH4RIA approach as an exponent of the MDD paradigm. The main disadvantages reported as regards this method were a higher learning curve (6), a low compatibility with their previous experience (5), and a

lack of reliability (7), which may have been due to some bugs of the OOH4RIA IDE tool that appeared during the development phase. An overwhelming majority of students (24 out of 26) recognized an important increase in development speed, however, along with a significant reduction in effort with repetitive tasks. Another result was that maintainability was regarded as an advantage by 3 subjects.

3.4. Threats to validity

The threats to validity evaluate under what conditions our experiment is applicable, offering benefits, and under what circumstances it might fail. We have classified the threats into four families [54]: internal, external, construction and conclusion threats.

3.4.1. *Threats to conclusion validity*

These refer to the relationship between the treatment and the outcome. All the statistical analyses have been preceded by tests that assured that the assumptions of the statistical procedure were not being violated. The alpha has been adjusted, following the most conservative approach (Bonferroni), which also contributes to the conclusion validity. The intra-subject design also contributes to protecting the results against heterogeneity of subjects, since all of them were presented with the three methods. Notwithstanding, the number of observations per cell was 5, an absolute minimum to be able to apply this kind of analyses. Also, given the duration of the different treatments (each subject was working with each method for 2 weeks), random irrelevancies in the experimental setting might have occurred that may have affected the data. We can assume, however, that such irrelevancies have affected all the levels of the treatment equally.

3.4.2. *Threats to internal validity*

These are concerned with the possibility of hidden factors that may compromise the conclusion that it is indeed the treatment which causes the differences in outcome. In order to increase the internal validity of the design, subjects received sufficient training in all three methods. All treatments were applied in random order (with only the MDD-Code-MBD order being left in the experiment, due to a group dropping out of the course soon

after the experiment began). There was consequently no reason for compensatory rivalry, equalization or demoralization.

All the applications and modules on which the subjects worked were of similar complexity. The lack of influence of the particular application on the results was tested statistically. However, even when we counter balanced the design by randomizing the order in which the different subjects used the different methods, subjects still applied the three methods in a row. This makes it possible the existence of carryover effects to (e.g. subjects could benefit in the course of the experiment from increasing familiarity with the experiment setting, the development environments, etc.) or, on the contrary, there may be fatigue effects (e.g. through a loss of interest in the registration of the experiment data). It is also possible for there to be an effect coming from the spread of treatment imitation. By this we mean the phenomenon of users learning from the previous method and imitating the practices when applying the next method assigned. To diminish such risks, we supervised the whole process, and maintained the amount of information that needed to be provided by the users to a minimum. We also automated the data gathering process, to prevent coding errors.

3.4.3. *Threats to construct validity*

These refer to the relationship between theory and observation. During the experiment, both the theoretical model and the hypotheses were carefully kept from the subjects. No special emphasis was put on the pros and cons of any of the methods until after the experiment was finished and the opinions were gathered. Methods were used together with their supporting tools, and the particular technologies used to test each development approach are widely used in practice.

However, to the best of our knowledge, there are no standard instruments to measure the components of the theoretical model applied in this article. Although the reliability of the scales has been checked to increase the construct validity, the number of items of some of the scales used should definitely be increased and validated through further research if

they are to become more robust. Particularly, a careful study of all the aspects that may play a role in perceived usefulness, ease of use and compatibility (e.g. the perceived quality of the resulting product as part of the perceived usefulness scale) should be carried out to back the inclusion of new subjective items to each scale.

3.4.4. *Threats to external validity*

Lastly, threats to external validity are concerned with the generalization of the results. The type of application used for the experiment, far from being a toy example, is a real application, defined on the basis of true client requirements. The subjects are graduate students, many of whom are already working as developers and who are therefore true representatives of junior developers. Having said that, given the relatively small sample used (26 subjects), and the fact that the sample was not random, but chosen based on their participation in a master degree, our experiment still presents a lack of sampling representativeness. It is also the case that we have used similar size applications, all of the same type (social networks). We therefore cannot generalize the results to applications of different sizes or different application types without more replicas. Finally, as regards these kinds of threats, although the tools used in conjunction with the methodologies are widely used among practitioners, they are not the only possibilities. The experiment thus needs to be replicated in order to make sure that it is indeed the method used, not the tool, that causes the differences observed.

4. Related work

Over the last years we have witnessed how the number of empirical studies regarding the subjective perceptions while applying different methods has increased. In [55], the author empirically assessed the satisfaction of an MDD method (called MIMAT) that included Functional Usability Features (FUFs) in an MDD software development process. The study concluded that the user's satisfaction improved after including FUFs in the software development process. Our experiment does not centre on the enrichment of a given method with a new artefact/technique, but compares different methods with respect

not only to the developer's perceived ease of use but also to the perceived usefulness and compatibility.

Closer to our experiment, Arisholm et al. [56] reported on a set of controlled experiments that investigated the impact of UML documentation (thus assuming an MBD approach) on software maintenance. Results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements in the functional correctness of changes and the design quality. This study differs from ours both in focus (maintainability vs. intention to adopt) and methods covered (in our case including not only an MBD but also an MDD method). Another recent experiment [57] compares three treatments or levels in the use of UML models (no modeling conventions, with modeling conventions and with tool-supported modeling conventions). The experiment was performed with 106 M.Sc. students, organized in 35 teams. The findings showed that development effort with tool-supported modelling conventions is approximately half of the effort invested when using traditional development (code-centric). It also revealed that there is no significant correlation between class-count and the effort spent in modeling. Again, our experiment differs from this one in focus, since we have not centered our study on effort, but rather on intention to adopt the method, based on subjective perceptions.

There are also studies that have centered on the set of tools that accompany each paradigm: Pelechano et al. [58] performed an empirical comparison of Eclipse Modeling Plug-ins and Microsoft DSL Tools, on the basis of their utility and as regards satisfaction from the point of view of developers. The results showed that both tools are very useful and can be used in future projects. Eclipse users are 100 % faithful to this environment; on the other hand, 60 % of the DSL Tools users would migrate to Eclipse. In [59] an other author performed a similar comparison between Microsoft DSL Tools and Eclipse EMF/GEF/GMF Frameworks. Results show that the MS/DSL Tools metamodel designer is more usable than the EMF metamodel designer. Our study focuses less on tools and more on whole methods that can be classified in a given development paradigm.

5. Conclusions and further lines of research

This study makes three main contributions. On the one hand, it presents a tailored theoretical method adoption model that can be used as a starting point to study the adoption possibilities of existing or new development methods, based as it is on well-known evidence from different fields. Such a theoretical framework points to perceived ease of use, perceived usefulness, compatibility, subjective norm and voluntariness as being the main variables that influence the intention to adopt a given method. On the other hand, it presents an empirical comparison of the perceived usefulness, ease of use and compatibility of three methods and their corresponding developing environments which, in turn, are clear exponents of the three mainstream development paradigms nowadays: code-centric, MBD and MDD. This study was designed taking into account the common belief that the use of models improves the developers' perceived usefulness of the method [1-3], although it somehow made the development process more difficult. Contrary to our assumptions and the related work [56], using models in the context of an MBD environment is regarded as useless and difficult, even if the subjects did not show a great degree of incompatibility with the approach.

As expected, the MDD method was regarded as the least compatible with developers' current practices, but the most useful in the long run. The fact that four out of five teams chose to continue using the MDD method suggests that usefulness seems to have a much greater impact on intention to use a method than compatibility, at least among junior developers.

If we take a look at the reported advantages/disadvantages, which may explain the developers' decision to adopt/reject a given method, it seems that, for MDD, a lower development effort compensated for some perceived disadvantages, such as a greater learning curve, lower compatibility or even lower reliability. This poses interesting questions in relation to our theoretical model: should perceived usefulness be weighted as having more importance than perceived ease of use or compatibility? Our data backs the claim made in [40], where it was assessed that usefulness had a much greater impact on intention to

employ a method than ease of use. Moreover, in our experiment compatibility is negatively correlated with usefulness and ease of use. This poses some doubts about the relevance of compatibility in a software development method adoption model as an independent dimension, at least in the context of junior software developers.

The last contribution we would like to mention is that, to the best of our knowledge, this study provides the first set of evidence on the perceived strengths and weaknesses of the different paradigms with respect to intentions to adopt them.

All three contributions open new lines of research. If we consider the theoretical model, we find that only three out of the five initial components have been studied. This, together with the correlations found between the three components included, suggest that it may be advisable to refine the model and the measurement instruments. That would diminish the co-linearity of the variables and make it possible to fine-tune the prediction power of the model. Such prediction power has not been formally tested, and remains as another line of work. As regards specific methods, further experimentation is needed to separate the effect of methods from that of their accompanying development environments (Visual Studio 2010, Rational Software Modeler or the OOH4RIA tool) and to be able to generalize the results to a different population, or to different methods and languages, application types or application sizes. The perceived advantages and disadvantages of the methods and their potential impact on the final decision to adopt them provide not only valuable information for further refining the theoretical framework; they also give some clues as to how to make decisions about the prioritization of improvements that need to be made to the development environments.

Acknowledgements

This paper has been co-supported by the Spanish Ministry of Education, and the University of Alicante under contracts TIN2010-15789 (SONRIA) and GRE10-23 (DISEM-RIA). The authors also wish to thank their students for taking the time to participate in this empirical study. In addition, we would also like to thank Jose Javier Martinez and

Juan Antonio Osuna who contributed to the development of the OOH4RIA Tool. Special thanks to the reviewers for their useful comments, which greatly helped to improve the quality of this work.

References

[1] G. Cernosek, E. Naiburg, The Value of Modeling, 2004.

[2] Wikipedia, Leaky abstraction. ¡http://en.wikipedia.org/wiki/Leaky-abstraction¿(accessed 01.03.12).

[3] B. Shneiderman, The Eyes Have It: A Task by Data Type Taxonomy for, Information Visualizations, 1996.

[4] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 2004.

[5] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[6] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, 2003. p. 298.

[7] M. Eichberg, M. Monperrus, S. Kloppenburg, M. Mezini, Model-driven engineering of machine executable code, Writing 6138 (2010) 104 – 105.

[8] R. Picek, Suitability of modern software development methodologies for model driven development, System 33 (2) (2009) 285-295.

[9] S. Ambler, Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, Wiley, 2002.

[10] MagicDraw UML. https://www.magicdraw.com/ (accessed 01.03.12).

[11] Visual Paradigm for UML 8.3. http://www.visual-paradigm.com/product/vpuml/ (accessed 01.03.12).

[12] Rational Software Architect. http://www.ibm.com/developerworks/rational/products/rsa/ (accessed 01.03.12).

[13] Evolving Software, True View Domain Modeller. http://www.evolving-software.co.uk-/trueview-domain-modeller.html (accessed 03.09.11).

[14] Borland, Together: Visual Modeling for Software Architecture Design. http://www.borland.com/us/products/together/ (accessed 01.03.12).

[15] S. Meliá, J. -javier Martínez, S. Mira, J.A. Osuna, J. Gómez, An eclipse plug-in for model-driven development of rich internet applications, Development 67078 (2010) 514-517.

[16] P. Milano, Designing Data-Intensive Web Applications WebML - WebRatio, Database, No. October, 2004.

[17] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, S. Neema, Developing applications using model-driven design environments, vol. 39 (2), IEEE Computer Society Press, 2006, pp. 33-40.

[18] R. Picek, V. Strahonja, Model driven development - future or failure of software development?, IIS 7 (2007) 407-413

[19] P. Mohagheghi, V. Dehlen, Where is the proof? - A review of experiences from applying MDE in industry, in: Model Driven Architecture-Foundations and Applications, vol. 5095, 2008, pp. 432-443.

[20] M. Guttman, J. Parodi, Real-Life MDA, in: Reallife MDA Solving Business Problems with Model Driven Architecture, Elsevier, 2007.

[21] T. Stahl, M. Völter, J. Bettin, A. Haase, S. Helsen, K. Czarnecki, Model-Driven Software Development: Technology, Engineering, Management, John Wiley and Sons, 2006. pp. 970-978, (ISBN 9780470025703).

[22] H. Gustavsson, B. Lings, B. Lundell, A. Mattsson, M. Beekveld, Integrating proprietary and open-source tool chains through horizontal interchange of XMImodels, in: IEEE International Conference in Software,Maintenance, 2007, pp. 521-522.

[23] W. Heijstek, M.R.V. Chaudron, Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process, in: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, August 2009, pp. 113-120.

[24] S.J. Mellor, A.N. Clark, T. Futagami, Model-driven development - guest editor's introduction, IEEE Software 20 (5) (2003) 14-18.

[25] J. Muñoz, V. Pelechano, MDA vs Factorías de Software, Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos MDA y Aplicaciones DSDM 2005, 2005, p. 41.

[26] S.L. Pfleeger, Understanding and improving technology transfer in software engineering, Journal of Systems and Software 47 (2-34) (1999) 111-124.

[27] D.L. Moody, The method evaluation model: a theoretical model for validating information systems design methods, in: ECIS, 2003.

[28] R.P. Abelson, A. Levi, Decision making and decision theory, Handbook of Social Psychology 1 (1985) 231-309.

[29] T. Adelbratt, H. Montgomery, Attractiveness of decision rules, Acta Psychologica 45 $(1-3)$ (1980) $177-185$.

[30] P. Wright, Consumer choice strategies: simplifying vs. optimizing, Journal of Marketing Research 12 (1) (1975) $60-67$.

[31] A. Kleppe, J.Warmer,W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, vol. 83 (7), Addison-Wesley, 2003, p. 192.

[32] S.Walderhaug, M. Mikalsen, I. Benc, S. Erlend, Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project, in: From Code

Centric to Model Centric Software Engineering: Practices, Implications and ROI. Workshop at European Conference on Model-Driven Architecture, 2008.

[33] R. Agarwal, J. Prasad, A field study of the adoption of software process innovations by information systems professionals, IEEE Transactions on Engineering Management 47 (3) (2000) 295 – 308.

[34] G.C. Moore, I. Benbasat, Development of an instrument to measure the perceptions of adopting an information technology innovation, Information Systems Research 2 (3) (1991) 192 – 222.

[35] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: International Symposium on, Empirical Software Engineering, 2005, pp. 92 – 101.

Appendix 5.A

5.A.1. Perceived usefulness

Please rate your level of agreement with the following statements:

	1	**2**	**3**	**4**	**5**	**6**	**7**
PU1: I believe that my overall throughput with this software development method has been equal to that of an expert							
PU2: I believe that this software development method is efficient							
PU3: I believe that this software development method is reliable							
PU4: I believe that this software development method is useful							

5.A.2. Perceived ease of use

In general, I feel that this software development method is. . .

	1	2	3	4	5	6	7
PEU1: (1) Easy to use .. Difficult to use (7)							
PEU2: (1) Tiring .. Not tiring (7)							
PEU3: (1) Demanding .. Simple (7)							
PEU4: (1) Fun .. Boring (7)							
PEU5: (1) Stressful .. Relaxing (7)							
PEU6: (1) Pleasant to use .. Unpleasant to use (7)							
PEU7: (1) Inacceptable .. Acceptable (7)							

5.A.3. Perceived compatibility

PC1: What is your level of previous experience with the techniques and tools involved in this method?

- Low
- Medium
- High

5.A.4. Main advantages/disadvantages

Please note down the main three DISADVANTAGES of using this software development method

1. -
2. -
3. -

Please note down the main three ADVANTAGES of using this software development method

1. -
2. -
3. -

Parte III

APÉNDICES

.

APÉNDICE A
MODEL DRIVEN MAINTAINABILITY ASSESSMENT OF WEB APPLICATIONS.

El contenido de este capítulo corresponde con el siguiente artículo: Martínez, Y., Cachero, C. and Meliá, S. - Model Driven maintainability assessment of Web Applications. Actualmente en proceso de revisión en la revista Empirical Software Engineering.

Empirical Software Engineering es una revista que provee una vía para la discusión de investigaciones relacionadas con la ingeniería de *software* con un fuerte componente empírico. Los estudios empíricos presentados usualmente involucran la recopilación y análisis de información y experiencia que pueden ser utilizadas para caracterizar, evaluar e identificar relaciones entre prácticas, tecnologías y resultados en el desarrollo de *software*. Los temas de interés son: ingeniería de *software*, programación y sistemas operativos, lenguajes de programación y compiladores. La revista también ofrece revisiones detalladas de aplicaciones de tecnologías de *software* -procesos, métodos o herramientas-, así como su eficacia en la industria.

La revisata es editada por *Springer Netherlands* y tiene un factor de impacto de 1.854 en el 2011, de acuerdo con el índice de cita de *Thomson's Science Citation Index* (SCI) (http://www.isiwebofknowledge.com/).

Model Driven maintainability assessment of Web Applications

Yulkeidi Martínez[1], Cristina Cachero[2], and Santiago Meliá[2]

[1] Universidad Máximo Gómez Báez de Ciego de Ávila, Cuba.

[2] DLSI. Universidad de Alicante, Spain.

Abstract. BACKGROUND: Model-driven Engineering (MDE) approaches are often acknowledged to improve the maintainability of the resulting applications. However, there is a scarcity of empirical evidence that backs their claimed benefits and limitations with respect to *code-centric* approaches.

OBJECTIVE: To compare the performance and satisfaction of junior software maintainers while executing corrective and perfective maintainability tasks on Web applications with two different development approaches, one being OOH4RIA, a *Model-driven* approach, and the other being a traditional, *code-centric* approach based on the Agile Unified Process.

METHOD: We have conducted an intra-subject quasi-experiment with 27 graduated students from the University of Alicante. They were aleatory divided into two groups, and each group was assigned to a different Web application (PetStore and MediaPlayer), on which they performed a set of maintainability tasks.

RESULTS: The data indicates that using the OOH4RIA MDE approach improves the performance of subjects when they carry out maintainability tasks. It also tips the satisfaction balance in favor of MDE approaches.

CONCLUSIONS: Model-driven development methods seem to improve both maintainability and satisfaction during the maintenance phase of Web development. Consequently, they show a great potential for adoption. However, further experimentation is needed to be able to generalize the results to different populations, methods, languages and tools, different domains and different application sizes.

1. INTRODUCTION

Model Driven Engineering (MDE) is a software development approach that heavily relies on models and model transformations for the development, maintenance and evolution of software [37, 43, 38]. The MDE community claims several advantages over traditional development (*code-centric*) approaches, among which short and long term productivity gains, improved project communication and improved quality of the resulting application outstand [30, 20, 22]. The reason for such claims is that current MDE approaches offer high-level abstractions that capture some of the most salient characteristics of modern applications. Such abstractions can speed up the definition of application models that are abstracted from implementation details, and therefore can be used to generate applications for different implementation environments.

In particular, software maintenance is one of the areas in which MDE claims to be able to make a greater impact in terms of both reliability and efficiency [10]. Software maintenance is defined in the IEEE Standard 1219 [28] as "the modification of a software product after being delivered to correct faults, improve performance or other attributes, or to adapt the product to a modified environment". More concisely, the ISO/IEC 25010 standard [29] states that maintainability is "the capability of the software product to be modified". Software maintenance is important because: (a) it consumes between 45 % and 60 % of the overall life cycle costs [50] and, (b) the inability to change software quickly and reliably means that business opportunities may be lost.

However, despite the maintainability advantages claimed by the MDE community and the potential impact of such claim, practitioners still lack a body of practical evidence that soundly backs this assertion [24, 41]. This contrasts with other disciplines and even other areas of Software Engineering (SE) [56]. Without empirical evidence, there is a danger that resources be wasted and that software tools fail to develop appropriately [26]. Such evidence can be provided in the shape of empirical studies, which are crucial to the evaluation of processes and human-based activities. The empirical studies include surveys, experiments, case studies and postmortem analyses [55]. Experimentation, specifically,

provides a systematic, disciplined, quantifiable and controlled way of evaluating human-based activities.

In an effort to increase the empirical evidence regarding the impact of MDE approaches on maintainability, this paper presents a quasi-experiment in which two groups of junior developers have been asked to perform a set of maintainability tasks on two Web applications.

The paper is structured as follows: Section 2 characterizes the maintainability concept. Section 3 presents the empirical evidence regarding how maintainability is impacted by the adoption of MDE methods. This characterization and the empirical evidence found in literature set the context for the definition in Section 4 of the experimental design (context, planning, operation and data collection mechanisms). Section 5 presents the data analysis and an interpretation of results that takes into account the identified threats to validity. Last, Section 6 concludes the paper and presents some further lines of research.

2. MAINTAINABILITY CONCEPTS

As we have previously mentioned, the ISO/IEC 25010 standard [29] defines maintainability as "the capability of the software product to be modified". Such modifications can be of four different types [14]:

- *Corrections*: Corrective maintainability refers to the capability to detect errors, diagnose the problems and fix them.
- *Improvements*: Perfective maintainability refers to the capability to extend the software according to new requirements or enhancements.
- *Adaptations*: Adaptive maintainability refers to the capability to modify the software in order to cope with the effects of environmental changes.
- *Preventions*: Preventive maintainability refers to the software capability to support internal reengineering processes without adverse impact.

These four types can be grouped into *corrective tasks* (corrections) and *perfective tasks* (which encompass improvements, adaptations and preventions). From them, corrections

and improvements are the two most common types of maintainability tasks in software development [41], and therefore the two types of tasks that most heavily influence the global maintainability of software applications.

Regardless of the type of maintainability task, in order to assess the maintainability of software applications the ISO standard establishes that five different sub-characteristics must be taken into account:

- *Analysability*: Capability of the software product to be diagnosed for deficiencies or causes of failure in the software, or for the parts to be modified to be identified.
- *Changeability*: Capability of the software product to enable the application of a specified modification.
- *Stability*: Capability of the software product to avoid unexpected effects from modifications of the software.
- *Testability*: Capability of the software product to enable modified software to be validated.
- *Compliance*: Capability of the software product to meet the standards or conventions relating to maintainability.

The evaluation of such sub-characteristics may greatly vary depending on the way in which the maintainability task is carried out. Maintainability tasks can be tackled in either an unstructured or a structured manner. On the one hand, unstructured maintenance postulates wading straight into the code of the target application in order to make the necessary changes, normally with the help of specialized tools. Sometimes this process relies on -automatic - maintainability measures that operate over the code of the target application [16]. On the other hand, structured maintenance postulates examining and modifying a pre-existing design, and then, either manually or automatically, redefining the code to match it. Many authors claim that structured maintenance is a more reliable and efficient process than unstructured maintenance [10].

3. RELATED WORK

According to a systematic mapping covering the period 2001-2010 [34], the empirical assessment of maintainability activities in MDE methodologies has been gaining momentum during the last years. This notwithstanding, according to this study, out of the 599 publications related to the impact of the MDE paradigm on the product quality and process productivity, only 14 (that is, barely a 2,34 %) focused on maintainability. Regarding the impact of MDE over maintainability (with respect to traditional, *code-centric* approaches), there is empirical evidence of the need of around 37 % less time to evaluate the impact of a change in academic settings [38]. In industry, evidence has been gathered on how the use of MDE causes defect reductions, reduced need for code inspections, and produces maintenance gains, although such evidence is anecdotal [42]. In [25], based on the results of an online survey, the authors report how the majority of respondents (between 58 and 66 %) considered the use of MDE on their projects to be beneficial in terms of personal and team productivity, maintainability and portability. Organizationally, the benefits of MDE have been defined in terms of communication and control: almost 75 % of respondents agreed that using MDE made them faster at implementing new requirements while nearly two thirds of respondents reported that their use of MDE helped in the understanding and communication of organizational knowledge among stakeholders.

Last but not least, a recent empirical study [33] compared an MDE method (WebML) [13] and a *code-centric* method (based on PHP), with respect to the performance and satisfaction of junior software developers while executing analysability, corrective and perfective maintainability tasks on Web applications. This study showed that the involved subjects performed better with WebML than with PHP, although they showed a slight preference towards tackling maintainability tasks directly over the source code.

4. DESCRIPTION OF THE EXPERIMENT

In May 2011, a quasi-experiment was conducted at the University of Alicante. A quasi-experiment is a type of controlled experiment in which individuals or teams (the

study units) engage in one or more tasks for the sake of comparing different processes, methods, techniques, languages or tools (the treatments) [2]. In quasi-experiments subjects are not randomly chosen but rather selected on the basis of certain criteria. In our study, such criterion has been their attendance to a Master's degree. Many authors have written about the importance of providing empirical evidence in SE [18, 30]. Quasi-experiments, although suffering from a lower internal validity than true experiments, are widely used and deemed useful in the Empirical SE field, since they allow investigations of cause-effect relations in settings such as ours, in which randomization is too costly [2].

In the context of the characterization presented in Section 2, the global aim of our study has been to compare structured *vs.* unstructured changeability when dealing with corrections and improvements of modern Web applications.

Unstructured maintenance has been performed with a code-centric approach (based on *C#* and the .NET framework), while structured maintenance tasks have been performed with an MDE approach, based on the OOH4RIA approach and supported by the OOH4RIA IDE tool. We are conscious that the choice of specific tools and languages diminishes the external validity of the study, that is, the generalizability of the results. Such a selection was necessary, nevertheless, in order to provide a real working environment, and thereby increase the construct validity of the experiment. This issue will be further discussed in Section 5.6.

4.1 Goals and Context Definition. Following the GQM template [48], this empirical study is aimed *at analyzing* OOH4RIA and .NET *for the purpose of* evaluating model-driven (structured) against code-centric (unstructured) changeability practices *with respect to* their performance and satisfaction *from the point of view of* junior developers. *The context of the study* is graduate students enrolled in the "Master in Web Applications Development" at the University of Alicante, where both a code-centric Web development process (based on a .NET language) and a model-driven Web development process (based on the OOH4RIA method) are taught.

On the one hand, the focus on changeability is due to the fact that, specially in MDE environments, not all the maintainability sub-characteristics are equally critical. Provided that the chosen MDE approach is mature enough (such as is the case of OOH4RIA), we can safely assume testability and compliance: in an automated code generation environment these issues are taken good care of during the code generation process. Also, OOH4RIA provides many clues about possible errors and side effects -a facility that is not available for the .NET environment-, which hampers the comparability of any kind of analyzability or stability measures.

On the other hand, as we have previously mentioned, the selection of corrections and improvements as the task types included in our study is due to the fact that corrections and improvements are the two most common types of maintainability tasks in software development [41]. The design of the experiment is based on the framework for experimentation in SE research [55]. Also, a laboratory package has been compiled for the sake of replicability [7].

The research questions addressed in this study have been formulated as follows:

RQ1: How does the maintainability performance (objective efficiency and objective effectiveness) of the OOH4RIA methodology compare with respect to the maintainability performance of a traditional, *code-centric*, methodology using .NET?

RQ2: How does the maintainability satisfaction (perceived usefulness and perceived ease of use) of the OOH4RIA methodology compare with respect to the maintainability satisfaction of a traditional, *code-centric*, methodology using .NET?

All the questions have been devised to be answered by quantitative means.

4.1.1 Subjects. The experimental subjects of our study were 27 students enrolled in the "Master in Web Applications Development" at the University of Alicante during the year 2010-2011. The sample comprised 26 men and 1 woman, of whom 75 % had more than 2 years of experience developing Web applications. The mean age of the participants

was 25,6 years old and all of them were Computer Engineering graduates of the University of Alicante.

Regarding the subjects' level of knowledge with respect to the different technologies and methods used during the experiment, a pretest questionnaire showed that the subjects had no previous practical knowledge of MDE, although 56 % of them were aware of the existence of the paradigm. This notwithstanding, 81 % knew UML (the standard on which the OOH4RIA method is based), and, from them, 12 % considered that they had a high-level of knowledge of UML. It should also be noted that 76 % of the subjects had previously programmed with .NET technology during their degree course, although only 12 % had applied it in industry. By the time the experiment took place, the subjects had received additional training both in .NET and OOH4RIA. Such training consisted in 30 hours of training in programming in *C#* using Visual Studio 2010 and 30 hours of training in UML modeling and the OOH4RIA tool.

4.1.2 Sensitivity analysis of the design. Since our sample size was fixed, and came determined by the number of students enrolled in the master degree, we decided to conduct a power analysis before going on with the experiment. The objective of this analysis was to make sure that the analyses had enough power as to limit the risk of a Type II error (accepting the null hypothesis when in fact is false) to a minimum. Cohen [15] suggests that the power of an analysis should be greater or equal to 0,7 to be of use. This means a maximum of a 30 % chance of failing to detect an effect that is there.

Also based on the work of Cohen [15] and some previous results reported in literature (see Section 3), we decided to set 0.3 as the minimum effect size we were interested in (that is, the method used accounting for at least 30 % of overall (effect+error) variability). Lower effects are of little interest in our context, given the great investment needed to change a development method inside an organization.

The 27 subjects of our experiment yielded 54 observations, which is the actual sample size, since we did not take advantage of the repeated measures character of our design. The reason is that the tasks performed with each methodology were not equivalent.

A sensitivity analysis of all these parameters ($alpha = 0{,}05; desiredpower = 0{,}70; totalsamplesize$ 54) yielded a detectable effect size of 0,34, which approaches the 0,3 limit set by our design. Therefore we decided to go on with the experiment. This calculation was done with the statistical tool G*power 3.1.5 [1].

4.1.3 Applications. Subjects were randomly assigned to either one of the following two applications:

- A Petstore application (an adaptation of an example used in [51]). This application is a virtual store of pets (Petstore) in which a client (Client) can carry out many purchase orders (Orders). The client can add to her purchase as many order lines (OrderLines)as she needs. A pet (Article) can be associated to many order lines. Pets (Article) can be classified into categories (Category), *e.g.* Bird, and subcategories (Subcategory), *e.g.* Predator.
- A Mediaplayer application (used in [19, 20]). This application is a music reproducer that allows a User (User) to manage a group of songs (Song), and to define lists of reproduction (Playlist). The application allows to organize each song according to different criteria (gender, year, etc.). Also, each song, besides the URL location, may contain information on the record company, cover, letter, duration, etc. The application also stores the artists (Artist) and the albums (Album) to which such song belongs.

For both applications, subjects performed the maintainability tasks on the server side. Both applications share the application type (data-intensive) and the complexity. Such complexity was controlled by assuring that both applications had a similar number of lines of code in .NET [4] and, at the same time, a similar number of conceptual classes in OOH4RIA. Lines of code is generally preferred over functional points to measure complexity due to its higher reliability since it does not involve human judgement [55].

4.1.4 Implementation environment.

The implementation environment was chosen taking into account the need for such environment to be able to successfully deal with the kind of important challenges posed by modern Web applications regarding architecture, functionality and user interfaces. To face these new necessities, a number of technologies have been proposed, fostering the adoption of server-side and client-side languages for the Web. Within this plethora of proposals, some technologies have been largely adopted by practitioners. Server-side scripting is one of such technologies, through which the generation of HTML markup is achieved by means of languages like JSP, PHP, or any .NET language (such as Visual Basic, Visual *C++* with Managed Extensions, *C#* or JScript), to name a few. Among them, *C#* [9] is defined as a simple, modern, object-oriented, and type-safe programming language derived from *C* and *C++*, developed especially for the .NET platform. The .NET runtime components, frameworks, and languages are all tied together under the Visual Studio environment [23]. In our experiment we have chosen *C#* and Visual Studio .NET as representatives of a modern programming environment for the Web that is currently being used by a large community of software practitioners.

With respect to the MDE paradigm, there are two tool mainstreams:

- Some proposals opt to apply the UML profiling mechanism to extend the different UML elements by defining constraints and adding tag values, so as to introduce domain-specific semantics. In this category, there is a large set of tools that enables code to be generated. However, only two of these tools can generate an NHibernate-based object-oriented business logic,as well as persistence layers. These are Modelio [6] and Visual Paradigm [3]. Both of them are commercial, and they do not provide academic licenses.
- A second set of proposals, which has been gaining momentum over recent years due to the increasing complexity of Web applications [53], defines Domain-Specific Languages (DSLs), that is, relatively simple languages (graphical or textual) that are created to model some problem domain. To do that, each DSL requires the definition of a MOF metamodel (including OCL constraints) and a graphical or textual language to represent

its concrete syntax. Among the commercial MDE tools in this second set we can cite Integranova [5] and RadarC [27]. Also, several proposals have been developed in universities, such as OOH4RIA [36], OOHDM [52], RUX [31], WebML [13], UWE [49] and OOWS [54].

Given the fact that none of the aforementioned proposals is standard or even widely adopted, and due to the lack of academic licenses for any of the commercial tools, we have opted to use OOH4RIA as a representative of a modern MDE environment. The OOH4RIA approach extends the OOH method [12] and proposes a complete development process based on a set of models and transformations that allow to go from conceptual models to code. OOH4RIA is also equipped with an Implementation Development Environment (IDE) [37] that offers support for both the design activities and the automatic code generation process. This IDE is based on the Eclipse Modeling Project [21], an open source software whose main purpose is to provide a highly integrated platform tool [11]. OOH4RIA is free of charge for universities, and it is taught as part of the Master's degree in which the experiment took place.

Summarizing, the following implementation environment was set up:

- Development framework: .NET framework, Silverlight 4.0 and NHibernate (Object-Relational Mapping).
- Coding IDE: Visual Studio 2010.
- Modelling tool: OOH4RIA
- Code Generation tool: OOH4RIA
- Languages: *C#* and XML Mapping (ORM mapping of NHibernate).
- Other tools: The set of questionnaires filled-in by each developer, which were published online.

4.2 Experiment Planning. As we have mentioned previously, and due the fact that the subjects were not randomly chosen but rather selected on the basis of their attendance to a Master's degree, our study belongs to the quasy-experiment category.

The subjects were randomly assigned to either one of the two available applications, and they were asked to perform ten maintainability tasks, five with the code-centric approach and five with the model-driven approach. Some of the tasks were corrective and some were perfective. For each task type they started with a new project that they downloaded from the master's Web site [52]. They had a time limit of two hours. The order in which each subject applied each method was randomized to avoid order effects. Such randomization can be seen in Table A–1.

Tabla A–1: Experiment cross-tabulated design

Subject	**Corrections**	**Improvements**	**Application**
S1-S6	OOH4RIA	.NET	Petstore
S7-S13	.NET	OOH4RIA	Petstore
S14-S20	OOH4RIA	.NET	MediaPlayer
S20-S27	.NET	OOH4RIA	MediaPlayer

The subjects were supervised by two instructors in order to avoid interaction bias. After each group of tasks, the students were asked to rate their experience regarding the method used.

4.2.1 Variables. Given the research questions presented in Section 4, we have defined the following Independent (experimentally manipulated) Variables (IV) or factors:

- Meth: Method, a categorical variable with two levels: code-centric and MDD. It is important to note that, in this experiment, when we refer to method, we are in fact talking about a compound variable (method*tool), due to the coupling of these two variables in our experimental settings.
- App: Application, a categorical variable with two possible values: Petstore and Mediaplayer

We can characterize Meth as a fixed factor (since it includes the two methods we are interested in testing), while App can be regarded as a random factor (since these two applications are just two examples of the application population). Furthermore, as we have previously mentioned, we have treated the observations of subjects working with

each method as independent, which makes of our design a two-way mixed inter-subject design.

The dependent (measurable) variables (DV) have been defined based on a well-known Method Adoption Model [44], which is rooted in sound evidence from related fields [43], and has been further refined in [35]. According to this model, performance of developers is determined by their actual -objective- effectiveness and efficiency while using the method. Satisfaction, in turn, is determined by their perceived usefulness and their perceived ease of use while using the method. The definition of these variables as they are to be understood in the context of our study is as follows:

- Actual Effectiveness (AEffv). It represents the number of modifications both syntactically and semantically correct carried out by the subjects. Semantically correct modifications not only preserve the executability of the program (correct syntax) but they also fulfill the modification requirement. In our study the corresponding variable has been defined as a ratio variable whose value ranges from 0 (the subject did not carry out any task correctly) to 10 (all the tasks were successfully completed).
- Actual Efficiency (AEffc). It represents the tasks carried out per second. In our study it is defined as a ratio scale. The derived measure associated is calculated as AEffv divided by time (in hours).
- Perceived Usefulness (PU). It represents the extent to which a developer believes that using the method will enhance her job performance. In our study it has been measured through four measures:
  - Perceived Effectiveness (PEffv): An interval variable, measured through a 5-point questionnaire item that represents the subjective percentage of tasks that the subject thinks she correctly addressed.
  - Perceived Certainty (PCert): An interval variable, measured through a 5-point questionnaire item that represents the subjective certainty of the subject with respect to the correction of the proposed modifications.

  - Perceived Stability (PStab): An interval variable, measured through a 5-point questionnaire item that represents the subjective opinion of the subjects with respect to the proposed modifications not having any collateral or unexpected effect in the application (e.g. the introduction of new errors).

- Perceived Ease of Use (PEU). It represents the degree to which a developer believes that using a particular method will be free of effort. The easier developers consider the method to be, the more likely they are to adopt it. Again, in our study we have divided this concept into a number of subcomponents:
  - Perceived Efficiency (PEffc): An interval variable, measured through a 7-point questionnaire item that represents the subjective efficiency of the subject while carrying out the maintainability tasks.
  - Perceived Complexity (PCompl): An interval variable, measured through a 7-point questionnaire item that represents the subjective difficulty of carrying out the maintainability tasks.
  - Perceived Learnability (PLearn): An interval variable, measured through a 7-point questionnaire item that represents the subjective opinion of the subjects with respect to how easy it is to learn how to perform maintainability tasks with the method.
  - Perceived Satisfaction (PS): an interval variable based on a semantic-differential Likert scale made up of 11 items, each one being a 7-point item.

This model also establishes the influences among variables. Such influences are represented in Figure A–1 as arrows. Particularly, this model states that it is the subjective experience of developers what determines their intention to adopt a given method, and, eventually, what determines their actual behaviour (be it adopting or rejecting the method).

In this figure (Figure A–1), the variables (clouds), the measures (hexagons) and the influences (arrows) actually put to test by our experiment have been stressed with solid colours and lines, while the parts of the model that have been left out of our experiment are marked with shaded colors and dashed lines. Figure A–1 also reflects the direction

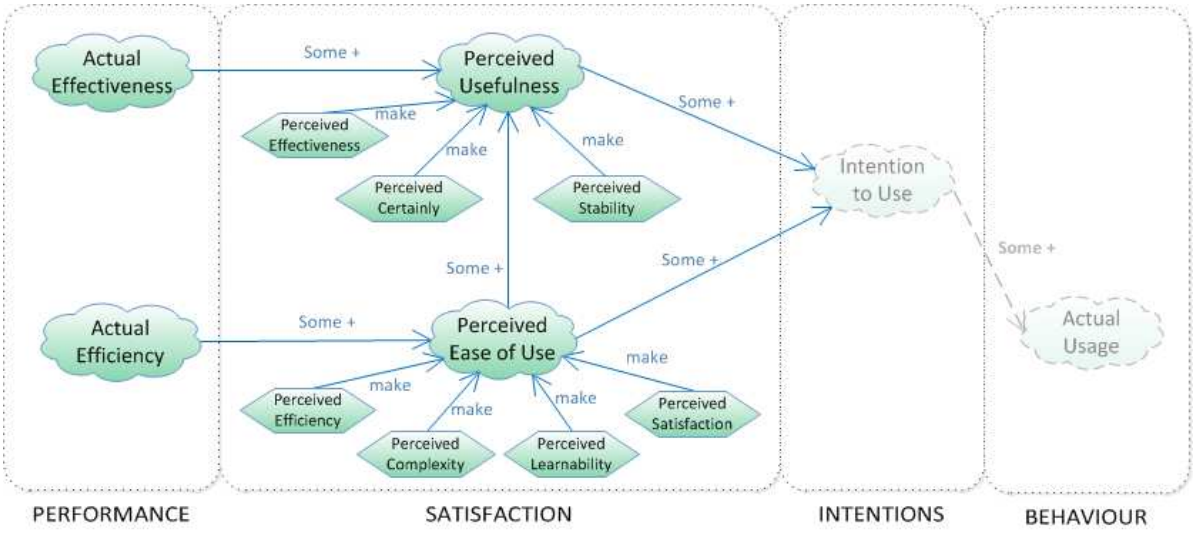


Figura A–1: Theoretical Method Adoption Model components and their associated experimental variables (adapted from [58]).

of the effect: positive (*some* +) or negative (*some* −). In particular, the model states that AEffv positively affects PU, and AEffc positively affects PEU. Additionally, PEU positively affects PU.

4.2.2 Hypotheses. These model dimensions and measures have been used to define the following null and alternative hypotheses, which are based on the research questions presented in Section 4. In the case of the alternative hypotheses, the sense of the inequalities is rooted in the the existing empirical evidence, which was presented in Section 4.1. This notwithstanding, in order to test the hypotheses, two-tailed tests have been used, implying that no direction of the possible inequality has been assumed whatsoever.

- Actual Effectiveness Hypothesis (HAEffv, RQ1).
  - $HAEffv_0$: AEffv(MDE) = AEffv(*code-centric*). The effectiveness of junior software developers while carrying out maintainability tasks over OOH4RIA and over the .NET framework do not significantly differ. This fact holds regardless of the actual application being developed.
  - $HAEffv_A$: AEffv(MDE) > AEffv(*code-centric*).

- Actual Efficiency Hypothesis (HAEffc, RQ1).
  - $HAEffc_0$: AEffc(MDE) = AEffc(*code-centric*). The efficiency of junior software developers while carrying out maintainability tasks with OOH4RIA and with Visual Studio do not significantly differ. This fact holds regardless of the particular application being developed.
  - $HAEffc_A$: AEffc(MDE) > AEffc(*code-centric*).
- Perceived Efficiency Hypothesis (HPEffc, RQ2).
  - $HPEffc_0$: PEffc(MDE) = PEffc(*code-centric*). Performing maintainability tasks with the OOH4RIA method and environment make subjects feel as efficient/inefficient as performing maintainability tasks with Visual Studio and .NET code. This fact holds regardless of the particular application being developed.
  - $HPEffc_A$: PEffc(MDE) > PEffc(*code-centric*).
- Perceived Complexity (HPCompl, RQ2).
  - $HPCompl_0$: PCompl(MDE) = PCompl(*code-centric*). Performing maintainability tasks with OOH4RIA is regarded by subjects as complex as performing them with Visual Studio and .NET code. This fact holds regardless of the particular application being developed.
  - $HPCompl_A$: PCompl(MDE) > PCompl(*code-centric*)[1].
- Perceived Learnability (HPLearn, RQ2).
  - $HPLearn_0$: PLearn(MDE) = PLearn(*code-centric*). Learning how to carry out maintainability tasks with OOH4RIA is regarded by subjects as easy/difficult as learning to carry them out over .NET code. This fact holds regardless of the particular application being developed.
  - $HPLearn_A$: PLearn(MDE) > PLearn(*code-centric*).

[1]This scale item has been reversed so that higher ratings correspond to more positive feelings, that is, to less perceived complexity.

- Perceived Satisfaction (HPSatisf, RQ2).
  - $HPGS_0$: PSatisf(MDE) = PSatisf(*code-centric*). Using the OOH4RIA method and environment to carry out maintainability tasks is regarded by subjects, generally speaking, as satisfying as using Visual Studio to directly manipulate the .NET code. This fact holds regardless of the particular application being developed.
  - $HPGS_A$: PSatisf(MDE) > PSatisf(*code-centric*).
- Perceived Ease of Use (HPEU, RQ2).
  - $HPGEU_0$: PGEU(MDE) = PGEU(*code-centric*). Using the OOH4RIA method and environment to carry out maintainability tasks is regarded by subjects, generally speaking, as easy/difficult as using Visual Studio to directly manipulate the .NET code. This fact holds regardless of the particular application being developed.
  - $HPGEU_A$: PGEU(MDE) > PGEU(*code-centric*).
- Perceived Effectiveness Hypothesis (HPEffv, RQ2).
  - $HPEffv_0$: PEffv(MDE) = PEffv(*code-centric*). Performing maintainability tasks with the OOH4RIA method and environment make subjects feel equally effective/ineffective than performing maintainability tasks with Visual Studio and .NET code. This fact holds regardless of the particular application being developed.
  - $HPEffv_A$: PEffv(MDE) > PEffv(*code-centric*).
- Perceived Certainty Hypothesis (HPCert, RQ2).
  - $HPCert_0$: PCert(MDE) = PCert(*code-centric*). Subjects feel as secure/insecure with the result of carrying out maintainability tasks with the OOH4RIA method as with the result of carrying them out with Visual Studio and .NET code. This fact holds regardless of the particular application being developed.
  - $HPCert_A$: PCert(MDE) > PCert(*code-centric*).

- Perceived Stability Hypothesis (HPStab, RQ2).
  - $HPStab_0$: PStab(MDE) = PStab(*code-centric*). Subjects feel that the result of carrying out maintainability tasks with the OOH4RIA method are as stable as the result of carrying them out with Visual Studio and .NET code. This fact holds regardless of the particular application being developed.
  - $HPStab_A$: PStab(MDE) > PStab(*code-centric*).
- Perceived Usefulness Hypothesis (HPU, RQ2).
  - HPU_0: PU(MDE) = PU(*code-centric*). Subjects feel that OOH4RIA is as useful to carry out maintainability tasks as Visual Studio and the .NET code. This fact holds regardless of the particular application being developed.
  - HPU_A: PU(MDE) > PStab(*code-centric*).

4.2.3 Experiment instrumentation. The materials used in our quasi-experiment, which are included in a replication package [7], have the following structure:

1. Subject confidential agreement
2. Subject instruction sheet.
3. Project Booklet. There are two modalities: A (corresponding to the Petstore application) and B (Mediaplayer application). Whatever the modality, the contents of the booklet include a) a description of the architecture of the application, b) a functional description of the application and, c) an URL where the subjects can download the file with the application. The file contains both an OOH4RIA project and a Visual Studio 2010 Gene project
4. Pre-experiment questionnaire: It includes demographic questions as well as questions about subjects' previous experience with Web application development, Web programming and application modeling.
5. Experimental tasks: Sheet with the two blocks of tasks: five corrective and five perfective. It also includes a sheet where subjects self-report the time that it has taken them to carry out the tasks in that block.

6. Post-experiment questionnaire. Questions to gather the subjective opinions of the subjects regarding each method: PCert, PEffc, PStab, PEffv, PCompl, PLearn and PSatisf.

To make the comparison possible, the OOH4RIA and the .NET experiment materials for both applications and both blocks of tasks were designed to be as similar as possible in terms of both the layout of the information and the information content. Tasks were controlled to be equivalent for both applications.

4.3 Operation and data collection procedures.

As we have previously mentioned, the experiment was carried out during the last session of the Master, once the subjects had received all the needed training on both methods. Previous to this session, a pilot test was run with two subjects having attended the Master the year before. With this pilot, it was decided that a two-hour period was enough to finish the assignment.

The operation phase of the experiment was defined as follows. First, the students filled in the pre-experiment questionnaire with demographic and previous experience data. Then, half of the students received Modality A - Petstore - and half of them received Modality B - Mediaplayer -. In both modalities the user had to perform a set of corrective and perfective changeability tasks.

The instructors indicated to each subject which block they were to carry out with the *code-centric* approach, and which one had to be completed with OOH4RIA.

After completing both sets of tasks, the subjects were asked to complete a post-experiment questionnaire, where they had to subjectively rate their efficiency and effectiveness with each method, as well as the perceived certainty, stability, complexity, learnability and satisfaction of their maintainability work.

To maintain the comparability of the data collected during the experiment, no feedback was given to the subjects on their performance with the tasks. We also controlled that no interaction between participants occurred.

The performance-based variables (AEffc and AEffv) were manually calculated by one of the instructors, based on the code and the OOH4RIA projects that the students handed in. Since the assignments were use to rate the students, all students completed them, so there are no missing values to be taken care of. The time it took them to finish each type of task was self-reported through a form. The subjective variables were recorded with a set of online questionnaires. The tool used to manage such questionnaires is Qualtrics [8]. For the subjective measures, some students did leave some of the questions blank. In such cases, we have followed the strategy of leaving those observations out of the analysis. For this reason, the degrees of freedom of the subjective hypotheses may slightly vary from hypothesis to hypothesis.

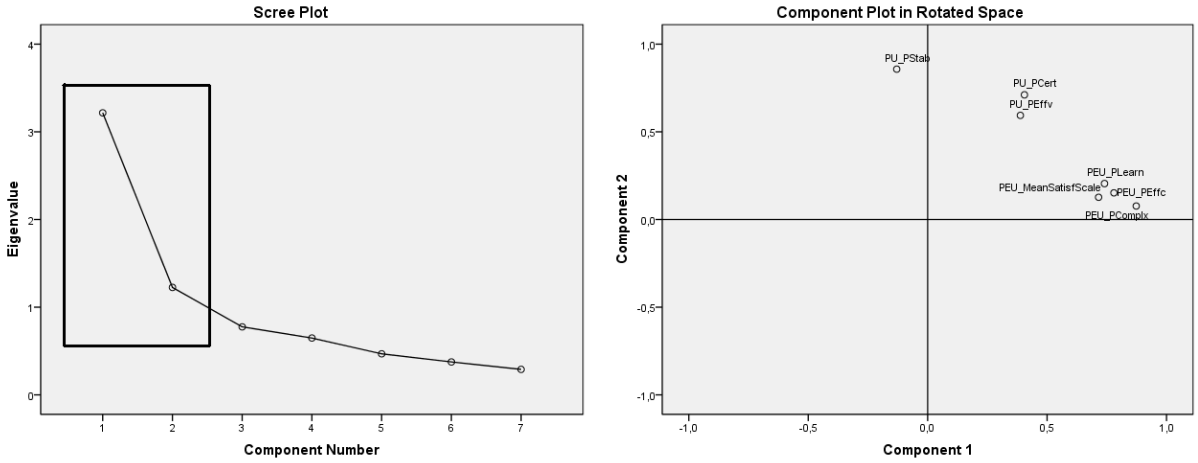
5. DATA ANALYSIS AND INTERPRETATION OF RESULTS

All the analyses were performed with the PASW (Predictive Analytics SoftWare) package, v18 [47].

5.1 Theoretical Model Validation. In order to check whether the grouping of the satisfaction variables into two main constructs (PEU and PU), suggested by the theoretical model (see Fig. A–1), actually fitted our data, we have performed a Principal Component Analysis (PCA).

The first step of such analysis is to determine the number of components. The screen plot presented in Fig. A–2 (a) helps us to determine that the optimal number of components to be extracted is two (see the steep slope in Figure A–2 (a)), while the components on the shallow slope contribute little to the solution.

Then, the rotated component matrix (varimax rotation) has helped us in determining which variables belong to each component. In the scatterplot matrix of Figure A–2 (b) we can observe how all measures associated with PEU variable in the theoretical model form a cluster, while all the measures related with PU form a second cluster. Therefore we can conclude that our data fits well with the variable grouping suggested by the theoretical model.



(a) Scree Plot of initial solution.

(b) The scatterplot matrix.

Figura A–2: Plots of Factor Analysis.

5.2 Reliability of the measurement instruments. Prior to the assessment of the hypotheses, we checked the reliability of the scales in the context of our experimental settings. We applied the Cronbach's Alpha test, which revealed the following results:

- For the PSatisf scale (a semantic-differential scale made up of eleven items), all the items showed a correlation higher than 0,3 with the general construct, while the global Cronbach's Alpha of 0,89, giving proof of high internal consistency among the PSatisf items. It is therefore meaningful to calculate the mean, in order to use it as a global rating of PSatisf.
- For the PU scale (made up of three items: PEffv, PCert and PStab, see Fig. A–1), although all the items showed a correlation higher than 0,3 with the general construct, the Cronbach's Alpha remained rather low ($alpha = 0{,}676$), giving proof of low internal consistency among the items.
- For the PEU scale (made up of four items: PSatisf, PCompl, PLearn and PEffc, see Fig. A–1) the Cronbach's Alpha showed a moderate degree of reliability ($\alpha = 0{,}764$), with all items showing an item-total correlation higher than 0,3. This internal consistency also makes possible to calculate a global PEU value (PEU-Total) by averaging scorings for the items.

The remaining subjective measures were single-item scales.

5.3 Descriptive statistics.

Table A–2 summarizes the means (Mean) and Standard Deviations (SD) for each variable. Both the tasks (corrective and perfective) used to measure efficiency and effectiveness, and the subjective scales, are part of the replication package [7]. The whole questionnaires are presented in Appendix A.A

Tabla A–2: Descriptive statistics

Variables	**.NET**		**OOH4RIA**		**Petstore**		**Mplayer**	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
AEffv	3,33	1,11	4,22	0,89	3,92	1,02	3,64	1,16
AEffc	3,84	2,07	12,34	6,21	9,73	7,34	6,56	4,75
PU-PEffv	3,74	0,91	4,23	0,87	4,21	1,03	3,81	0,80
PU-PCert	3,33	0,78	3,74	0,81	3,73	0,72	3,36	0,87
PU-PStab	3,26	0,76	3,07	1,0	3,46	0,58	2,89	1,03
PU-Total	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
PEU-PEffc	4,57	1,16	5,37	1,21	5,00	1,33	4,96	1,20
PEU-PLearn	4,27	1,37	4,62	1,52	4,84	1,49	4,07	1,33
PEU-PCompl	3,96	1,40	4,88	1,42	4,68	1,40	4,18	1,52
PEU-PSatisf	4,63	0,96	4,56	0,94	4,69	1,00	4,51	0,88
PEU-Total	4,35	0,88	4,83	1,05	4,77	1,10	4,43	0,86

(NA: Not applicable)

Next we present the analyses of the different hypotheses. It is important to note that, since ANOVA presents a high degree of robustness with respect to both ordinality and non-normality of the scale [46], treating all the scales associated with the different subjective variables as interval measures for the refutation of the hypotheses derived from RQ1 and RQ2 does not pose an important threat to the conclusion validity of the study. This is also consistent with the general opinion that Likert scales that are defined by means of sufficient (typically 7 or 9-point) symmetric and equidistant Likert items approximate an interval-level measurement. Furthermore, treating the scale as interval can be beneficial, because otherwise some valuable information (the 'distance' between opinions) could be lost [46].

5.4 RQ1: Performance of treatments.

As it was depicted in Figure A–1, performance is made up of two components: actual efficiency and actual effectiveness.

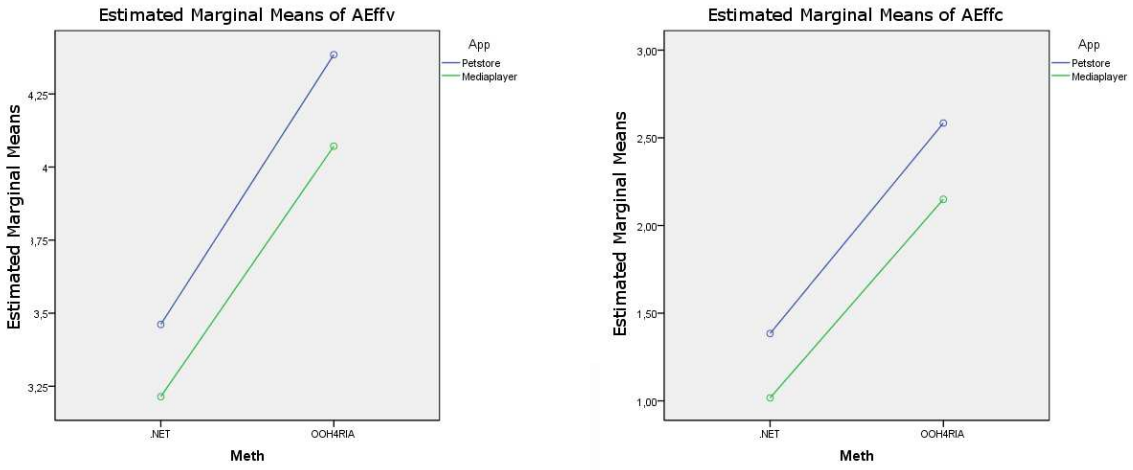
For the refutation of all the hypotheses related to performance we have applied a two-way mixed design ANOVA ($\alpha = 0{,}05$), in which App is a random factor, Meth is a fixed factor and the different components of performance (AEffv and AEffc) are the DVs. The fact that the number of observations per cell (Meth*App combination) is as equal as possible contributes to the robustness of these analyses.

To test the **HAEffv hypothesis** (concerning the existence of significant differences in the actual effectiveness of the two methods), we have first calculated the Levene's statistic. The result ($F(3, 50) = 0{,}919; p = 0{,}439$) allows us to assume homogeneity of variance and go on with the ANOVA analysis.

The results show that the interaction Meth*App is not significant ($F(1, 50) = 0{,}014; p > 0{,}05$). We can then safely examine the main effects of the two independent variables (App and Meth) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the subject's effectiveness when using OOH4RIA ($M = 4{,}22; SD = 0{,}89$) is significantly greater than the effectiveness of subjects using .NET code ($M = 3{,}33; SD = 1{,}11$): ($F(1, 50) = 729; p < 0{,}05$). Furthermore the effect size is very high. The partial Eta squared is 0,999, which means that the factor Meth by itself accounts for 99,9 % of the overall (effect+error) variance. They also show that subject's effectiveness is slightly greater with Petstore ($M = 3{,}92; SD = 1{,}02$) than with Mediaplayer ($M = 3{,}64; SD = 1{,}16$). However, this difference is not significant ($F(1, 50) = 72{,}25; p > 0{,}05$). This means that the differences in Effectiveness are significantly affected by the method used, regardless of the particular application being developed.

We can observe these results graphically in Figure A–3 (a). The fact that the particular application is not significant is reflected in the lines being quite close to each other. The

Meth variable influence is reflected in the slope of the lines. Finally, the lack of significance of the Meth*App interaction is reflected in the lines being more or less parallel (all of them showing the same tendency with each method). The same graphical clues hold for the rest of the graphics.



(a) Effectiveness.

(b) Efficiency.

Figura A–3: Actual Performance.

Then, we have tested the **HAEffc hypothesis** (in tasks per hour). The Levene's statistic ($F(3,50) = 5{,}788; p = 0{,}002$) indicates a violation of the assumption.

In order to overcome this problem, we have applied a logaritmic transformation of the data: we consider the variable LAEffc now as Ln(AEffc). With the transformed variable, the Levene's coefficient is $F(3,50) = 0{,}421; p > 0{,}05$, which allows us to accept the assumption and continue with the analysis.

The results show that the interaction Meth*App is not significant ($F(1,50) = 0{,}042; p > 0{,}05; \beta = 0,945$). We can then safely examine the main effects of the two independent variables (App and Meth) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the Ln of the subject's efficiency when using OOH4RIA ($M = 12{,}34; SD = 6{,}21$) is significantly greater than the Ln of the subject's efficiency when acting on the .NET code ($M = 3{,}84; SD = 2{,}07$): ($F(1,50) = 1197{,}29; p < 0{,}05$). Again, the effect size is very high. The partial Eta squared

is 0,999, which means that the factor Meth by itself accounts for 99,9 % of the overall (effect+error) variance. They also show that subject's efficiency is not significantly different between Petstore ($M = 9{,}73; SD = 7{,}34$) and Mediaplayer ($M = 6{,}56; SD = 4{,}75$): ($F(1, 50) = 141{,}42, p > 0{,}05$). This means that the differences in efficiency are significantly affected by the method used, regardless of the particular application being developed. We can observe these results graphically in Figure A–3 (b).

5.5 RQ2: Satisfaction of treatments.

As presented in Figure A–1, satisfaction is made up of two components: PU and PEU.

On the one hand, PU is made up of three components: perceived certainty, perceived stability and perceived efficiency. They make up a three-item scale that, however, according to the data gathered in our experiment, has a rather low level of reliability (see Section 5.2). For this reason, we have finally left out of our experiment the HPU hypothesis, since we lack a reliable enough instrument to measure such construct, and we have centered on HPEffv, HPCert and HPStab.

PEU, on the other hand, is made up of four different components: perceived complexity, perceived learnability, perceived satisfaction and perceived efficiency. Since the four items make up a scale with sufficient internal reliability, we can safely assume that all these constructs can be used to calculate their mean as the global value for the PEU of the method. Therefore, all five hypotheses (HPEffc, HPLearn, HPCompl, HPSatisf and HPEU) can be tested.

Again, for the refutation of all the hypotheses related to both PEU and PU we have applied a two-way mixed design ANOVA ($\alpha = 0{,}05$), in which App is a random factor, Meth is a fixed factor and the different components of PU are the DVs. The fact that the number of observations per cell (Meth*App combination) is as equal as possible contributes to the robustness of these analyses. Previous to the analyses, we reversed some of the questionnaire items so that higher ranks always corresponded to more positive feelings (more learnable, less complex, more efficient, and so on).

Next, we present the data analyses.

5.5.1 Perceived usefulness. To test the **HPEffv hypothesis** (concerning the existence of significant differences in the percentage of tasks that the subject thinks she correctly addressed with the two methods), we have calculated the Levene's statistic to check whether the population variances are equal. The result ($F(3, 41) = 0{,}66; p > 0{,}05$) allows us to assume homogeneity of variance.

The results show that the interaction Meth*App is not significant ($F(1, 41) = 0{,}26; p > 0{,}05$). We can then safely examine the main effects of the two IV (App and Meth) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the subject's perceived effectiveness when using .NET ($M = 3{,}74; SD = 0{,}91$) is slightly lower than the effectiveness of subjects working over OOH4RIA ($M = 4{,}23; SD = 0{,}87$). However, this difference is not significant: ($F(1, 41) = 14{,}74; p > 0{,}05$). The results also show that subject's perceived effectiveness is slightly greater with Petstore ($M = 4{,}21; SD = 1{,}03$) than with Mediaplayer ($M = 3{,}81; SD = 0{,}80$), although this difference, again, is not significant: ($F(1, 41) = 9{,}62; p > 0{,}05$). This means that the differences in Perceived Effectiveness are not significantly affected neither by the method used nor by the application implemented. These results can be graphically observed in Figure A–4.

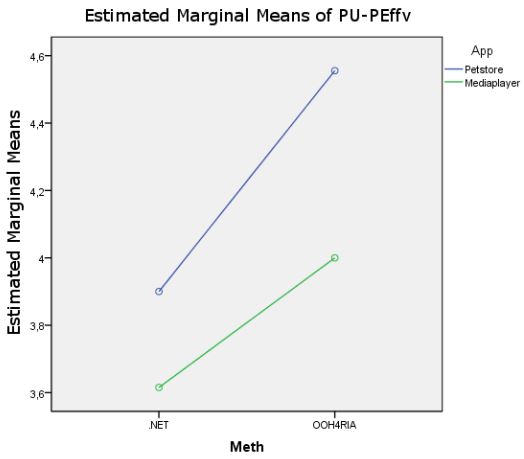
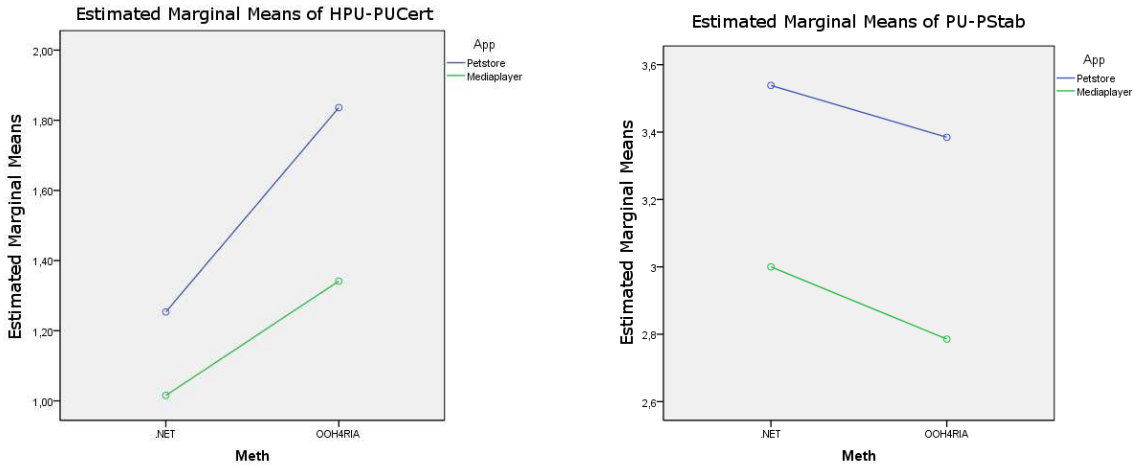


Figura A–4: Perceived Effectiveness

Regarding the **HPCert hypothesis** (concerning the existence of significant differences of subject's secure/insecure with the result of carrying out maintainability tasks with the two methods), the Levene's statistic ($F(3,50) = 3{,}02; p < 0{,}05$) indicates a violation of the assumption. In order to overcome this problem, we have applied a logaritmic transformation of the data: we consider the variable LPCert now as Ln(PCert). Again the result of the Levene's statistic ($F(3,50) = 2{,}97; p < 0{,}05$) indicates a violation of homogeneity of error variance. For this reason, another logaritmic transformation has been applied, and now LPCert equals $LnGamma$(PCert). With the transformed variable, the Levene's statistic is $F(3,50) = 2{,}08; p > 0{,}05$, which allows us to accept the assumption of homogeneity of variance and continue with the analysis.

The results show that interaction Meth*App is not significant ($F(1,50) = 0{,}31, p > 0{,}05$). Therefore we can safely examine the main effects of the two IV (Meth and App) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the *LnGamma* of subject's perceived certainty with respect to the correction of the proposed modifications when using OOH4RIA ($M = 3{,}74; SD = 0{,}81$) is slightly higher than the *LnGamma* of subject's perceived certainty when using .NET ($M = 3{,}33; SD = 0{,}78$): ($F(1,50) = 12{,}55; p > 0{,}05$), but the main effect of method does not attain significance. Also, the *LnGamma* of subject's perceived certainty is slightly higher with Petstore ($M = 3{,}73; SD = 0{,}72$) than with Mediaplayer ($M = 3{,}36; SD = 0{,}87$): ($F(1,50) = 8{,}16; p > 0{,}05$), although this difference, again, is not significant. This means that the differences in perceived certainty is not affected significantly neither by the methodology nor by the application. We can observe these results graphically in Figure A–5 (a).

As for the **HPStab hypothesis** (concerning the existence of significant differences of perceived stability of the subjects with respect to the proposed modifications not having any collateral or unexpected effect in the application with the two methods), the Levene's statistic ($F(3,50) = 2{,}45; p > 0{,}05$) allows us to assume homogeneity of variance.



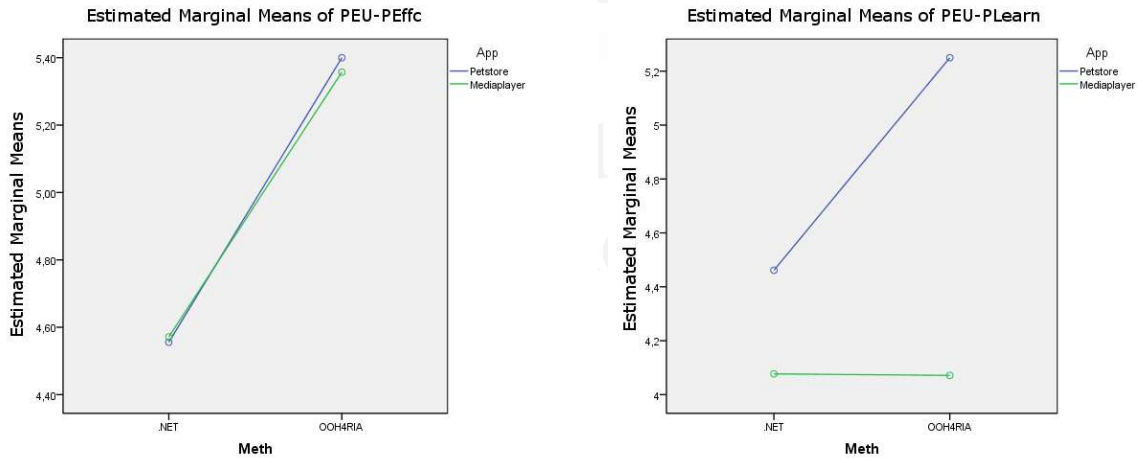
(a) Perceived Certainty. (b) Perceived Stability.

Figura A–5: Certainty and Stability in Perceived Usefulness.

The results show that the interaction Meth*App is not significant ($F(1,50) = 0{,}02; p > 0{,}05$). We can then safely examine the main effects of the two IV (Meth and App) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the subjects perceived stability of proposed modifications with .NET code ($M = 3{,}26; SD = 0{,}76$) is slightly superior to the subjects' perceived stability when using OOH4RIA ($M = 3{,}07; SD = 1{,}00$), although not significantly: ($F(1,50) = 37{,}10; p > 0{,}05$). The results also show that the perceived stability of subjects is significantly greater with Petstore ($M = 3{,}46; SD = 0{,}58$) than with Mediaplayer ($M = 2{,}86; SD = 1{,}03$): ($F(1,50) = 354{,}124; p < 0{,}05$). Furthermore, the App effect size is very high. The partial Eta squared is 0,997, which means that the factor App by itself is able to account for 99,7 % of the overall (effect+error) variance. This means that the differences in perceived stability are significantly affected by the particular application being developed, regardless of the method used. We can observe these results graphically in Figure A–5 (b).

5.5.2 Perceived ease of use. To test the **HPEffc hypothesis** (concerning the existence of significant differences of the subjective performance of the subject while performing the maintainability tasks with the two method), the Levene's statistic ($F(3,43) = 1{,}93; p = 0{,}14$) allows us to assume homogeneity of variance.

The results show that the interaction Meth*App is not significant ($F = 0{,}007, p > 0{,}05$). We can then safely examine the main effects of the two IV (App and Meth) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the subjects' perceived efficiency when using OOH4RIA ($M = 5{,}37; SD = 1{,}21$) is significantly superior to the perceived efficiency of subjects using .NET ($M = 4{,}57; SD = 1{,}16$): ($F = 770{,}438; p < 0{,}05$). Furthermore the effect size is very high. The partial Eta squared is 0,999, which means that the factor Meth by itself accounts for 99,9 % of the overall (effect+error) variance. Also, subjects' perceived efficiency is slightly higher with Petstore ($M = 5{,}00; SD = 1{,}33$) than with Mediaplayer ($M = 4{,}96; SD = 1{,}20$): ($F = 0{,}211; p > 0{,}05$), although not significantly. This means that the differences in perceived efficiency are significantly affected by the method used, regardless of the particular application being developed. The results can be graphically seen in Figure A–6 (a).



(a) Perceived Efficiency.

(b) Perceived Learnability.

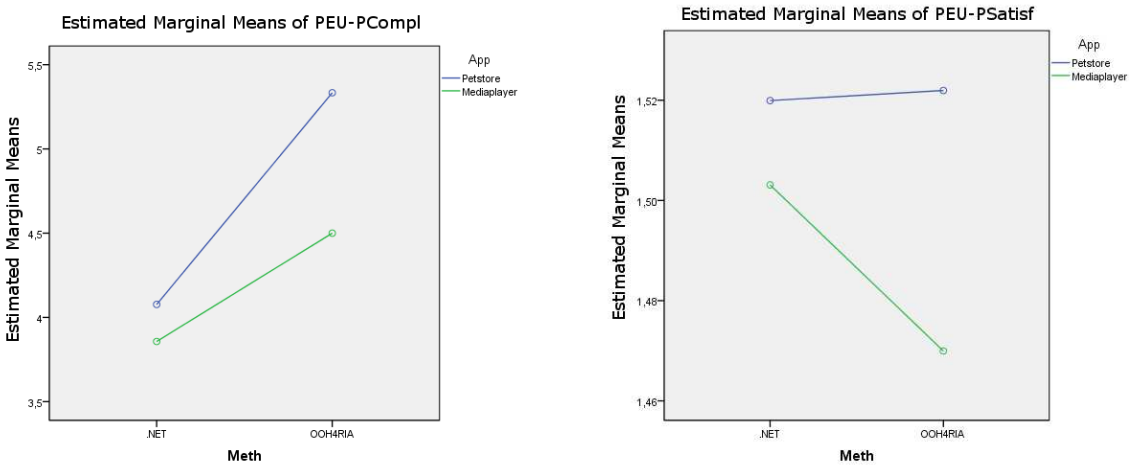
Figura A–6: Efficiency and Learnability in Perceived Ease of Use.

To test the **HPLearn** hypothesis (concerning the existence of significant differences of subjects' perceived learnability while performing the maintainability tasks with the two methods), the Levene's statistic ($F(3, 48) = 0{,}12; p => 0{,}05$) allows us assume homogeneity of variance.

The results show that the interaction Meth*App is not significant ($F(1,48) = 1{,}028; p > 0{,}05$). We can then safely examine the main effects of the two IV (Meth and App) on these means without needing to qualify the results by the existence of a significant interaction. For the Meth variable, the subjects' perception of the method learnability with OOH4RIA ($M = 4{,}62; SD = 1{,}52$) is slightly better than the perceived learnability with .NET code ($M = 4{,}27; SD = 1{,}37$), but this difference is not significant: ($F(1,48) = 0{,}97; p > 0{,}05$). The results also show that the subjects' learnability is greater with Petstore ($M = 4{,}84; SD = 1{,}49$) than with Mediaplayer ($M = 4{,}07; SD = 1{,}33$), although not significantly: ($F(1,48) = 3{,}876; p > 0{,}05$), . This means that the differences in perceived learnability are not affected significantly neither by the methodology nor by the application. We can observe these results graphically in Figure A–6 (b).

Regarding the **HPCompl hypothesis** (concerning the existence of significant differences of subjects's perceived level of difficulty of carrying out the maintainability tasks with each method), the Levene's statistic ($F(3,49) = 1{,}36; p > 0{,}05$) allows us to assume homogeneity of error variance.

The results show that the interaction Meth*App is not significant ($F(1,49) = 0{,}63; p > 0{,}05$). We can then safely examine the main effects of the two IV (App and Meth). For the Meth variable, the subject's perceived complexity of performing maintainability tasks with OOH4RIA ($M = 4{,}88; SD = 1{,}42$) is slightly lower (more positive feelings) than the perceived complexity of performing maintainability tasks over .NET code ($M = 3{,}96; SD = 1{,}40$), although not significantly: ($F(1,49) = 9{,}58; p > 0{,}05$). The results also show that the subject's perceived complexity is slightly lower with Petstore ($M = 4{,}68; SD = 1{,}41$) than with Mediaplayer ($M = 4{,}18; SD = 1{,}52$), although, again, not significantly: ($F(1,49) = 2{,}95; p > 0{,}05$). This means that the differences in perceived complexity when subjects carry out maintainability tasks are not affected significantly neither by the methodology nor by the application. These results are graphically depicted in Fig. A–7(a).



(a) Perceived Complexity.

(b) Perceived Satisfaction.

Figura A–7: Complexity and Satisfaction in Perceived Ease of Use.

To test the **HSatisf hypothesis**, related to the the existence of significant differences of subjects's perceived satisfaction while performing the maintainability tasks with the two methods, the Levene's statistic ($F(3,50) = 4{,}06; p < 0{,}05$) shows a violation of homogeneity of variance. To overcome this problem, we have applied a logaritmic transformation of the data: we consider the variable LPSatisf now as Ln(PSatisf). With the transformed variable, the Levene's statistic is $F(3,50) = 2{,}19, p > 0{,}05$, which allows us to accept the assumption and continue with the analysis.

The results show that the interaction Meth*App is not significant ($F(1,50) = 0{,}08, p > 0{,}05$), which allows us to safely examine the main effects of the two IV (App and Meth). For the Meth variable, the Ln of the subjects' perceived satisfaction with OOH4RIA ($M = 4{,}56; SD = 0{,}94$) is slightly lower than the Ln of the perceived satisfaction with .NET code ($M = 4{,}63; SD = 0{,}96$), although not significantly: ($F(1,50) = 0{,}783; p > 0{,}05; \beta = 0{,}932$). The results also show that the Ln of subjects' perceived satisfaction is slightly superior with Petstore ($M = 4{,}69; SD = 1{,}00$) than with Mediaplayer ($M = 4{,}51; SD = 0{,}89$), although, again, not significantly: ($F(1,50) = 3{,}83; p > 0{,}05$).

This means that the differences in perceived satisfaction are not affected significantly neither by the methodology nor by the application. These results can be graphically seen in Fig. A–7 (b).

Last but not least, to test the **HPEU-Total** hypothesis (concerning the existence of significant differences in the global ease of use of the methods while performing maintainability tasks), the Levene's statistic ($F(3,50) = 3{,}26; p < 0{,}05$) indicates a violation of the assumption. In order to overcome this problem, we have applied a logaritmic transformation of the data: we consider the variable LPEU-Total now as Ln(PEU-Total). The result ($F(3,50) = 2{,}29; p > 0{,}05$) allows us to accept the assumption and continue with the analysis.

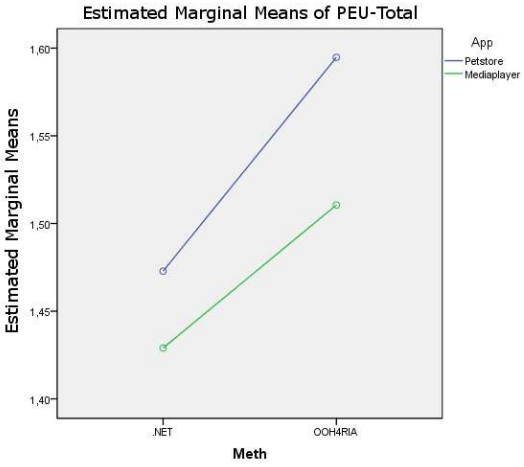


Figura A–8: Global Perceived Ease of Use

The results of the two-way ANOVA analysis show that the interaction among variables (Meth*App) is not significant ($F(1,50) = 0{,}11; p > 0{,}05$). For the Meth variable, subject's perceived global ease of use with OOH4RIA ($M = 4{,}83; SD = 1{,}05$) is slightly superior to the perceived ease of use with .NET code ($M = 4{,}35; SD = 0{,}88$), although not significantly: ($F(1,50) = 25{,}42; p > 0{,}05$). The results also show the subjects's perceived ease of use is slightly superior with Petstore ($M = 4{,}77; SD = 1{,}10$) than with Mediaplayer ($M = 4{,}43; SD = 0{,}86$), although, again, not significantly ($F(1,50) = 10{,}06; p > 0{,}05$). This means that the differences in perceived global ease of use are not affected significantly

neither by the methodology nor by the particular application being developed. These results can be graphically seen in Figure A–8.

A summary of all these results can be seen in Table 3.

Tabla A–3: Data Analysis Results: Summary

Variables	Meth*App	Meth	App	Effect Size
AEffv	N	Y	N	High
AEffc	N	Y	N	High
PU-PEffv	N	N	N	NA
PU-PCert	N	N	N	NA
PU-PStab	N	N	Y	High
PU-Total	NA	NA	NA	NA
PEU-PEffc	N	Y	N	High
PEU-PLearn	N	N	N	NA
PEU-PCompl	N	N	N	NA
PEU-PSatisf	N	N	N	NA
PEU-Total	N	N	N	NA
(NA: Not applicable)				

5.6 Threats to Validity. The analysis of the threats to validity evaluates under which conditions our experiment is applicable and offer benefits, and under which circumstances it might fail [17]. It therefore serves to qualify the experiment results.

Threats to Conclusion Validity refer to issues that affect the ability to draw the correct conclusion about relationships between the treatment and the outcome of the experiment. In our experiment we have performed a sensitivity analysis that assures a power of 0,7 for effect sizes greater than 0,34. The main assumptions of the statistical tests have been checked, and data has been transformed if necessary to fulfill them. Although the consideration of five and seven-point Likert items as interval variables is a matter of controversy, the use of ANOVAs, which present a high degree of robustness with respect to both ordinality and non-normality of the scale [46], significantly reduces this threat to the conclusion validity of the study. The reliability of measures involving scales has been tested, and hypotheses where the scale did not show a high enough degree of reliability have been left out of the study. This notwithstanding, the fact that some of the variables have been measured through a single questionnaire item poses a threat to conclusion

validity that can only be overcome with the definition and validation of standard scales to measure such constructs. To our knowledge extent such scales are not yet available for researchers. The treatments were consistently applied among subjects. However, the fact that the objective results of the experiment were used to grade the students may have introduced a noise in the results (e.g. by making some students nervous about their performance). This notwithstanding, given the fact that all the subjects applied both treatments (methods), we can assume that such effect, if present, has equally affected all the levels of the treatment. Also, this fact diminishes the risk of a random heterogeneity of subjects, since both treatments were applied by the same group of subjects.

Threats to Internal Validity are concerned with the possibility of hidden factors that may compromise the conclusion that is indeed the treatment what causes the differences in outcome. In our experiment all the subjects enrolled in the master had to participate in the experiment (so no selection bias beyond that inherent to quasi-experiments was present). The subjects applied the treatments with different types of task, which diminishes the history risk. The two-hour limit of the experiment is short enough to avoid a learning effect, while the fact that they had to apply two treatments that involved a completely different set of skills limits the maduration effect. Two instructors supervised the whole process in order to disminish the interaction bias. However, there is a threat of ceiling effect that we have tried to control with the pilot test, which gave us confidence to assume that two hours was enough time for the subjects to finish the assignments. Another internal threat is the fact that we did not oblige subjects to answer all the questions in the questionnaire. Also, students self-reported the time it took them to finish the tasks. Tasks were manually corrected by one of the instructors. The risk for subjectivity was controlled by making sure that the application handed in by the subject run as expected after the maintainability work. The fact that they applied the treatments on two different applications also poses a threat to internal validity that has been controlled by using applications of similar complexity, measured in number of conceptual constructs (for the

MDE treatment) and code lines (for the .NET treatment). This latter measure is generally preferred over functional points due to its higher reliability [55].

Threats to Construct Validity refer to the generalization of the result of the experiment to the concept or theory behind the experiment. In this sense, the theoretical model is well defined. A PCA has been performed to check that our data fitted the variable grouping of the theoretical model. However, some of the used measures have not been tested for reliability, and therefore they may have introduced a measurement bias, even if they have been previously widely used in literature. Also, the exam condition may have posed additional stress upon subjects (evaluation apprehension) or, on the contrary, may have caused them to be more efficient and effective than usual because they were being graded. Also, we have used similar size applications, all belonging to the same domain (mono-operation bias). Therefore we cannot generalize the results to applications of different sizes or different domains. Also, the hypothesis of the experiment (that is, a higher maintainability of MDE environments) was quite easy to guess, so students may have felt bound to report less time when using MDE. Anyway, the experiment observers took special care not to disclose this hypothesis to the students. Additionally, the experiment suffers from a restricted generalizability across constructs: we have checked a positive outcome between maintainability and OOH4RIA, but we cannot assure that this does not hamper other quality attributes or any additional characteristic.

Last but not least, **Threats to External Validity** are concerned with generalization of the results to the industrial practice. The subjects are graduate students (M.Sc. students), many of them already working as developers, and therefore true representatives of junior developers, but the small sample used and the fact that these subjects are highly motivated people may not be representative of the population of junior developers. Also, the particular methodologies and languages we have used, despite being broadly used in industry, constitute a limited environment. The applications chosen, although limited in size, are also representatives of the kind of applications that are being developed in industry, but the scope of the maintainability tasks had to be limited due to time constraints, so

there is a risk that different results may happen if bigger applications or different maintainability tasks had been used to perform the experiment. Therefore, this experiment needs to be replicated with different languages, tools, applications, tasks and MDE approaches. For this purpose, the replication package of this experiment can be found at [7].

6. CONCLUSIONS

This study concludes that the OOH4RIA approach improves the actual performance (by means of the increment of the effectiveness and the efficiency) when the subjects carry out maintainability task over .NET applications. The satisfaction, however, doesn't reveal significant differences, although the means slightly favor the MDE approach in all the variables except for perceived stability and perceived satisfaction.

Another contribution of this experiment is the Theoretical Model of Adoption of Methods, that integrates the variables on those that a bigger consent grade exists among the investigating community about its power of prediction of the grade of intention of adoption of a new methodology. Nevertheless, further analysis should be carried out with the stability and complexity measures, as the measures more correlated with those the two main components (PEU and PU).

Finally, in the answers to the open-questions of the post-test, students are in correspondence with the assumption of the MDE approaches. They feel that MDE technologies are easier to use and they allow to carry out the maintainability tasks quickly. They also consider that this methodology is less flexible than other development approaches because the code generated is not always enough to solve all the requirements of the software and at the end, it is necessary to make modifications directly on code. Besides, the subjects sometimes find difficulties when carrying out maintainability task with OOH4RIA methodology; due to the main fact that they have more experience carrying out these tasks directly in the code (curve of learning of MDE technologies).

Our results augment the repository of empirical data comparing maintainability performance and satisfaction of MDE methodologies with respect to traditional *code-centric*

approaches. As in the literature, not everything has been said, and further experimentation is needed to be able to generalize the results to a different population, different methods and languages, different application types, different maintainability tasks or different application sizes. Also, extensive work needs to be done in the matter of defining reliable measures. Last but not least, more data is needed to perform a much more extensive validation of the theoretical model, particularly with respect to the causal relationships inferred from that model.

Acknowlegments

The authors wish to thank their students for taking the time to participate in this empirical study. In addition, we would also like to thank Jose Javier Martínez and Juan Antonio Osuna who contributed to the development of the OOH4RIA Tool. Special thanks to Dania Suárez and Milton García for their help.

REFERENCES

1. *G*power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences.*, Behavior Research Methods 39, 175-191.

2. *A systematic review of quasi-experiments in software engineering*, Softw. Technol. 51, 71-82.

3. *ISO/IEC 9126-1, Software enginering -Product quality - Part 1: Quality model,* 2001

4. *Visual Paradigm for UML 8.0 Community Edition*, http://www.visual-paradigm.com/ (Last update August 16, 2010), 2005.

5. *Métrica y Complejidad del software: ToroMetrics v1.0.*, http://www.moisesdaniel.com/es-/wri/torometrics.htm, 2006.

6. *Integranova M.E.S (Integranova Software Solutions)*, http://www.slideshare.net/mhoubraken/-catalogo-integranova-esp-mayo-2011, 2011.

7. *Modelio open source modeling environment v2.2*, http://www.modelio.org, 2011.

8. *Experiment replication package*, http://www.dlsi.ua.es/ ccachero/labPackages/Maintainability2012.rar, 2012.

9. *Qualtrics online survey software*, http://www.qualtrics.com/, 2012.

10. T Adrian, *C#. NET web developer's guide*, (2002).

11. D Ameller, F Gutiérrez, and J Cabot, *Dealing with non-functional requirements in model-driven development*, (2010).

12. F Budinsky, *Eclipse modeling framework: a developer's guide*, Prentice Hall Ptr, 2004.

13. C Cachero, G Poels, and C Calero, *Towards a Quality-Aware Web Engineering Process*, Proceedings of the 12th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'2007), Citeseer, 2007, pp. 7-16.

14. S Ceri, P Fraternali, and M Matera, *Conceptual modeling of data-intensive Web applications*, Internet Computing, IEEE 6 (2002), no. 4, 20-30.

15. N Chapin, J E Hale, K M Khan, J F Ramil, and W G Tan, *Types of software evolution and software maintenance*, Journal of Software Maintenance and Evolution: Research and Practice 13 (2001), no. 1, 3-30.

16. J. Cohen, *Statistical power analysis for the behavioral sciences*, Lawrence Erlbaum, 1988.

17. D Coleman, D Ash, B Lowther, and P Oman, *Using metrics to evaluate software system maintainability*, Computer 27 (2002), no. 8, 44-49.

18. T D Cook, D T Campbell, and A Day, *Quasi-experimentation: Design & analysis issues for field settings*, Houghton Miffin Boston, 1979.

19. T Dyba, B A Kitchenham, and M Jorgensen, *Evidence-based software engineering for practitioners*, Software, IEEE 22 (2005), no. 1, 58-65.

20. X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, *Cans: composable, adaptive network services infrastructure*, Proceedings of the 3rd conference on USENIX Symposium on Internet Technologies and Systems-Volume 3, USENIX Association, 2001, pp. 12-12.

21. R.H. Glitho, F. Khendek, and A. De Marco, *Creating value added services in internet telephony: an overview and a case study on a high-level service creation environment*, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 33 (2003), no. 4, 446-457.

22. R C Gronback, Eclipse Modeling Project: *A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley Professional, 2009.

23. H Gustavsson, B Lings, B Lundell, A Mattsson, and M Beekveld, *Integrating proprietary and open-source tool chains through horizontal interchange of XMI models*, Software Maintenance,2007. ICSM 2007. IEEE International Conference on, IEEE, 2007, pp. 521-522.

24. S Hanselman and D Verma, *Visual Studio 2010 SP1 for Web Developers*, MSDN Magazine-Louisville (2011).

25. W Heijstek and M R V Chaudron, E*mpirical investigations of model size, complexity and effort in a large scale, distributed model driven development process*, Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on, IEEE, 2009, pp. 113-120.

26. J Hutchinson, J Whittle, M Rouncefield, and S Kristoffersen, *Empirical assessment of MDE in industry*, Proceeding of the 33rd International Conference on Software Engineering, ACM, ACM, 2011, pp. 471-480.

27. J Hutchinson, J Whittle, M Rouncefield, *Empirical assessment of mde in industry*, Interpretation A Journal Of Bible And Theology (2011), 471-480, ¡m:note/¿.

28. Icinetic TIC S.L., *RADARC (Rapid Application Development Architecture)*, http://www.-radarc.net/, 2012.

29. IEEE Std.1219-1998, *IEEE Standard for Software Maintance*, 1998.

30. B Kitchenham, D Budgen, P Brereton, M Turner, S Charters, and S Linkman, *Large-scale software engineering questions-expert opinion or empirical evidence?*, Software, IET 1 (2007), no. 5, 161-171.

31. M Linaje, J Preciado, and F Sánchez-Figueroa, *A method for model based design of rich internet application interactive user interfaces*, Web Engineering (2007), 226-241.

32. E.D. López, M González, M López, and E.L. Iduñate, *Proceso de Desarrollo de Software Mediante Herramientas MDA*, Revista Iberoamericana de Sistemas, Cibernética e Informática 3 (2006), no. 2, 6-10.

33. Y Martínez, C Cachero, M Matera, S Abrahao, and S Luján, *Impact of MDE Approaches on the Maintainability of Web Applications: An Experimental Evaluation*, Conceptual Modeling-Er 2011: 30th International Conference on Conceptual Modeling, Brussels, Belgium, October 31-November 3, 2011. Proceedings, vol. 6998, Springer-Verlag New York Inc, Springer-Verlag New York Inc, 2011, pp. 233-246.

34. Y Martínez, C Cachero, and S Meliá, *Evidencia empírica sobre mejoras en productividad y calidad mediante el uso de aproximaciones MDD: un mapeo sistemático de la literatura.*, 2011.

35. Y Martínez, C Cachero, and S Meliá, *Mdd vs. traditional software deveopment: A practitioner's subjective perspective*, Softw. Technol. (2012).

36. S Meliá, J Gómez, S Pérez, and O Díaz, *A model-driven development for GWT-based Rich Internet Applications with OOH4RIA*, Web Engineering, 2008. ICWE'08. Eighth International Conference on, IEEE, 2008, pp. 13-23.

37. S Meliá, J J Martínez, S Mira, J Osuna, and J Gómez, *An Eclipse Plug-in for Model-Driven Development of Rich Internet Applications*, Web Engineering (2010), 514-517.

38. N Mellegård and M Staron, *Improving Efficiency of Change Impact Assessment Using Graphical Requirement Specifications: An Experiment*, Product-Focused Software Process Improvement (2010), 336-350.

39. S J Mellor, T Clark, and T Futagami, *Model-driven development: guest editors' introduction.*, IEEE software 20 (2003), no. 5, 14-18.

40. T Mens and P Van Gorp, *A taxonomy of model transformation*, Electronic Notes in Theoretical Computer Science 152 (2006), 125-142.

41. P Mohagheghi and R Conradi, *An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes*, Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on, IEEE, 2004, pp. 7-16.

42. P Mohagheghi and V Dehlen, *Where is the proof? - A review of experiences from applying MDE in industry*, European Conference on Model Driven Architecture-Foundations and Applications (ECMDA 2008), Springer, 2008, pp. 432-443.

43. D L Moody, *Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models* (PhD Thesis), Melbourne, Australia: Department Of Information Systems, University of Melbourne (2001).

44. D L Moody, *The method evaluation model: a theoretical model for validating information systems design methods*, 11th European Conference on Information Systems (ECIS 2003), Naples, Italy, Citeseer, 2003, p. 79.

45. J Muñoz and V Pelechano, *MDA vs Factorías de Software*, Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM 2005) (2005), 1.

46. G. Norman, *Likert scales, levels of measurement and the laws of statistics*, Advances in health sciences education 15 (2010), no. 5, 625-632.

47. M.J. Norusis et al., *Pasw statistics 18 guide to data analysis*, Prentice Hall Press, 2010.

48. D E Perry, A A Porter, and L G Votta, *Empirical studies of software engineering: a roadmap*, Proceedings of the conference on The future of Software engineering, ACM, 2000, pp. 345-355.

49. J C Preciado, M Linaje, R Morales-Chaparro, F Sanchez-Figueroa, G Zhang, C Kroiÿ, and N Koch, D*esigning rich internet applications combining uwe and rux-method*, Web Engineering, 2008. ICWE'08. Eighth International Conference on, IEEE, 2008, pp. 148-154.

50. F Ruiz and M Polo, *Mantenimiento del Software*, Master en ingeniería del software, 2007, p. 109.

51. SUN Microsystems, *Java Pet Store Sample Application*, Blueprints Online, 2010.

52. University of Alicante, *Master in Web Applications Development site*, http://www.eps.-ua.es/masterweb, 2010

53. M Urbieta, G Rossi, J Ginzburg, and D Schwabe, *Designing the interface of rich internet applications*, Web Conference, 2007. LA-WEB 2007. Latin American, IEEE, 2007, pp. 144-153.

54. A Vallecillo, N Koch, C Cachero, S Comai, P Fraternali, I Garrigós, J Gómez, G Kappel, A Knapp, M Matera, and Others, *MDWEnet: A practical approach to achieving interoperability of model-driven Web engineering methods*, Workshop Proc. of 7th Int. Conf. on Web Engineering (ICWE'07), Italy, Citeseer, 2007.

55. F Valverde and O Pastor, *Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach*, Web Information Systems Engineering-WISE 2009 (2009), 131-144.

56. C Wohlin, P Runeson, and M Höst, *Experimentation in software engineering: an introduction*, Springer Netherlands, 2000.

57. M V Zelkowitz, An update to experimental models for validating computer technology, Journal of Systems and Software 82 (2009), no. 3, 373-376.

Appendix A.A

A.A.1. Actual Effectiveness

Sum of the number of maintainability tasks, corrective and perfective, carried out by students does the solution what it was asked to do.

	1	**2**	**3**	**4**	**5**
Corrective Maintainability Task					
Perfective Maintainability Task					

A.A.2. Actual Efficiency

The actual efficiency, is also calculated manually, by one of the instructors as the *Actual Effectiveness* divide among the time. Also, the time it took them to finish each type of task was self-reported through a following form:

Activity 1 .. Activity 10.

1. Write the hour of beginning (hh:mm:ss):
2. Write the hour of end (hh:mm:ss):

A.A.3. Perceived Usefulness

Please rate your level of agreement with the following statements:

	1	**2**	**3**	**4**	**5**
PU-PEffv: I believe that this software development method is effective					
PU-PCert: I am sure of the modifications carried out with this development method					
PU-PStab: I believe that this software development method is stable					

A.A.4. Perceived Ease of Use

Please rate your level of agreement with the following statements:

	1	2	3	4	5	6	7
PEU-PEffc: I believe that this software development method is efficient							
PEU-PCompl: I believe that this software development method is easy							
PEU-PLearn: I believe that this software development method is easy to learn							

In general, I feel that this software development method is...

PEU-PSatisf	1	2	3	4	5	6	7
(1) Difficult to use .. Easy to use (7)							
(1) Tiring .. Not tiring (7)							
(1) Stranger .. Family(7)							
(1) Useless .. Useful (7)							
(1) Demanding .. Simple (7)							
(1) Inefficient .. Efficient (7)							
(1) Boring .. Fun (7)							
(1) Unreliable .. Reliable (7)							
(1) Stressful .. Relaxing (7)							
(1) Unpleasant to use .. Pleasant to use (7)							
(1) Unacceptable .. Acceptable (7)							

A.A.5. Main Advantages/Disadvantages

Please note down the main three Advantages of using this software development method

1-

2-

3-

Please note down the main three Disadvantages of using this software development method

1-

2-

3-

APÉNDICE B
ESTADÍSTICA DESCRIPTIVA DEL EXPERIMENTO DE MANTENIBILIDAD: WEBML VS PHP

Measure	Methodology	N	Mean	Standard Dev	Error
AnalysabilityPrecision	WebML	30	70,5556	44,08137	8,04812
	PHP	12	21,3333	31,35815	9,05232
AnalysabilityRecall	WebML	30	12,6913	10,50606	1,91814
	PHP	12	13,8333	18,17007	5,24525
AnalysabilityTime	WebML	30	216,17	144,158	26,320
	PHP	12	1118,47	752,466	217,218
AnalysabilitySubjectiveComplexity	WebML	27	2,19	1,001	,193
	PHP	11	1,55	1,214	,366
AnalysabilitySubjectiveCertainty	WebML	27	2,96	1,315	,253
	PHP	11	2,09	1,044	,315

Figura B–1: Analizabilidad

Measure	Methodology	N	Mean	Standard Dev	Error
CorrectiveChangeabilityPrecision	WebML	29	66,5517	42,68269	7,92598
	PHP	12	31,9167	44,07114	12,72224
CorrectiveChangeabilityRecall	WebML	29	62,8276	44,11741	8,19240
	PHP	12	41,6667	51,49287	14,86471
CorrectiveChangeabilityTime	WebML	28	139,68	171,176	32,349
	PHP	12	303,42	205,183	59,231
CorrectiveChangeabilitySubjectiveComplexity	WebML	27	2,63	,884	,170
	PHP	10	2,70	1,252	,396
CorrectiveChangeabilitySubjectiveCertainty	WebML	27	2,96	1,315	,253
	PHP	10	2,80	1,229	,389
CorrectiveStabilitySubjectiveCertainty	WebML	27	2,26	1,023	,197
	PHP	10	3,70	1,160	,367

Figura B–2: Cambiabilidad Correctiva

Measure	Methodology	N	Mean	Standard Dev	Error
PerfectiveChangeabilityPrecision	WebML	29	75,8621	41,44853	7,69680
	PHP	12	66,6667	32,56695	9,40127
PerfectiveChangeabilityRecall	WebML	29	67,2414	40,69894	7,55760
	PHP	12	66,6667	32,56695	9,40127
PerfectiveChangeabilityTime	WebML	29	186,52	147,946	27,473
	PHP	12	592,08	198,293	57,242
PerfectiveChangeabilitySubjectiveComplexity	WebML	27	2,63	,629	,121
	PHP	11	2,27	,786	,237
PerfectiveChangeabilitySubjectiveCertainty	WebML	27	2,22	,934	,180
	PHP	11	2,82	1,328	,400
PerfectiveChangeabilitySubjectiveStability	WebML	27	2,48	1,014	,195
	PHP	10	2,50	1,080	,342

Figura B–3: Cambiabilidad Perfectiva