UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Experimentation to Evaluate the Benefits of Model Driven Development

JUNE 2022

Author: África Domingo Montes

Directors: Dr. Óscar Pastor López
Dr. Carlos Cetina Englada

Dedicatoria

A mi familia. Por todos los ratos de ausencia.

Familia definida de forma amplia, incluyendo a quienes están cuando estas a gusto siendo tú mismo.

Agradecimientos

Comenzar agradeciendo la generosidad de mis directores, Dr. Carlos Cetina y Dr. Óscar Pastor. Carlos, gracias por regalarme confianza, pero, sobre todo, gracias por los sacos de ideas de cada reunión. Gracias por acogerme en SVIT, por guiarme y acompañarme. Óscar, gracias por ampliar mis horizontes, por invitarme a las lecciones magistrales de los grandes del modelado y ayudarme a entender que el código es el modelo.

Gracias a todos mis compañeros, tanto del grupo de investigación SVIT, como de la Escuela de Arquitectura e Ingeniería. Durante el desarrollo de esta tesis, de una forma u otra, todos me han ayudado. El Dr. Jorge Echeverría me acompaña en cada artículo, la Dra. Lorena Arcega aparece en cada tabla, y los consejos del Dr. Jaime Font están apuntados al margen de muchos borradores. Gracias sobre todo por vuestros ánimos. Gracias a Raúl, por su pasión por las lenguas; a Rodrigo, por aprender conmigo PHP; a Violeta, Paqui, y Ana por su generosidad y consejos. A todos ellos, y al resto de compañeros (Isabel, Moneo, Bergues, Santi, Javi, Antonio, Gabriel...), gracias por estar cuando hay que estar, y por las risas. Y cómo no, gracias a mis estudiantes, los que fueron, y los que son, por enseñarme el camino a la excelencia académica.

Terminar agradeciendo el apoyo y la paciencia de mi familia. Gracias por vuestra capacidad para celebrar a lo grande los pequeños hitos de la vida. Os quiero una cantidad tan grande, que, si le sumas uno, se queda igual.

Abstract

Model Driven Development (MDD) is a software engineering approach in which the code of a software product is generated and evolutionated from conceptual models that abstractly represents the system. For nearly two decades, the scientific community has described many of the advantages of MDD over other approaches. Despite the benefits of MDD, its use in real practical developments is merely anecdotal.

To understand why MDD has not replaced other software engineering approaches, I have conducted an empirical investigation through three controlled experiments. The first experiment aims to clarify whether the benefits of MDD compared to code-centric development (CcD) match the reality of development in real environments. In the second experiment, I compare engineers' assessment of the models they develop with the usefulness of these models to be used in MDD contexts. In the third experiment, I analyze the performance of software professionals in maintenance tasks in MDD contexts.

Our results confirm the benefits of MDD over other approaches; however, the intention to use MDD does not reach maximum values. Subjects underestimate the potential of the models they develop and use in MDD contexts. The adoption problem seems to be linked to human factors, not to technical factors.

Resumen

El Desarrollo Dirigido por Modelos, MDD por sus siglas en inglés (Model Driven Development), es un enfoque de ingeniería del software que centra la creación y evolución de productos software en el modelado. Desde hace casi dos décadas, la comunidad científica ha descrito muchas de las ventajas de MDD frente a otros enfoques, sin embargo, su adopción en el entorno industrial es muy poco frecuente.

Con el objetivo de entender por qué MDD no ha reemplazado otros enfoques de ingeniería software, he realizado una investigación empírica a través de tres experimentos controlados. Con el primer experimento pretendo aclarar si los beneficios de MDD frente al Desarrollo centrado en Código, CcD por sus siglas en inglés (Code Centric Development), son fieles a la realidad del desarrollo software actual. En el segundo experimento comparo la valoración que realizan los ingenieros de los modelos que utilizan, con su utilidad para ser utilizados en contextos MDD. En el tercer experimento analizo el desempeño de los profesionales software en tareas de mantenimiento en contextos MDD.

Nuestros resultados confirman los beneficios de MDD frente a otros enfoques, pero también, que la intención de uso de MDD no alcanza valores máximos. Los sujetos subestiman el potencial de los modelos que desarrollan y utilizan en contextos MDD. El problema de adopción parece estar ligado a factores humanos, no a factores técnicos.

Resum

El Desenvolupament Dirigit per Models, MDD (Model Driven Development), és un enfocament d'enginyeria del programari que centra la creació i evolució de productes programari en el modelatge. Des de fa quasi dues dècades, la comunitat científica ha descrit moltes dels avantatges de MDD enfront d'altres enfocaments, no obstant això, la seua adopció en l'entorn industrial és molt poc freqüent.

Amb l'objectiu d'entendre per què MDD no ha reemplaçat altres enfocaments d'enginyeria programari, he realitzat una investigació empírica a través de tres experiments controlats. Amb el primer experiment pretenc aclarir si els beneficis de MDD enfront d'altres enfocaments, com el Desenvolupament centrat en Codi, CcD (Code Centric Development), són fidels a la realitat del desenvolupament programari actual. En el segon experiment compare la valoració que realitzen els enginyers dels models que utilitzen, amb la seua utilitat per a ser utilitzats en contextos MDD. En el tercer experiment analitze l'acompliment del professional programari en tasques de manteniment en contextos MDD.

Els nostres resultats confirmen els beneficis de MDD enfront d'altres enfocaments, però també, que la intenció d'ús de MDD no aconsegueix valors màxims. Els subjectes subestimen el potencial dels models que desenvolupen i utilitzen en contextos MDD. El problema d'adopció sembla estar lligat a factors humans, no a factors tècnics.

Índice general

Parte I

Introducción

Introducción

Este capítulo presenta la motivación para realizar esta tesis, cuál es el problema que se pretende resolver y qué objetivo se plantea, cuáles serán las preguntas de investigación que guiarán el estudio empírico, y el compendio de artículos que conforman la investigación. También se analiza la estrategia empírica seleccionada y se describe la metodología seguida para desarrollar el trabajo de investigación. Cierra este capítulo la descripción de la estructura de este libro.

El Desarrollo Dirigido por Modelos, MDD por sus siglas en inglés (Model Driven Development), es un enfoque de ingeniería del software que centra la creación y evolución de productos software en el modelado [28]. Diferentes modelos conceptuales representan el sistema a desarrollar de forma abstracta [46]. El proceso de desarrollo software se automatiza aplicando transformaciones de modelo a modelo y de modelo a código. De esta forma, el esfuerzo de los desarrolladores se centra en construir los modelos que describen las características del sistema, relegando la generación de código a procesos automáticos. MDD puede entenderse como la utilización de un lenguaje de programación de cuarta generación. De la misma manera que los lenguajes de tercera generación (JAVA, C, Python, etc.) potencian la capa de abstracción para desarrollar sistemas (los desarrolladores no trabajan con lenguaje ensamblador), MDD pretende aumentar aún más esta abstracción con la utilización de modelos más cercanos al dominio de interpretación del problema.

Desde hace casi dos décadas, la comunidad científica ha descrito muchas de las ventajas de MDD, entre ellas: mejora en la calidad del software [47], software más consistente [25], rápida generación y migración de código a diferentes plataformas [44], eliminación de código repetitivo [62], modelos siempre actualizados puesto que el modelo es el código [28], reducción del esfuerzo de los desarrolladores [61], incremento de su satisfacción [43], y aumento de su productividad [13]. A principios de siglo, autores como Bran Selic [60] anunciaban la llegada de la era de MDD, en la que los modelos serían la piedra angular del desarrollo. Sin embargo, a pesar de las anteriores evidencias sobre sus beneficios, MDD no ha reemplazado al desarrollo de software tradicional, y el uso de MDD en los desarrollos reales parece meramente anecdótica [26]. La mayoría de las empresas siguen realizando sus desarrollos utilizando un enfoque centrado en el código (CcD, por sus siglas en inglés, Code-centric Development), donde los modelos sólo se utilizan para documentar el sistema, y la mayoría del esfuerzo para desarrollar un producto software recae en la implementación manual del código [8].

Para conocer el uso de los modelos y de los lenguajes de modelado en el desarrollo de software, los investigadores han realizado estudios empíricos involucrando a más de cinco mil sujetos [27, 19, 15, 72, 49, 58, 18,

15, 1, 26, 40, 10, 63, 2, 52]. A partir de las respuestas de los sujetos, los autores clasifican los estilos de modelado en tres categorías: *sketch*, modelos informales para ayudar a la comunicación; *blueprint*, modelos como base para que los programadores creen código; y *programs*, modelos que incluyen todos los detalles necesarios para generar código [27, 15, 26, 63, 2]. Estos trabajos coinciden en que la mayoría de los sujetos hacen un uso informal de los modelos, lo que se clasifica como *sketch* y que el uso de los modelos como *programs* es muy poco frecuente.

Quizá, el hecho de que la adopción de MDD no sea mayoritaria hace que las investigaciones se centren en el desarrollo de productos software en contextos MDD, y que no se profundice en otros aspectos fundamentales de la ingeniería del software como pueden ser el mantenimiento o la evolución de los productos para satisfacer nuevos requisitos. Para entender el problema de adopción de MDD, consideramos también necesario averiguar cómo es el desempeño de los ingenieros y desarrolladores cuando realizan diferentes tipos de tareas en contextos MDD.

Con este trabajo se pretende ampliar el conocimiento sobre por qué MDD no ha reemplazado otros enfoques de ingeniería del software. La investigación empírica, al igual que en otras disciplinas que tienen que ver con el comportamiento humano (p.e., ciencias sociales o psicología), permite construir una base fiable de conocimiento, reduciendo la incertidumbre sobre la adecuación de herramientas, métodos y teorías en ingeniería del software [64, 71]. Aproximarnos al problema de forma empírica a través de la experimentación, nos permitirá evaluar MDD de forma precisa y considerando el punto de vista de los desarrolladores e ingenieros software. De esta forma se profundizará en las causas por las que MDD no ha tenido el éxito pronosticado por la comunidad.

La Figura 1 resume el trabajo realizado en esta tesis, como, desde el análisis de los trabajos empíricos previos en MDD, se determinan un objetivo general y tres preguntas de investigación, para ahondar en las causas por las que MDD no ha sustituido a otras alternativas de desarrollo software. El interés de estas pregunta, reside en que, a pesar de los casos de éxito de MDD, MDD no ha reemplazado a CcD y el mantenimiento en MDD no ha recibido tanta atención. Para dar respuesta a estas preguntas, se

profundiza en los beneficios, la adopción y el mantenimiento en contextos MDD, utilizando la experimentación.

Objetivos

El objetivo fundamental de este trabajo es estudiar empíricamente las causas por las que MDD no ha tenido el éxito pronosticado por la comunidad. En primer lugar, nos cuestionamos si los beneficios de MDD frente a otros enfoques como CcD son reales, si son vigentes en entornos reales de desarrollo software. En segundo lugar, nos preguntamos si los ingenieros y desarrolladores software evalúan correctamente los modelos que utilizan, y si conocen la potencialidad de los lenguajes de modelado a su disposición. En tercer lugar, queremos averiguar cómo es el desempeño de los ingenieros en tareas de mantenimiento software en contextos MDD. Por tanto, las preguntas de investigación que guiarán este trabajo son las siguientes:

RQ_1: ¿Son los beneficios de MDD frente a otros enfoques como CcD fieles a la realidad del desarrollo software actual?

Los beneficios de MDD frente a CcD han sido previamente evaluados empíricamente por diferentes autores [6, 3, 37, 29, 34, 45, 30, 42, 51, 50, 48]. Los trabajos coinciden en que MDD disminuye el tiempo de desarrollo en relación con CcD [37, 51, 29] o en que MDD mejora la calidad del software implementado [34, 43, 50, 48]. Sin embargo, algunos trabajos sugieren que las ganancias sólo se pueden conseguir en ejercicios académicos [50, 34] o que sólo los desarrolladores sin experiencia se benefician de MDD [34, 42]. También observamos que ninguno de los trabajos analizados tienen en cuenta frameworks de dominio, código previamente escrito, que los desarrolladores utilizan para acelerar el proceso de desarrollo software en los contextos CcD, y que en el contexto MDD, ayudan en las transformaciones del modelo a la hora de llenar la brecha de abstracción entre los modelos y el código.

Para entender los beneficios de MDD frente a CcD es fundamental realizar una comparación justa y fiel a los estándares actuales en materia de desarrollo software. Consideramos necesario evaluar empíricamente los

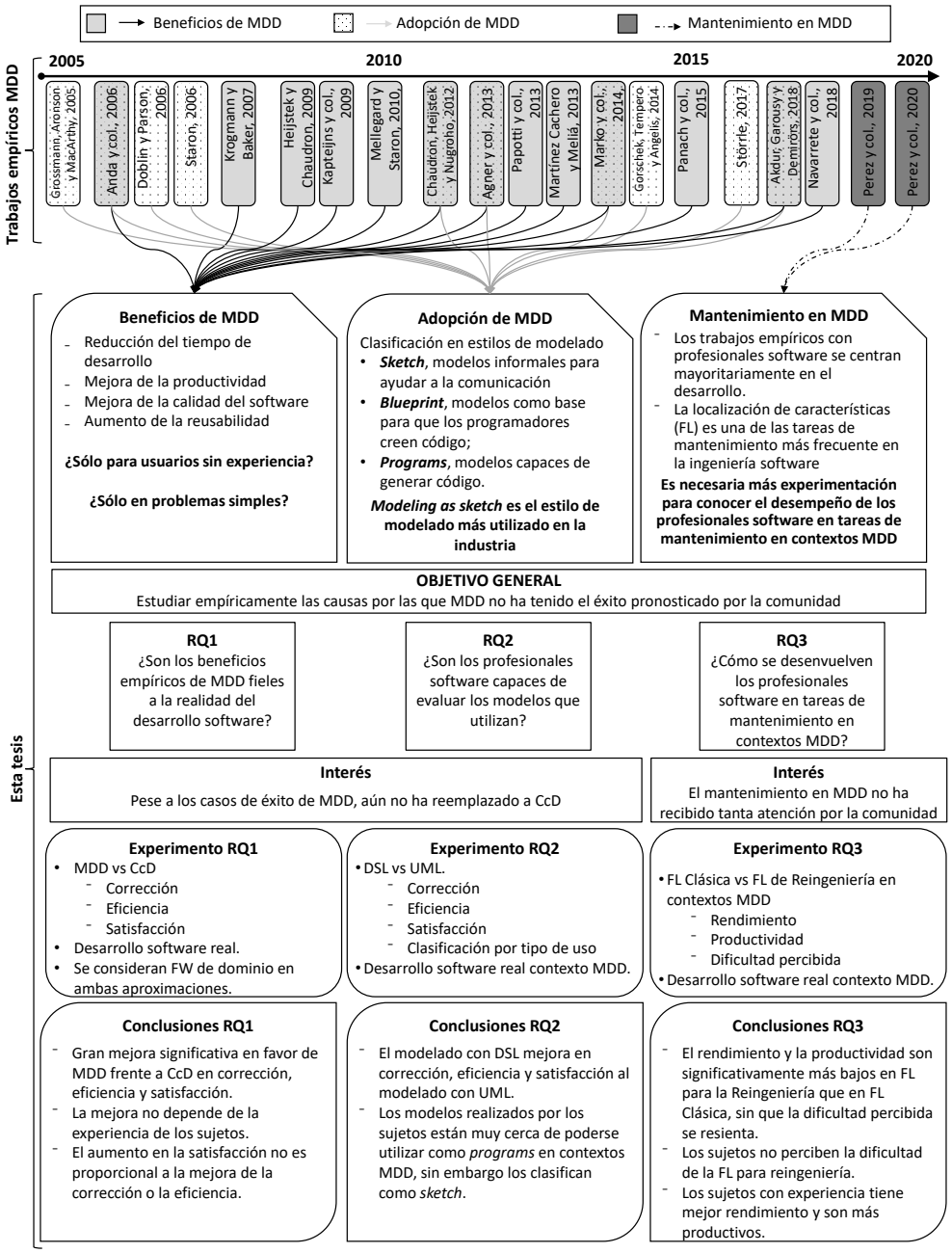

Figura 1: Resumen de esta tesis

beneficios de MDD frente a CcD aumentando la fidelidad de la comparación con la práctica profesional real, utilizando tareas del contexto industrial y teniendo en cuenta frameworks de dominio en el desarrollo de las tareas, no solo en MDD sino también en CcD. El estado del arte ha ignorado el uso de frameworks en CcD cuando comparaba ambos métodos.

Para responder a la primera pregunta de investigación, realizamos un experimento con tareas extraídas de un entorno de desarrollo software real, en las que se utilizan frameworks de dominio tanto en el contexto MDD como en el contexto CcD. *Evaluating the Benefits of Model-Driven Development* [21] es el primer resultado que se presenta en el compendio de artículos que componen esta tesis.

RQ_2: ¿Los profesionales software valoran adecuadamente los modelos que utilizan para el desarrollo de software?

Los trabajos previos sobre adopción de MDD [27, 15, 26, 63, 2] coinciden en que la mayoría de los sujetos hacen un uso informal de los modelos, lo que se clasifica como *sketch*. En estos trabajos sólo se utiliza la evaluación subjetiva de los sujetos para clasificar su estilo de modelado y el uso que hacen de los modelos, sin tener en cuenta medidas objetivas que cuantifiquen la adecuación de esos modelos a un contexto MDD.

Nos parece necesario evaluar los modelos que realizan los sujetos en un contexto MDD considerando tanto medidas objetivas como subjetivas. Las medidas objetivas nos permitirán evaluar cuán de útiles son los modelos realizados por los sujetos en un contexto MDD. Las medidas subjetivas nos ayudaran a entender la percepción que tiene los sujetos sobre los modelos que desarrollan y los lenguajes de modelado que utilizan para hacerlo.

Para responder a la segunda pregunta de investigación, realizamos un experimento en el que los sujetos debían modelar parte de un proyecto profesional utilizando dos lenguajes de modelado diferentes. Evaluamos sus modelos utilizando medidas objetivas (corrección y eficiencia) y subjetivas (satisfacción y clasificación de la utilidad del modelo). *Comparing UML-based and DSL-based Modeling from Subjective and Objective Pers-*

pectives [20] es el segundo resultado que se presenta en el compendio de artículos que componen esta tesis.

RQ_3: ¿Cómo es el desempeño de los profesionales software en tareas de mantenimiento en contextos MDD?

Los trabajos analizados previamente sobre MDD centran su análisis en el desarrollo de software, ignorando otros aspectos fundamentales de la ingeniería del software como pueden ser el mantenimiento o la evolución de los productos para satisfacer nuevos requisitos. La localización de características (FL, por sus siglas en inglés, Feature Location) puede entenderse como el proceso de encontrar subconjuntos de elementos (no todos conocidos), en un conjunto mayor de elementos que representa un producto software. Los elementos que intervienen en la búsqueda vienen determinados por los artefactos en los que se realiza la búsqueda (código, modelos, etc.) y la granularidad del problema (líneas de código, funciones, elementos de modelo, etc.). La localización de características es una de las tareas de mantenimiento más frecuentes que realizan los ingenieros software [38, 31, 67, 57, 11, 36], ya que el mantenimiento y la evolución del software implican la adición de nuevas características a los programas, la mejora de las funcionalidades existentes y la eliminación de las funcionalidades no deseadas. Para ello, es esencial que los ingenieros software encuentren los elementos asociados a ciertas funcionalidades del software. FL también puede ayudar a la reingeniería de una familia de productos software en una Línea de Productos Software (SPL, por sus siglas en inglés, Software Product Line) [5, 41], ya que una SPL implica la formalización de características que son compartidas por varios productos de una familia. También es esencial que los ingenieros de una SPL sean capaces de localizar esas características comunes en una familia de productos.

Los trabajos previos analizados sobre localización de características en los que se involucra a sujetos finales, sólo consideran la localización en código y no en modelos, no tienen en cuenta contextos MDD [70, 59, 66, 67, 32, 17, 38]. La localización de características en contextos MDD a escala industrial es un tema central en trabajos previos de nuestro grupo de investigación SVIT [24, 39, 55, 23, 4, 14, 7, 53, 54]. Sin embargo, sólo dos de estos trabajos [53, 54] abordan la perspectiva de los ingenieros

en la localización de características. Para poder tener una imagen más clara de lo que significa la adopción de MDD en la industria software, nos parece necesaria más investigación que analice el desempeño de los ingenieros en tareas de localización de características en contextos MDD.

Para dar respuesta a la tercera pregunta de investigación de este trabajo, realizamos un experimento en el que los sujetos debían realizar localización de características en contextos MDD, tanto cuando la FL se orienta a la resolución de errores en el software existente, como cuando se orienta a la evolución de una familia de productos a una línea de productos software. *Evaluating the Influence of the Scope on Feature Location* [22] es el tercer resultado que se presenta en el compendio de artículos que componen esta tesis.

Compendio de artículos

Como resultado de la investigación que se ha llevado a cabo para esta tesis, se han elaborado y publicado tres artículos de investigación. Estos trabajos se han desarrollado en el contexto de un proyecto de investigación del Plan Estatal de Investigación denominado ALPS (RTI2018-096411-8-I00).

Evaluating the benefits of Model-Driven Development

África Domingo, Jorge Echeverría, Óscar Pastor y Carlos Cetina. En: International Conference on Advanced Information Systems Engineering. Springer. 2020, págs. 353-36.

El primer trabajo de este compendio presenta los resultados de un experimento en el que comparamos MDD y CcD en términos de corrección, eficiencia y satisfacción. Las tareas de nuestro experimento se extrajeron de las tareas de desarrollo de un videojuego comercial real (Kromaia, lanzado en PlayStation 4 y Steam). Un total de 44 sujetos, clasificados en dos grupos según su experiencia desarrollando software, realizaron las tareas del experimento. Los sujetos desarrollaron un enemigo del videojuego ('boss') desde cero, contando con frameworks de dominio tanto para desarrollarlo en el contexto CcD como en el contexto MDD.

Nuestros resultados desafían a aquellos que sugieren que los beneficios de MDD sólo se dan en entornos académicos. En nuestro experimento, con tareas más fieles al desarrollo actual, MDD mejora la corrección y la eficiencia en un 41 % y un 54 % respectivamente. Los resultados también contradicen las afirmaciones que limitaban los beneficios de MDD a desarrolladores sin experiencia. En el experimento, tanto los desarrolladores sin experiencia como los desarrolladores con experiencia mejoraron significativamente sus resultados con MDD. Los resultados también revelaron una paradoja: A pesar de las significativas mejoras en corrección y eficiencia, la intención de uso de MDD no alcanza los valores máximos.

Comparing UML-Based and DSL-Based Mode- ling from Subjective and Objective Perspectives

África Domingo, Jorge Echeverría, Óscar Pastor y Carlos Cetina. En: International Conference on Advanced Information Systems Engineering. Springer. 2021,

En este trabajo, se presenta un experimento que compara UML y un DSL cuando los sujetos modelan un 'boss' o enemigo del videojuego comercial Kromaia. En Kromaia, los modelos se interpretan en tiempo de ejecución para crear los objetos C++ del juego [12] (modelos como programas). Para el modelado, los sujetos utilizaron el lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) y el lenguaje de dominio específico (DSL, por sus siglas en inglés, Domain Specific Language) utilizado en Kromaia [1]. Un total de 31 sujetos (clasificados en dos grupos en función de su experiencia profesional) realizaron las tareas de modelado del experimento, expresaron su satisfacción con los lenguajes utilizados y clasificaron sus modelos en función de su utilidad en el desarrollo software. En este experimento tuvimos en cuenta tanto medidas objetivas (corrección y eficiencia), como medidas subjetivas (satisfacción con los lenguajes y clasificación de los modelos) a la hora de evaluar los modelos de los sujetos.

Nuestros resultados muestran que la corrección, la eficiencia y la satisfacción mejoran cuando los sujetos modelan utilizando el DSL. Al igual que

[1] Más información sobre el DSL utilizado en Kromaia en: `https://youtu.be/Vp3Zt4qXkoY`

en otros trabajos empíricos previos [27, 15, 26, 63, 2], los sujetos de nuestro experimento también afirman que el uso principal de sus modelos es para asistir en la comunicación del equipo de desarrollo (como 'sketch'). Sin embargo, los modelos elaborados por los sujetos en el experimento no están lejos de obtener un 100 % de corrección para ser utilizados como programas. Por término medio, sólo habría que corregir un 28 % en los modelos UML y un 10 % en los modelos DSL. Nuestro trabajo revela un problema que no se había encontrado en las evaluaciones de otros autores: por muy correctos que sean los modelos, los sujetos los consideran más como 'sketch' que como 'programs'. Nuestros resultados revelan que los sujetos subestiman el potencial de sus propios modelos, y que probablemente, el problema de adopción de MDD no se limita a elegir el lenguaje de modelado que mejor funciona en un determinado dominio.

Evaluating the influence of scope on feature location

África Domingo, Jorge Echeverría, Óscar Pastor y Carlos Cetina. En: Information and Software Technology 140 (2021), pág. 106674

El tercer trabajo del compendio que compone esta tesis, consiste en un experimento en el que los sujetos realizan tareas de localización de características (FL, por sus siglas en inglés, Feature Location) en una familia de modelos de productos software. Los sujetos realizan dos tareas diferentes de localización: FL clásica, cuando la localización de característica se realiza en único producto para resolver un problema en la característica o actualizarla, y FL para la reingeniería, cuando el objetivo es localizar características comunes en una familia de productos para generar, por ejemplo, una línea de producto software (SPL). En este trabajo comparamos el rendimiento, la productividad y la dificultad percibida por un total de 18 sujetos (clasificados en dos grupos en función de su experiencia con el DSL) cuando localizan características en diferentes contextos: en un único producto (FL clásica) o en una familia de productos (FL para reingeniería). Los objetos experimentales se extraen de una SPL del mundo real que utiliza un lenguaje específico de dominio (DSL) para generar el firmware de sus productos a partir de modelos.

Nuestros resultados muestran cómo el rendimiento y la productividad disminuyen significativamente cuando los ingenieros localizan características en una familia de productos (FL para reingeniería), independientemente de su experiencia utilizando el DSL. Sin embargo, nuestros resultados también descubren una paradoja: aunque el rendimiento y la productividad son significativamente peores cuando los sujetos realizan FL para reingeniería, la dificultad que perciben los sujetos no predice unos resultados tan negativos. La inclinación inicial de los sujetos es pensar que localizar características en una familia de productos sólo lleva más tiempo que localizar características en un producto. Aparentemente, los sujetos no son conscientes del reto que supone la FL para reingeniería y subestiman la dificultad de hacer mantenimiento en contextos MDD. Un análisis más profundo de los resultados indica cómo el tamaño de las características a localizar, o su dispersión en la familia de productos, explican algunos de los cambios en el rendimiento cuando se realiza FL para reingeniería. Conocer las razones que dificultan las tareas de localización de características, puede ayudar a los ingenieros a aplicar y desarrollar técnicas que mejoren el rendimiento y la productividad de las tareas de localización en contextos MDD.

Estrategia empírica

La experimentación en ingeniería del software es una herramienta imprescindible para comprender mejor qué es lo que hace que el software sea bueno y cómo hacerlo bien [56], sin embargo, son muchos los autores que reclaman más experimentación en ingeniería del software [33, 71, 64, 56]. Por esta razón, la estrategia empírica seleccionada en esta investigación son los experimentos controlados. Estos experimentos permiten controlar la situación y manipular el contexto de forma directa, precisa y sistemática. Como estudios cuantitativos, generan variables numéricas, útiles para cotejar ideas o teorías con la realidad, y ampliar de forma fiable el conocimiento, en este caso, sobre MDD. En la Figura 2 se representan las características de los experimentos realizados durante el desarrollo de la tesis. Se detallan aquellas características necesarias para satisfacer el diseño, ejecución, análisis e interpretación de un experimento controlado según Wohlin y col. [71].

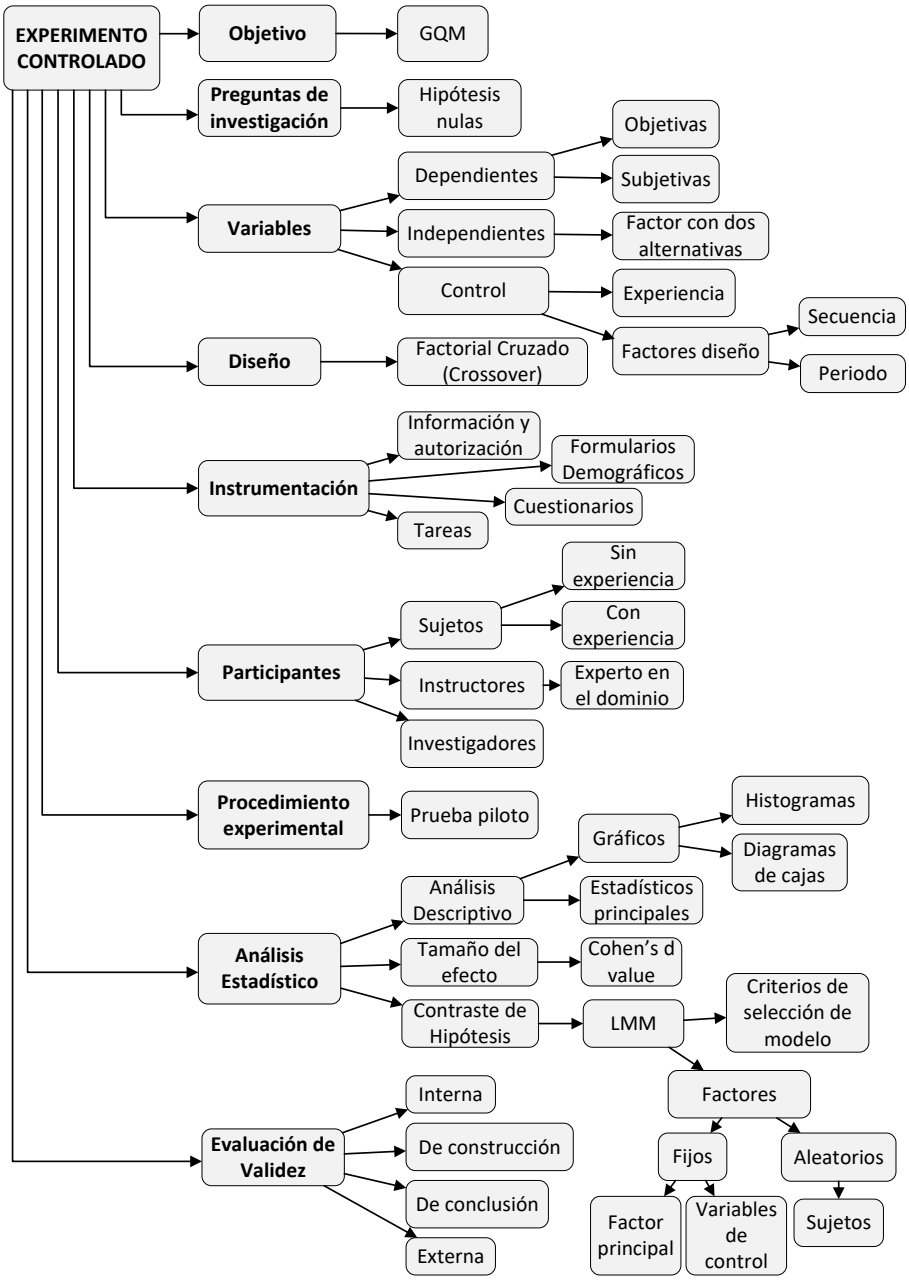

Figura 2: Características de los experimentos de esta tesis

Objetivo

En cada uno de nuestros experimentos definimos el objetivo de investigación utilizando la plantilla Goal Question Metric [9] para experimentos en ingeniería del software. Por cada objetivo se determina el objeto de estudio, el propósito, los efectos que se pretenden estudiar, el punto de vista que se analiza y el contexto.

Preguntas de investigación e hipótesis

Tras establecer el objetivo de cada experimento, se formulan preguntas de investigación. Estas preguntas permiten establecer las hipótesis nulas que se responderán a través del análisis estadístico de los datos recogidos tras la ejecución del experimento.

Variables

El factor de estudio en todos nuestros experimentos cuenta con dos alternativas. En el primer experimento comparábamos dos métodos de desarrollo software (MDD/CcD), en el segundo comparábamos dos lenguajes de modelado (UML/DSL) y en el tercero, dos tareas de localización de características (FL clásica/FL para reingeniería).

En los experimentos hemos utilizado dos tipos de variables dependientes, objetivas y subjetivas. En los experimentos sobre desarrollo (RQ_1 y RQ_2), como medidas objetivas hemos utilizado *Corrección* y *Eficiencia*, asociadas al software desarrollado por los sujetos y al tiempo que utilizan para hacerlo. Como medidas subjetivas hemos utilizado *Satisfacción*, descompuesta en *Facilidad de Uso Percibida*, *Utilidad Percibida* e *Intención de Uso*. En el experimento RQ_2, también hemos utilizado la clasificación que los sujetos hacen de los modelos que construyen durante el experimento. En el experimento RQ_3, en el que se analiza localización de características, como variables dependientes objetivas se utiliza *Rendimiento*, a partir de las F-measure de la matiz de confusión, y *Productividad*, en la que se tiene en cuenta el tiempo utilizado por los sujetos para realizar la localización. Como variable dependiente subjetiva se utiliza la dificultad percibida por los sujetos en las tareas realizadas.

En nuestros experimentos también hemos considerado variables de control, factores, como pueden ser la experiencia de los sujetos o aquellos que tienen que ver con el diseño del experimento, que pueden afectar a las variables dependientes al mismo tiempo que lo hace el factor principal del estudio.

Diseño

En todos los experimentos realizados hemos elegido un diseño factorial cruzado con dos periodos, utilizando dos tareas diferentes, una para cada periodo. Como muestra la Tabla 1, los sujetos se dividen en dos grupos, y en cada periodo cada sujeto realiza la tarea correspondiente a ese periodo, utilizando una de las alternativas del factor que se compara. En el primer periodo el primer grupo realiza la tarea utilizando una de las alternativas del factor, mientras el otro grupo realiza la misma tarea utilizando la otra alternativa del factor. En el segundo periodo, ambos grupos realizan la segunda tarea, cada grupo con la alternativa del factor que no ha utilizado en la primera tarea. Al final del experimento todos los sujetos han realizado las dos tareas, empleando en cada una de ellas una alternativa diferente del factor que se compara.

Tabla 1: Diseño factorial cruzado

	Periodo (Tarea)	
Secuencia (Grupo)	*Periodo 1 (Tarea 1)*	*Periodo 2 (Tarea 2)*
Secuencia 1 (G1)	Alternativa I	Alternativa II
Secuencia 2 (G2)	Alternativa II	Alternativa I

Este diseño de medidas repetidas aumenta la sensibilidad del experimento [65]: la observación del mismo sujeto utilizando las dos alternativas controla las diferencias entre sujetos, mejorando la robustez del experimento en cuanto a la variación entre sujetos. Este diseño también contrarresta algunos de los efectos causados por el uso de las alternativas del factor en un orden específico (efecto de aprendizaje, fatiga), puesto que se utilizan dos tareas diferentes (diferentes periodos) y se aplican en diferente orden en cada grupo (dos secuencias diferentes). No obstante, se recomienda tener en cuenta los efectos de los factores que intervienen en un diseño cruzado, *Periodo* y *Secuencia*, en los estudios estadísticos que se realicen.

Instrumentación

Para cada experimento se ha preparado un consentimiento informado que los sujetos deben revisar y aceptar voluntariamente. En él se explica claramente en qué consiste el experimento y cuál será el tratamiento de los datos personales. También se prepara un cuestionario demográfico que se utiliza para caracterizar la muestra, y se diseñan cuestionarios específicos para recoger las respuestas de los sujetos durante el experimento.

Las tareas que hemos utilizado en todos nuestros experimentos se extraen de desarrollos software del mundo real. Para los experimentos RQ_1 y RQ_2 se utilizaron tareas extraídas de un videojuego comercial, Kromaia. En Kromaia se utiliza el DSL Shooter Definition Modeling Language[2] para modelar, y los modelos se interpretan en tiempo de ejecución para crear los objetos de los juegos en C++ [12]. En el experimento RQ_3 se utiliza un subconjunto de la familia de modelos de placas de inducción de nuestro socio industrial BSH. BSH utiliza una línea de productos software y un lenguaje específico de dominio para generar el firmware de sus placas de inducción. Este DSL está compuesto por 46 metaclases, 74 referencias y más de 180 propiedades. Las tareas de todos los experimentos, y las plantillas de corrección, han sido diseñadas por expertos en el dominio de aplicación en cada caso.

Los objetos experimentales utilizados en los experimentos (que incluyen el material de formación, las tareas y los formularios utilizados para los cuestionarios), así como los resultados y el análisis estadístico, están disponibles en:

Experimento RQ_1: `http://svit.usj.es/MDD-experiment`

Experimento RQ_2: `http://svit.usj.es/UMLvsDSL-experiment`

Experimento RQ_3: `http://svit.usj.es/ManualFL-experiment`

[2]Aprende más de Kromaia DSL en: https://youtu.be/Vp3Zt4qXkoY

Participantes

Los sujetos de nuestros experimentos fueron seleccionados según un muestreo de conveniencia [71]. Por cada experimento se realizan dos sesiones, en una de ella participan sujetos clasificados como sin experiencia, y en otra los sujetos clasificados como con experiencia.

Los experimentos los ejecutan dos instructores. Un instructor dirige el experimento, informa a los sujetos sobre las actividades a realizar, aclara dudas durante el experimento y apoya en el reparto y recogida del material asociado. El otro instructor reparte y recoge el material asociado al experimento, apoya en la aclaración de dudas y toma notas durante el focus group. En los experimentos RQ_1 y RQ_2 también participó, como instructor adicional, un experto en el dominio, que explicó a los participantes el DSL de Kromaia y resolvió dudas específicas. En RQ_3 los instructores eran ingenieros senior de la empresa, por lo que uno de ellos explicó la utilización del DSL utilizado en el experimento y resolvió las dudas específicas.

La corrección de tareas y el tratamiento de datos corresponde a uno o varios investigadores que no participan en el diseño de las tareas y no siempre participan en la ejecución del experimento.

Procedimiento experimental

El procedimiento experimental recoge la secuencia de pasos que se han de seguir en el experimento, indicando la actividad a realizar, el tiempo estimado y el rol de los participantes en cada uno de esos pasos.

Para verificar el procedimiento de cada experimento, y el diseño en general, se realiza un estudio piloto previo con dos o tres sujetos. Este estudio permite estimar el tiempo necesario para resolver las tareas y completar los cuestionarios, detectar errores tipográficos y semánticos, y probar el entorno para desarrollar el experimento. Los sujetos del estudio piloto no participan posteriormente en el experimento ni se incluyen sus datos en el estudio estadístico que se realiza.

Los experimentos RQ_1 y RQ_2 se realizaron de forma presencial en las instalaciones de la Universidad San Jorge (Zaragoza, España). El experimento RQ_3 se realizó en línea debido a las restricciones de la pandemia COVID'19.

Análisis estadístico

Para realizar los contrastes de hipótesis se ha elegido el Modelo Lineal Mixto (LMM, por sus siglas en inglés, Linear Mixed Model). En particular, utilizamos la prueba de tipo III de efectos fijos con covarianza repetida no estructurada. El supuesto para aplicar el LMM es la normalidad de los residuos, pero no exige que éstos sean independientes o que tengan varianza no constante [68]. Un LMM es un modelo lineal paramétrico que cuantifica la relación entre una variable dependiente continua y diferentes factores. LMM debe su nombre a que se pueden incluir en el modelo estadístico factores asociados a efectos fijos y a efectos aleatorios. Los parámetros de efectos fijos describen las relaciones de los variables independientes y de control con la variable dependiente para toda la población, los parámetros de efectos aleatorios son específicos para cada sujeto dentro de una población.

Utilizar LMM permite incluir en los modelos estadísticos factores que representen el efecto de variables de control (como en la experiencia en nuestros experimentos), o los efectos de los factores que intervienen en un diseño cruzado (como son período, secuencia o sujeto) [65]. Para los contrastes se pueden utilizar modelos estadísticos simples, como el Modelo 0 de la fórmula (1), en el que para estudiar los cambios en la variable dependiente (VD) solo se tiene en cuenta un factor fijo (FACTOR), que corresponde con el factor principal de estudio, y un factor aleatorio (1|Sujeto), correspondiente a los sujetos. Pero también se pueden utilizar modelos estadísticos más complejos como los Modelos 1, 2 y 3 de la misma fórmula (1), que incluyan también los efectos de una variable de control (Experiencia), de las interacciones de la variable de control con el factor principal (FACTOR*Experiencia), u otros factores (Periodo o Secuencia) y las interacciones entre ellos.

$$
\begin{array}{ll}
(Modelo\ 0) & VD \sim FACTOR + (1|\, Sujeto) \\
(Modelo\ 1) & VD \sim FACTOR + Experiencia + (1|\, Sujeto) \\
(Modelo\ 2) & VD \sim FACTOR + Experiencia + FACTOR * Experiencia + (1|\, Sujeto) \\
(Modelo\ 3) & VD \sim FACTOR + Experiencia + Periodo + Secuencia + (1|\, Sujeto)
\end{array}
\tag{1}
$$

Para escoger qué modelo estadístico explica mejor la variación en una variable dependiente, se utilizan medidas de bondad de ajuste, como el criterio de información de Akaike (AIC) o el criterio de información bayesiano de Schwarz (BIC). El modelo con el menor AIC o BIC se considera el modelo de mejor ajuste [35].

Para cuantificar el tamaño del efecto de un factor sobre la variable dependiente, medimos las diferencias de la variable dependiente en las alternativas del factor utilizando la diferencia estandarizada entre medias, el valor d de Cohen [16]. Los valores d de Cohen entre 0,2 y 0,3 indican un efecto pequeño, los valores en torno a 0,5 indican un efecto medio y los valores superiores a 0,8 indican un efecto grande. Este valor también nos permite medir el porcentaje de solapamientos entre las distribuciones de las variable dependiente para cada alternativa del factor.

En las publicaciones asociadas a nuestros experimentos, además de incluir un resumen de los valores de los estadísticos descriptivos de las variables dependientes en las diferentes alternativas de los factores de estudio, seleccionamos gráficos de caja e histogramas para describir gráficamente los datos y los resultados. En particular, se utiliza la superposición de los histogramas de la variable dependiente para las diferentes alternativas del factor. Este tipo de gráfico permite interpretar visualmente el tamaño del efecto: Cuanta menos área compartan los histogramas de las alternativas de un factor, mayor será la diferencia en la variable dependiente debida a ese factor.

Análisis de Validez

Para analizar la validez de nuestros experimentos, hemos utilizado la clasificación de Wohlin y col. [71] en cuatro dimensiones:

> **Validez de construcción**, que se consigue cuando las medidas representan realmente lo que se está investigando en base a las preguntas de investigación.
>
> **Validez interna**, que se consigue cuando las relaciones observadas entre el tratamiento y el resultado son relaciones causales y estas relaciones no son el resultado de un factor sobre el que no tenemos control o no hemos medido.
>
> **Validez de conclusión**, que se consigue cuando existe una relación estadística fiable, con cierta significación, entre el tratamiento y los resultados.
>
> **Validez externa**, que se consigue cuando los resultados pueden generalizarse fuera del entorno del experimento.

En cada experimento se ha realizado un análisis de las amenazas que afectan a cada una de estas dimensiones de validez. Estas amenazas se han mitigado o evitado en las diferentes fases experimentales siempre que ha sido posible.

Metodología de investigación

Para desarrollar este trabajo se ha seguido la metodología de investigación 'design science' [69], particularizada a la resolución de un problema de conocimiento científico a través de su ciclo empírico. En 'design science' se propone este ciclo cuando se pretende ejecutar una investigación empírica que pretenda incrementar el conocimiento sobre un contexto determinado. En este trabajo se pretende utilizar la experimentación para incrementar el conocimiento sobre las causas por las que MDD no ha reemplazado a otras alternativas como CcD a pesar de sus beneficios. Las cinco fases del ciclo empírico se materializan en este trabajo de la siguiente forma:

1. Investigación del problema de conocimiento

   Se han investigado trabajos previos que, desde una perspectiva empírica, analizan los beneficios, la adopción y el mantenimiento de MDD, para concluir que es necesaria más investigación para esclarecer por qué MDD no ha reemplazado a otras alternativas como CcD. A partir de la revisión bibliográfica, se definen tres preguntas de investigación que nos permiten profundizar en las razones que hacen que MDD no haya tenido el éxito pronosticado por la comunidad a pesar de sus beneficios.

2. Diseño de la investigación y de la inferencia

   Para contestar las preguntas de investigación, se han utilizado tres experimentos controlados. Se evalúan empíricamente los beneficios y el desempeño de los ingenieros en contextos MDD teniendo en cuenta medidas objetivas y subjetivas. Cada uno de los experimentos realizados cuenta con su propio diseño, a partir del cual se establecen, entre otros, el procedimiento para ejecutarlo con sujetos, o cómo debe ser la recogida, la validación y el posterior análisis estadístico de los datos. Tal como propone el ciclo empírico [69], la preparación de la investigación y la inferencia están estrechamente vinculadas. Ambos se diseñan antes de comenzar la investigación.

3. Validación de la investigación

   Cada uno de los experimentos que se realizan en el contexto de este trabajo incluye un análisis de las amenazas de validez (de construcción, interna, de conclusión y externa). El impacto de estas amenazas se ha mitigado o minimizado siempre que ha sido posible desde el propio diseño de los experimentos. Por otro lado, la publicación de nuestros resultados valida el interés de este trabajo para la comunidad científica.

4. Ejecución de la investigación

   La ejecución de los experimentos se ha llevado a cabo de acuerdo con su diseño, que incluye un procedimiento para llevar a cabo el experimento, recoger, validar y analizar los datos.

5. Evaluación de los resultados

   Los datos generados en cada experimento se han analizado siguiendo el modelo de inferencia seleccionado en su diseño. Como resultado, se han publicado tres trabajos de investigación, uno por cada pregunta de investigación y experimento. Cada uno de ellos contribuye con sus resultados a esclarecer las razones por las que la industria software no ha adoptado de forma mayoritaria un enfoque MDD para desarrollar software.

El ciclo empírico se materializa a través de una agrupación lógica de preguntas que, de forma flexible, guían el diseño y la ejecución de una investigación empírica, adaptándose a las características de nuestros experimentos. Además, esta metodología proporciona una lista de control que puede utilizarse para orientar la redacción y el análisis de informes científicos, lo que ha facilitado la organización de la información a la hora de preparar los artículos asociados a nuestros experimentos. La Figura 3 ilustra las fases del ciclo empírico y su relación con una selección de las preguntas propuestas por R.J.Wieringa [69] en su lista de control para documentar la información de investigaciones empíricas.

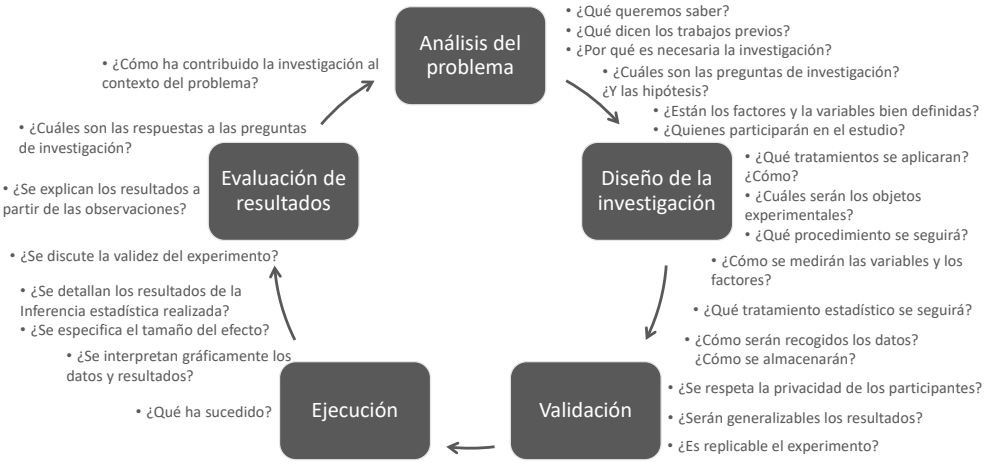

Figura 3: Ciclo empírico de *Design Science* y lista de control.

Durante el desarrollo de la tesis se han diseñado, validado, ejecutado y analizado tres experimentos controlados, uno por cada pregunta de investigación. Se puede interpretar en cada uno de ellos un ciclo empírico. La evaluación de resultados de cada experimento ha generado nuevas preguntas de investigación y refinado el diseño de los siguientes experimentos, propiciando nuevos ciclos. Sirvan de ejemplo lo trabajos [21] y [20], presentados y discutidos en CAiSE'20 y CAiSE'21 respectivamente. Las preguntas de los participantes y el debate de CAiSE'20 [21] me llevaron a investigar más a fondo sobre adopción de MDD y lenguajes de modelado. Parte de esa investigación está plasmada en el trabajo que presentamos al año siguiente en CAiSE'21 [20].

Estructura de la tesis

Siguiendo la normativa de la Universitat Politècnica de València para las tesis por compendio de artículos, la estructura de este trabajo se ajusta a las siguientes cuatro partes:

Parte I. Introducción: La primera parte de la tesis, la parte que finaliza esta sección, introduce la motivación de la investigación, la descripción del problema, los objetivos del trabajo, la relación de artículos científicos publicados para el cumplimiento de los objetivos de la tesis y la metodología seguida para desarrollar la investigación.

Parte II. Publicaciones: La segunda parte de la tesis, formada por tres capítulos (capítulos 1, 2 y 3) recoge el compendio de artículos científicos que resultan de la investigación realizada para la tesis. Las contribuciones están ordenadas cronológicamente y su formato ha sido adaptado al formato de esta tesis.

Parte III. Discusión: En la tercera parte de la tesis se realiza una discusión general de los resultados, relacionando las aportaciones de la tesis con el contexto de la investigación.

Parte IV Conclusiones: La cuarta y última parte de la tesis presenta las conclusiones sobre el trabajo realizado y las futuras líneas de investigación.

Bibliografía

[1] Luciane Telinski Wiedermann Agner y col. «A Brazilian survey on UML and model-driven practices for embedded software development». En: *Journal of Systems and Software* 86.4 (2013), págs. 997-1005 (vid. pág. 5).

[2] Deniz Akdur, Vahid Garousi y Onur Demirörs. «A survey on modeling and model-driven engineering practices in the embedded software industry». En: *Journal of Systems Architecture* 91 (2018), págs. 62-82 (vid. págs. 5, 8, 12).

[3] Bente Anda y Kai Hansen. «A case study on the application of UML in legacy development». En: *ISESE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering*. 2006. ISBN: 1595932186 (vid. pág. 6).

[4] Lorena Arcega y col. «Leveraging Models at Run-Time to Retrieve Information for Feature Location.» En: *MoDELS@ Run. time*. 2015, págs. 51-60 (vid. pág. 9).

[5] Wesley KG Assunção, Silvia R Vergilio y Roberto E Lopez-Herrejon. «Automatic extraction of product line architecture and feature models from UML class diagram variants». En: *Information and Software Technology* 117 (2020), pág. 106198 (vid. pág. 9).

[6] Paul Baker, Shiou Loh y Frank Weil. «Model-Driven Engineering in a Large Industrial Context — Motorola Case Study». En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2005. ISBN: 3540290109 (vid. pág. 6).

[7] Manuel Ballarín y col. «Measures to report the location problem of model fragment location». En: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, págs. 189-199 (vid. pág. 9).

[8] Simonetta Balsamo y col. «Model-based performance prediction in software development: A survey». En: *IEEE Transactions on Software Engineering* 30.5 (2004), págs. 295-310 (vid. pág. 4).

[9] Victor R. Basili y H. Dieter Rombach. «The TAME Project: Towards Improvement-Oriented Software Environments». En: *IEEE Transactions on Software Engineering* (1988). ISSN: 00985589 (vid. pág. 15).

[10] Maicon Bernardino, Elder M Rodrigues y Avelino F Zorzo. «Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches». En: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, págs. 1660-1665 (vid. pág. 5).

[11] Ted J Biggerstaff, Bharat G Mitbander y Dallas Webster. «The concept assignment problem in program understanding». En: *[1993] Proceedings Working Conference on Reverse Engineering*. IEEE. 1993, págs. 27-43 (vid. pág. 9).

[12] Daniel Blasco y col. «An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering». En: *Journal of Systems and Software* 171 (2021), pág. 110804 (vid. págs. 11, 17).

[13] Marco Brambilla, Jordi Cabot y Manuel Wimmer. «Model-driven software engineering in practice». En: *Synthesis lectures on software engineering* 3.1 (2017), págs. 1-207 (vid. pág. 4).

[14] Carlos Cetina y col. «Improving feature location in long-living model-based product families designed with sustainability goals». En: *Journal of Systems and Software* 134 (2017), págs. 261-278 (vid. pág. 9).

[15] Michel RV Chaudron, Werner Heijstek y Ariadi Nugroho. «How effective is UML modeling?» En: *Software & Systems Modeling* 11.4 (2012), págs. 571-580 (vid. págs. 4, 5, 8, 12).

[16] Jacob Cohen. «Statistical power for the social sciences». En: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (vid. pág. 20).

[17] Kostadin Damevski, David Shepherd y Lori Pollock. «A field study of how developers locate features in source code». En: *Empirical Software Engineering* 21.2 (2016), págs. 724-747 (vid. pág. 9).

[18] Andrea De Lucia y col. «An experimental comparison of ER and UML class diagrams for data modelling». En: *Empirical Software Engineering* 15.5 (2010), págs. 455-492 (vid. pág. 4).

[19] Brian Dobing y Jeffrey Parsons. «How UML is used». En: *Communications of the ACM* 49.5 (2006), págs. 109-113 (vid. pág. 4).

[20] África Domingo y col. «Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2021, págs. 483-498 (vid. págs. 9, 24).

[21] África Domingo y col. «Evaluating the benefits of model-driven development». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2020, págs. 353-367 (vid. págs. 8, 24).

[22] África Domingo y col. «Evaluating the influence of scope on feature location». En: *Information and Software Technology* 140 (2021), pág. 106674. DOI: https://doi.org/10.1016/j.infsof.2021.106674 (vid. pág. 10).

[23] Jaime Font y col. «Achieving feature location in families of models through the use of search-based software engineering». En: *IEEE Transactions on Evolutionary Computation* 22.3 (2017), págs. 363-377 (vid. pág. 9).

[24] Jaime Font y col. «Feature location in models through a genetic algorithm driven by information retrieval techniques». En: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. 2016, págs. 272-282 (vid. pág. 9).

[25] Karl Frank. *Keeping your business relevant with Model Driven Architecture (MDA). The Borland advantage*. White Paper. Object Management Group (OMG), 2004 (vid. pág. 4).

[26] Tony Gorschek, Ewan Tempero y Lefteris Angelis. «On the use of software design models in software development practice: An empirical investigation». En: *Journal of Systems and Software* 95 (2014), págs. 176-193 (vid. págs. 4, 5, 8, 12).

[27] Martin Grossman, Jay E Aronson y Richard V McCarthy. «Does UML make the grade? Insights from the software development community». En: *Information and Software Technology* 47.6 (2005), págs. 383-397 (vid. págs. 4, 5, 8, 12).

[28] Brent Hailpern y Peri Tarr. «Model-driven development: The good, the bad, and the ugly». En: *IBM systems journal* 45.3 (2006), págs. 451-461 (vid. pág. 4).

[29] Werner Heijstek y Michel R V Chaudron. «Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process». En: *Conference Proceedings of the EUROMICRO*. 2009. ISBN: 9780769537849 (vid. pág. 6).

[30] John Hutchinson, Mark Rouncefield y Jon Whittle. «Model-driven engineering practices in industry». En: *Proceedings - International Conference on Software Engineering*. 2011. ISBN: 9781450304450 (vid. pág. 6).

[31] Wenbin Ji y col. «Maintaining feature traceability with embedded annotations». En: *Proceedings of the 19th International Conference on Software Product Line*. 2015, págs. 61-70 (vid. pág. 9).

[32] Howell Jordan y col. «Manually locating features in industrial source code: the search actions of software nomads». En: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, págs. 174-177 (vid. pág. 9).

[33] Natalia Juristo y Ana M Moreno. *Basics of software engineering experimentation*. Springer Science & Business Media, 2013 (vid. pág. 13).

[34] T Kapteijns y col. «A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare». En: *From code centric to model centric software engineering Practices Implications and ROI* (2009) (vid. pág. 6).

[35] Evrim Itir Karac, Burak Turhan y Natalia Juristo. «A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development». En: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589 (vid. pág. 20).

[36] Andrew J Ko y col. «An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks». En: *IEEE Transactions on software engineering* 32.12 (2006), págs. 971-987 (vid. pág. 9).

[37] K Krogmann y S Becker. «A Case Study on Model-Driven and Conventional Software Development : The Palladio Editor». En: *Software Engineering* (2007) (vid. pág. 6).

[38] Jacob Krüger y col. «Towards a better understanding of software features and their characteristics: a case study of marlin». En: *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. 2018, págs. 105-112 (vid. pág. 9).

[39] Ana C Marcén y col. «Towards feature location in models through a learning to rank approach». En: *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 2017, págs. 57-64 (vid. pág. 9).

[40] Nadja Christiane Marko y col. «Model-based engineering for embedded systems in practice». En: *Research reports in software engineering and management* (2014), págs. 1-48 (vid. pág. 5).

[41] Jabier Martinez y col. «Automating the extraction of model-based software product lines from model variants (T)». En: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2015, págs. 396-406 (vid. pág. 9).

[42] Yulkeidi Martínez, Cristina Cachero y Santiago Meliá. «MDD vs. traditional software development: A practitioner's subjective perspective». En: *Information and Software Technology*. 2013 (vid. pág. 6).

[43] Martínez, Yulkeidi and Cachero, Cristina and Meliá, Santiago. «MDD vs. traditional software development: A practitioner's subjective perspective». En: *Information and Software Technology* 55.2 (2013), págs. 189-200 (vid. págs. 4, 6).

[44] McNeile, Ashley. «MDA: The vision with the hole». En: *Metamaxim Ltd* (2003) (vid. pág. 4).

[45] Niklas Mellegård y Miroslaw Staron. «Distribution of effort among software development artefacts: An initial case study». En: *Lecture Notes in Business Information Processing*. 2010. ISBN: 9783642130502 (vid. pág. 6).

[46] Stephen J Mellor, Tony Clark y Takao Futagami. «Model-driven development: guest editors' introduction. IEEE Software, 20 (5). pp. 14-18. ISSN 0740-7459». En: *IEEE software* 20.5 (2003), págs. 14-18 (vid. pág. 4).

[47] Thomas O Meservy y Kurt D Fenstermacher. «Transforming software development: an MDA road map». En: *Computer* 38.9 (2005), págs. 52-58 (vid. pág. 4).

[48] Jose Ignacio Panach Navarrete y col. «Evaluating Model-Driven Development Claims with respect to Quality: A Family of Experiments». En: *IEEE Transactions on Software Engineering* (2018) (vid. pág. 6).

[49] Mari Carmen Otero y José Javier Dolado. «Evaluation of the comprehension of the dynamic modeling in UML». En: *Information and Software Technology* 46.1 (2004), págs. 35-53 (vid. pág. 4).

[50] Jose Ignacio Panach y col. «In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction». En: *Information and Software Technology* (2015). ISSN: 09505849 (vid. pág. 6).

[51] Paulo Eduardo Papotti y col. «A quantitative analysis of model -driven code generation through software experimentation». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2013, págs. 321-337 (vid. pág. 6).

[52] Oscar Pastor y Juan Carlos Molina. *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Springer Science & Business Media, 2007 (vid. pág. 5).

[53] Francisca Pérez y col. «Collaborative feature location in models through automatic query expansion». En: *Automated Software Engineering* 26.1 (2019), págs. 161-202 (vid. pág. 9).

[54] Francisca Pérez y col. «Comparing manual and automated feature location in conceptual models: A Controlled experiment». En: *Information and Software Technology* 125 (2020), pág. 106337 (vid. pág. 9).

[55] Francisca Pérez y col. «Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study». En: *Information and Software Technology* 103 (2018), págs. 188-201 (vid. pág. 9).

[56] Shari Lawrence Pfleeger. «Understanding and improving technology transfer in software engineering». En: *Journal of Systems and Software* 47.2-3 (1999), págs. 111-124 (vid. pág. 13).

[57] Denys Poshyvanyk y col. «Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval». En: *IEEE Transactions on Software Engineering* 33.6 (2007), págs. 420-432 (vid. pág. 9).

[58] Iris Reinhartz-Berger y Dov Dori. «OPM vs. UML–Experimenting with Comprehension and Construction of Web Application Models». En: *Empirical Software Engineering* 10.1 (2005), págs. 57-80 (vid. pág. 4).

[59] Meghan Revelle, Tiffany Broadbent y David Coppit. «Understanding concerns in software: insights gained from two case studies». En: *13th International Workshop on Program Comprehension (IWPC'05)*. IEEE. 2005, págs. 23-32 (vid. pág. 9).

[60] Bran Selic. «The pragmatics of model-driven development». En: *IEEE software* 20.5 (2003), págs. 19-25 (vid. pág. 4).

[61] Shane Sendall y Wojtek Kozaczynski. «Model transformation: The heart and soul of model-driven software development». En: *IEEE software* 20.5 (2003), págs. 42-45 (vid. pág. 4).

[62] Yashwant Singh y Manu Sood. «Model driven architecture: A perspective». En: *2009 IEEE International Advance Computing Conference*. IEEE. 2009, págs. 1644-1652 (vid. pág. 4).

[63] Harald Störrle. «How are Conceptual Models used in Industrial Software Development? A Descriptive Survey». En: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017, págs. 160-169 (vid. págs. 5, 8, 12).

[64] Walter F Tichy. «Should computer scientists experiment more?» En: *Computer* 31.5 (1998), págs. 32-40 (vid. págs. 5, 13).

[65] Sira Vegas, Cecilia Apa y Natalia Juristo. «Crossover designs in software engineering experiments: Benefits and perils». En: *IEEE Transactions on Software Engineering* 42.2 (2015), págs. 120-135 (vid. págs. 16, 19).

[66] Jinshui Wang y col. «An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions». En: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. 2011, págs. 213-222 (vid. pág. 9).

[67] Jinshui Wang y col. «How developers perform feature location tasks: a human-centric and process-oriented exploratory study». En: *Journal of Software: Evolution and Process* 25.11 (2013), págs. 1193-1224 (vid. pág. 9).

[68] Brady T West, Kathleen B Welch y Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman y Hall/CRC, 2014 (vid. pág. 19).

[69] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014 (vid. págs. 21-23).

[70] Norman Wilde y col. «A comparison of methods for locating features in legacy software». En: *Journal of Systems and Software* 65.2 (2003), págs. 105-114 (vid. pág. 9).

[71] Claes Wohlin y col. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (vid. págs. 5, 13, 18, 21).

[72] Andreas Zendler y col. «Experimental comparison of coarse-grained concepts in UML, OML, and TOS». En: *Journal of systems and Software* 57.1 (2001), págs. 21-30 (vid. pág. 4).

Parte II

Compendio de publicaciones

Capítulo 1

Evaluating the Benefits of Model-Driven Development

Researchers have been evaluating the benefits of Model-Driven Development (MDD) for more than a decade now. Although some works suggest that MDD decreases development time, other works limit MDD benefits to academic exercises and to developers without experience. To clarify the benefits of MDD, we present the results of our experiment, which compares MDD and Code-centric Development (CcD) in terms of correctness, efficiency, and satisfaction. Our experiment achieves fidelity to real-world settings because the tasks are taken from real-world video game development, and the subjects use domain frameworks as they are used in real-world developments. Our results challenge previous ideas that limit the benefits of MDD to academic exercises and to developers without experience. Furthermore, our results also suggest that understanding the benefits of MDD might require researchers to rethink their experiments to include the social part of software development.

Versión del autor del artículo: Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Evaluating the Benefits of Model-Driven Development: Empirical Evaluation Paper", in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2020, vol. 12127 LNCS, pp. 353-367, doi: 10.1007/978-3-030-49435-3_22.

1.1 Introduction

Model-Driven Development (MDD) [19] promotes software models as the cornerstone of software development. In comparison to popular programming languages, these software models are less bound to the underlying implementation and are closer to the problem domain. Model transformation is at the heart of MDD since MDD aims to generate the software code from the models. This generation ranges from skeleton code to fully functional code systems.

For more than a decade, researchers have been evaluating the benefits of MDD [2, 1, 11, 6, 8, 13, 7, 12, 18, 17, 16]. Some works [11, 18, 6] conclude that MDD decreases development time (up to 89 %) relative to Code-centric Development (CcD) [11]. Other works [17, 8] suggest that gains might only be achieved in academic exercises. Furthermore, other works [8, 12] assert that only developers without experience benefit from MDD. Therefore, more experimentation is needed to clarify the benefits of MDD.

In the context of MDD, domain frameworks (bodies of prewritten code) help model transformations to fill the abstraction gap between models and code (see Fig 1 left). These frameworks are not exclusive of MDD. In the context of CcD, developers also use frameworks to accelerate development (see Fig 1 right). However, previous experiments neglect the use of domain frameworks. This triggers the question of whether MDD benefits would hold when frameworks are considered.

In this work, we present our experiment, which compares MDD and CcD in terms of correctness, efficiency, and satisfaction. The tasks of our expe-

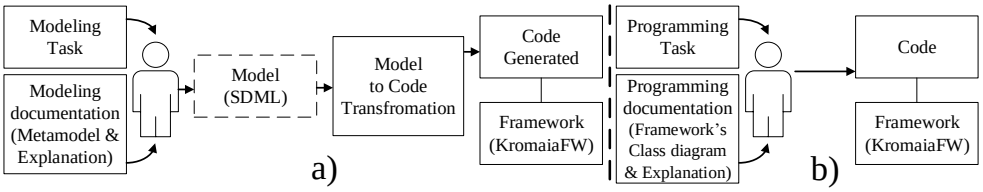

Figure 1.1: a) Artifacts in the MDD task b) Artifacts in the CcD task

riment are extracted from the real-world software development tasks of a commercial video game (Kromaia[1] released on PlayStation 4 and Steam). A total of 44 subjects (classified in two groups based on development experience) performed the tasks of the experiment. In our experiment, both MDD and CcD leverage a domain framework (see Fig 1.1).

Our results challenge previous ideas that suggest that MDD gains are only achieved in academic settings. In our experiment with tasks from real-world development, MDD improved correctness and efficiency by 41 % and 54 % respectively. The results also contradict the previous claim that MDD benefits are limited to developers without experience. In the experiment, developers without experience (49 % and 69 %) as well as developers with experience (39 % and 54 %) significantly improved their results with MDD.

Furthermore, the results also uncover a paradox. Despite the significant improvements in correctness and efficiency, the intention of use of MDD did not achieve maximum values. We think this should influence future experiments to go beyond the technical facet and explore the cultural aspect of software development.

The rest of the paper is organized as follows: Section 1.2 reviews the related work. Section 1.3 describes our experiment, and Section 1.4 shows the results. Section 1.5 describes the threats to validity. Finally, Section 1.6 concludes the paper.

[1] https://youtu.be/EhsejJBp8Go

1.2 Related Work

In the Motorola Case Study, Baker et al. [2] present their experiences deploying a top-down approach to MDD for more than 15 years in a large industrial context, resulting in a set of high-level strategies to enhance MDD deployment and success. Anda and Hansen [1] analyze the use of MDD with UML in software development companies. They show that developers who applied UML in modelling and enhancing legacy software experienced more challenges than those who modelled and developed new software from scratch. They propose a need for better methodological support in applying UML in legacy development.

Krogmann and Becker [11] present one of the first comparisons between MDD and CcD software development with respect to effort. Despite its limitations (it is not as a controlled experiment), their case study compares two software development projects, each of which was developed using a different method. They conclude that the model-driven approach could be carried out in only 11 % of the time that of the code-centric approach. Another comparative case study of model-driven development with code-centric development by Kapteijns et al. [8] claims that MDD can be successfully applied to small-scale development projects under easy conditions. Heijstek and Chaudron [6] focused their case study on report specific metrics to analyze model size, complexity, quality, and effort. They analyze an industrial MDD project that was developed for the equivalent of 28 full-time team members. They showed that an increase in productivity, a reduction in complexity, and benefits from a consistent implementation are attributed to the use of MDD techniques. Mellegård and Staron [13] analyze the main modelling artefacts in the analysis and design phase of projects with respect to required effort and perceived importance. They conclude that the distribution of effort between models and other artefacts is similar in code-centric development to that of model-driven development. The factors for a successful adoption of MDD in a company were analyzed later by Hutchinson et al. [7] through interviews of 20 subjects from three different companies. They claim that a progressive and iterative approach, transparent organizational commitment and motivation, integration with existing organizational

processes, and a clear business focus are required to guarantee success in the adoption of MDD techniques.

The experiment conducted by Martínez et al. [12] with undergraduate students compares three methods: Model-driven, Model-Based, and Code-Centric, regarding perceived usefulness, ease of use, intention of use, and compatibility. They conclude that the Model-Driven method is considered to be the most useful one, although it is also considered to be the least compatible with previous developers' experiences. Pappoti et al. [18] also present an experiment with a group of students in which an MDD based approach using code generation from models is compared with manual coding. When code generation was applied, the development time was consistently shorter than with manual coding. The participants also indicated that they had fewer difficulties when applying code generation. For Panach et al. [17], the benefits of developing software with MDD depend on the development context characteristics, such as problem complexity and the developers' background experience with MDD. However, in a later work [16], where they report the results of six replications of the same experiment, they confirm that MDD yields better quality independently of problem complexity, and that the effect is bigger when the problems are more complex.

Table 3.1 summarizes the related work. In contrast to previous works, we address the use of a domain framework as part of both MDD and CcD. This dimension has not been explored before. This contributes to achieving fidelity to real-world development since domain frameworks are fairly popular in both MDD and CcD contexts.

1.3 Experiment design

1.3.1 Objectives

According to the guidelines for reporting software engineering experiments [22], we have organized our research objectives using the Goal Question Metric template for goal definition, which was originally presented by Basili and Rombach [3]. Our goal is to:

Table 1.1: Empirical studies on MDD

Work	Modelling Language	Domain Framework	Sample size	Context	Type of Study	Variables
[2]	UML	No	Not given	Industry	Case Study	Quality | Productivity
[1]	UML	No	28	Industry	Experiment	Difficulty | Use | Utility
[11]	DSL (GMFML)	No	11	Academia	Case Study	Quality | Efficiency Time effort
[6]	UML	No	4	Industry	Case Study	Quality | Effort Size | Complexity
[8]	UML	No	1	Industry	Case Study	Quality | Productivity Maintainability
[13]	UML	No	3	Industry	Case Study	Effort
[7]	UML	No	20	Industry	Case Study	Factors for a successful adoption of MDD
[12]	UML DSL (OOH4RIA)	No	26	Academia	Experiment	Perceived usefulness Perceived ease of use Intention to adopt Compatibility
[18]	UML	No	29	Academia	Experiment	Efficiency | Effort Participants' opinion
[17, 16]	UML	No	26	Academia	Experiment	Quality |Effort Satisfaction |Productivity
This work	DSL(SDML)	Yes	44	Academia[2]	Experiment	Correctness | Efficiency Satisfaction

Analyze software development methods, **for the purpose of** comparison, **with respect to** correctness of the software developed, efficiency, and user satisfaction; **from the point of view of** novice and professional developers, **in the context of** developing software for a video game company.

1.3.2 Variables

In this study, the independent variable is the software development method (*Method*). It has two values, MDD and CcD, which are the methods used by subjects to solve the tasks.

Given that our experiment evaluates the benefits of MDD, and the most reported benefit of MDD is decreased development time, we consider two dependent variables, *Correctness* and *Efficiency*, which are related to the software that is developed. *Correctness* was measured using a correction template, which was applied to the programming artifacts developed by the participants after the experiment. *Correctness* was calculated as the percentage of passing assert statements with respect to the total num-

ber of assert statements. To calculate *Efficiency*, we measured the time employed by each subject to finish the task. *Efficiency* is the ratio of *Correctness* to time spent (in minutes) to perform a task.

We measured users' satisfaction using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [14], which is used for validating Information System Design Methods. We decompose satisfaction into three dependent variables as follows: *Perceived Ease of Use* (PEOU), the degree to which a person believes that learning and using a particular method would require less effort. *Perceived Usefulness* (PU), the degree to which a person believes that using a particular method will increase performance, and *Intention to Use* (ITU), the degree to which a person intents to use a method. Each of these variables corresponds to specific items in the TAM questionnaire. We averaged the scores obtained for these items to obtain the value for each variable.

1.3.3 Design

Since the factor under investigation in this experiment is the software development method, we compared MDD and CcD. In order to improve experiment robustness regarding variation among subjects [21], we chose a repeated measurement using the largest possible sample size. To avoid the order effect, we chose a crossover design and we used two different tasks, T1 and T2. All of the subjects used the two development methods, each one of which was used in a different task.

The subjects had been randomly divided into two groups (G1 and G2). In the first part of the experiment, all of the subjects solved T1 with G1 using CcD and G2 using MDD. Afterwards, all of the subjects solved T2, G1 using MDD and G2 using CcD.

Table 1.2: Results of the demographic questionnaire

	Age±σ	**Experience**±σ	**Code time**±σ	**Model time**±σ	**DSL Know**±σ	**PL Know**±σ
Undergraduate	22.2±0.4	0.6±1.4	1.4± 0.8	1.0±0.4	3±1.7	4.2 ±1.8
Masters	27±2.6	4.3±3.2	6.2±2.0	0.9±0.3	3.4±1.0	6.2±1.2
Total	21.6±3.2	1.4±2.4	2.4±2.3	1±0.4	3.1±1.7	4.6±1.9

1.3.4 Participants

The subjects were selected according to convenience sampling [22]. A total of 44 subjects performed the experiment. There were 35 second-year undergraduate students from a technological program and 9 masters students in a subject about advanced software modelling currently employed as professional developers. The undergraduate students were novice developers and the master students were professional developers.

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 1.2 shows the mean and standard deviation of age, experience, hours per day developing software (Code Time) and hours per day working with models (Model Time). On average, all of the masters students had worked four years developing software. They worked on software development six hours per day while the undergraduate students, on average, dedicated less than 1.5 hour to developing software each day. We used a Likert scale form 1 to 8 to measure the subjects' knowledge about domain-specific languages (DSL know) and programming languages (PL know). The mean and standard deviation of their answers are also in Table 1.2. All of them evaluated higher their programming language knowledge than their ability with models.

The experiment was conducted by two instructors and one video game software engineer (the expert), who designed the tasks, prepared the correction template, and corrected the tasks. The expert provided information about both the domain-specific language and the domain framework. During the experiment, one of the instructors gave the instructions and managed the focus groups. The other instructor clarified doubts about the experiment and took notes during the focus group.

1.3.5 Research questions and hypotheses

We seek to answer the following three research questions:

RQ1 Does the method used for developing software impact the *Correctness* of code? The corresponding null hypothesis is H_{C0}: The software development method does not have an effect on *Correctness*.

RQ2 Does the method used for developing software impact the *Efficiency* of developers to develop software? The null hypothesis for *Efficiency* is H_{E0}: The software development method does not have an effect on *Efficiency*.

RQ3 Is the user satisfaction different when developers use different methods of software development? To answer this question we formulated three hypotheses based on the variables *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*: H_{PEOU}, H_{PU} and H_{ITU} respectively. The corresponding null hypotheses are:

H_{PEOU0}: The software development method does not have an effect on *Perceived Ease of Use*.

H_{PU0}: The software development method does not have an effect on *Perceived Usefulness*

H_{ITU0}: The software development method does not have an effect on *Intention to Use*.

The hypotheses are formulated as two-tailed hypotheses since not all of the empirical or theoretical studies support the same direction for the effect.

1.3.6 Experiment procedure

The diagram in Fig. 1.2) shows the experiment procedure that can be summarises as follows:

1. The subjects received information about the experiment. An instructor explained the parts in the session, and he advised that it was not a test of their abilities.

2. A video game software engineer explained to the subjects the problem context and how to develop game characters on a video game with the domain framework to be used later in the experiment. The average time spent on this tutorial was 30 minutes.

3. The subjects completed a demographic questionnaire. One of the instructors distributed and collected the questionnaires, verifying that all of the fields had been answered and that the subject had signed the voluntary participation form in the experiment.

4. The subjects received clear instructions on where to find the statements for each task, how to submit their work, and how to complete the task sheet and the satisfaction questionnaire at the end of each task.

5. The subjects performed the first task. The subjects were randomly divided into two groups (G1 and G2) to perform the tasks with the domain framework. The subjects from G1 developed the first task coding with C++, and the subjects from G2 developed the task using MDD. The instructors used the distribution in the room to distinguish one group from another and to give specific guidance to each subject if requested.

6. The subjects completed a satisfaction questionnaire about the method used to perform the task.

7. The subjects answered an open-ended questionnaire about the method used to perform the task.

8. An instructor checked that each subject had filled in all of the fields on the task sheet and on the satisfaction questionnaire.

9. The subjects performed the second task exchanging methods. In other words, the subjects from G1 performed the second task using MDD, and the subjects from G2 performed the task using C++. Then, the subjects filled out the satisfaction questionnaire and the open-ended questionnaire that were related to the method used.

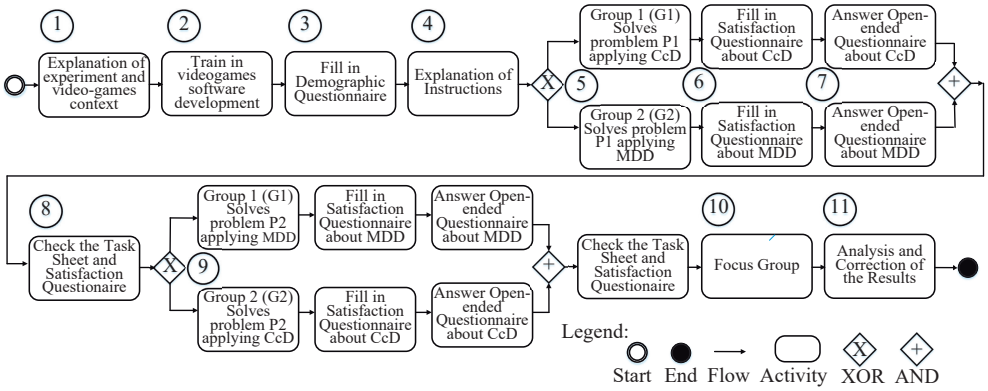

Figure 1.2: Experimental Procedure

10. A focus group interview about the tasks was conducted by one instructor while the other instructor took notes.

11. Finally, the video game software engineer corrected the tasks and an instructor analyzed the results.

In the tasks, the subjects were requested to develop the code of a part of the Kromaia video game, specifically a different game boss for each task. On average, the tasks took about 50 model elements of Shooter Definition Model Language (MDD), and about 300 lines of code (CcD). More information on the development of the Kromaia game bosses can be found at `https://youtu.be/Vp3Zt4qXkoY`. The materials used in this experiment (the training material, the consent to process the data, the demographic questionnaire, the satisfaction questionnaire, the open-ended questionnaire, the task sheet and the materials used in the focus group) are available at `http://svit.usj.es/MDD-experiment`.

The experiment was conducted on two different days at *Universidad San Jorge* (Zaragoza, Spain) by the same instructors and a video game software engineer. On the first day, the experiment was performed by the masters students. On the second day, the undergraduate students performed the experiment. The masters and undergraduate students did not know each other. Their schedules at the university were completely dif-

ferent and the programs they followed are aimed at different types of student profiles.

1.4 Results

For the data analysis we have chosen the Linear Mixed Model (LMM) test [20] also used in others crossover experiments in software engineering [9]. LMM handles correlated data resulting from repeated measurements. The dependent variables for this test are *Correctness*, *Efficiency*, *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*. In our study, the subjects are random factors and the *Method* used to develop software (MDD or CcD) is a fixed factor (the primary focus of our investigation). The statistical model (Model 1) used in this case is described as:

$$DV \sim Method + (1|\, subject) \tag{1.1}$$

Additionally, the subjects experience is also considered to be a fixed effect. We consider the variables *Experience* and the sequence *Method* and *Experience* to be fixed effects to account for their potential effects in determining the main effect of *Method* [9]. The statistical model (Model 2) used in this case is described in the following formula:

$$DV \sim \quad Method + Experience + Method * Experience + (1|\, Subject) \tag{1.2}$$

The statistical model fit for each variable has been evaluated based on goodness of fit measures such as Akaike's information criterion (AIC) and Schwarz's Bayesian information criterion (BIC). The model with the smaller AIC or BIC is considered to be the better fitting model. Additionally, the variance explained in the dependent variables by the statistical models is evaluated in terms of R^2 [15, 9].

To quantify the difference between MDD and CcD, we have calculated the effect size using the means and standard deviations of the dependent variables of each method to obtain the standardized difference between the two means, Cohen's d Value [4]. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. This value also allows

Table 1.3: Results of test of fixed effects for each variable and each model

	Model 1	Model 2		
	Method	*Method*	*Experience*	*Method*Experience*
Correctness	(F=643.3,p=.000)	(F=137.7,p=.000)	(F=2.5,p=.120)	(F=1.9,p=.175)
Efficiency	(F=1084.4,p=.000)	(F=83.8,p=.000)	(F=1.9,p=.171)	(F=.6,p=.447)
Ease of use	(F=451.7,p=.000)	(F=14.2,p=.001)	(F=.4,p=.508)	(F=.12,p=.730)
Usefulness	(F=545.7,p=.000)	(F=9.97,p=.003)	(F=2.0,p=.168)	(F=0.0,p=.965)
Intention to Use	(F=341.4,p=.000)	(F=5.3,p=.026)	(F=.2,p=.689)	(F=1.1,p=.965)

us to measure the percentage of overlaps between the distributions of the variables for each method.

We have selected box plots and histograms to describe the data and the results.

1.4.1 Hypothesis testing

The results of the Type III test of fixed effects [3] for the fixed factors for each one of the statistical models used in the data analysis are shown in Table 1.3.

For all of the variables, *Method* obtained p-values less than 0.05, regardless of the statistical model used for its calculation. Therefore, all the null hypotheses are rejected. Thus, the answers to the research questions **RQ1**, **RQ2** and **RQ3** are affirmative. The method used for developing software has a significant impact on the correctness of code, efficiency, and the satisfaction of developers.

However, the fixed factors *Experience* and *Method*Experience* obtained p-values greater than 0.05, which implies that neither the developers experience nor the combination of both fixed factors had a significant influence on the changes in correctness of code, or the efficiency and the satisfaction of the developers.

[3] Type III test of fixed effects is the default test which enables LMM to produce the exact F-values and p-values for each dependent variable and each fixed factor.

Table 1.4: Comparison of alternative models for each variable

	Variance explained		**Model fit**			
	R^2		AIC		BIC	
	Model 1	Model 2	Model 1	Model 2	Model 1	Model 2
Correctness	63.8 %	65.7 %	-59.617	-56.313	-49.847	-46.638
Efficiency	57.9 %	58.6 %	-412.078	-398.820	-402.261	-398.097
Ease of use	4.8 %	14.1 %	232.593	232.281	241.917	244.004
Usefulness	10.9 %	13.7 %	218.295	217.452	228.113	227.176
Intention to Use	4.8 %	6.2 %	270.298	268.847	280.115	278.570

1.4.2 Statistical model Validity and Fit

The use of the Linear Mixed Model test assumed that residuals must be normally distributed. The normality of the errors had been verified by the Shapiro-Wilk test and visual inspections of the histogram and normal Q-Q plot. All of the residuals, except the ones carried out for *Efficiency*, obtained a p-value greater than 0.05 with the normality test. We obtained normally distributed residuals for *Efficiency* by using square root transformation. For the statistical analysis of the variable *Efficiency* with LMM, we used $DV = sqrt(Efficiency)$ in formulas (1) and (2). For the rest of the variables, DV is equal to their value.

The assessment of statistical model fit is summarized in Table 1.4. The values of the fit statistics R^2, AIC and BIC for each one of the statistical models are listed for each dependent variable. The fraction of total variance explained by both statistical models is similar. Model 2 obtained slightly better values of R^2, but the difference is not big. The AIC and BIC criteria were smaller for Model 1 in *Correctness* and *Efficiency*, and the difference was small for the other variables in favor of Model 2. This suggests that the *Method* factor explains much of the variance in the dependent variables. The factors incorporated in Model 2 with respect to Model 1 did not have a significant influence on the changes in the correctness of code, or the efficiency and satisfaction of the developers, as we have reported in section 1.4.1.

1.4.3 Effect size

The effect size of a Cohen d value of 2.63 for *Correctness* indicates that the magnitude of the difference is large. The mean of *Correctness* for MDD is 2.63 standard deviations bigger than the mean of *Correctness* for CcD. The mean for MDD is the 99.5 percentile of the mean for CcD. The box plots in Fig. 1.3(a) illustrate this result. This means that the distributions only have 9.7 % of their areas on common, as shows Fig. 1.3(c).

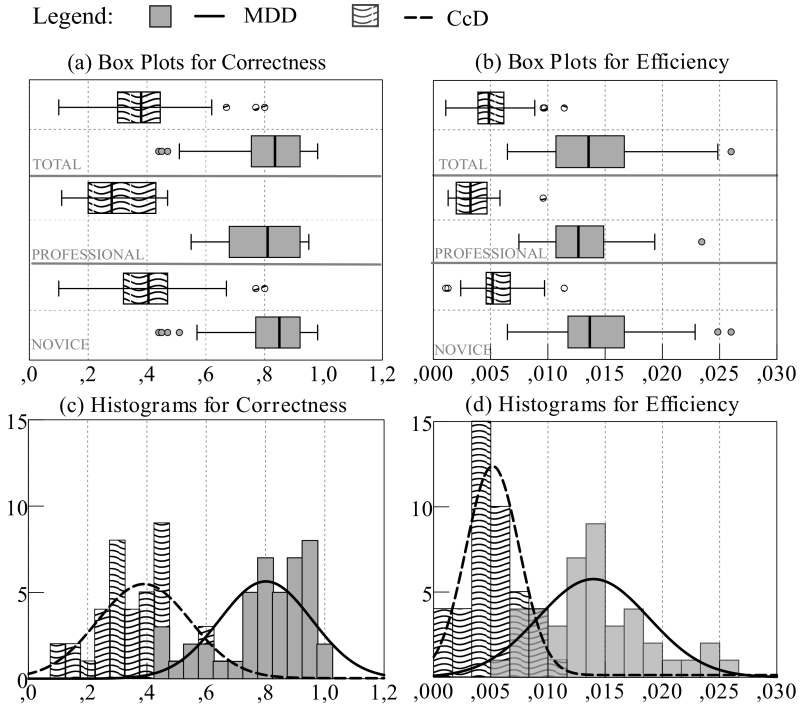

Figure 1.3: Box plots and histograms for *Correctness* and *Efficiency*: (a) and (b) for *Correctness*; (c) and (d) for *Efficiency*

There is also a large effect size with a Cohen d value of 2.33 for *Efficiency*. The magnitude of the difference is also large. The mean for *Efficiency* for MDD, is 2.33 standard deviations bigger than the mean for efficiency for CcD. Again the mean for MDD is the 99.5 percentile of the mean for CcD (Fig. 1.3(b)) and the distributions of efficiency are different for 88 % of their areas (Fig. 1.3(d)). The effect size of the differences based

on using MDD or CcD for *Perceived Ease of Use* is medium-high, with a Cohen d value of 0.78. The box plots of Fig. 1.4(a) and the histograms of Fig. 1.4(d) illustrate how the differences in *Perceived Ease of Use* are not as great as the ones for *Correctness* or *Efficiency*. The magnitude of the difference between methods of development decreases to medium in the case of *Perceived Usefulness* with a Cohen d value of 0.69. Both the box plots of Fig. 1.4(b) and the histograms of Fig. 1.4(e) illustrate a similar distribution to the one for *Perceived Ease of Use*. Again, there is not a big difference in the diagrams corresponding to each subject group (professionals or novices)

Intention to Use obtained the lowest Cohen's d value (0.445), which means that the effect size of the method in this case is small to medium: the histograms of Fig. 1.4(f) shows that the distributions have much in common. The box plots of Fig. 1.4(c) shows that, in this case, the difference in the mean scores of *intention of use* (in favor of MDD versus CcD), is greater for the group of professionals than for the group of novices or the total group.

These data allow us to give more precise answers to **RQ1**, **RQ2**, and **RQ3**: for *Correctness* and *Efficiency*, the impact of the method used for development is very large, while, for satisfaction, the magnitude of the difference is medium.

1.4.4 Interpretation of the results

Nowadays, domain frameworks are widely used in software development. The pre-implemented frameworks save developers time by using implementations that had been developed previously. In previous comparisons between MDD and CcD, one may argue that the benefits of MDD might come from the lack of a domain framework for CcD. This was not the case in our experiment.

Benefits must come from the software model itself and the model transformation (from model to code) that MDD adds to the framework. We still do not know if the key lies on the abstraction of models, the automation of transformations, or a combination of these. However, it turns

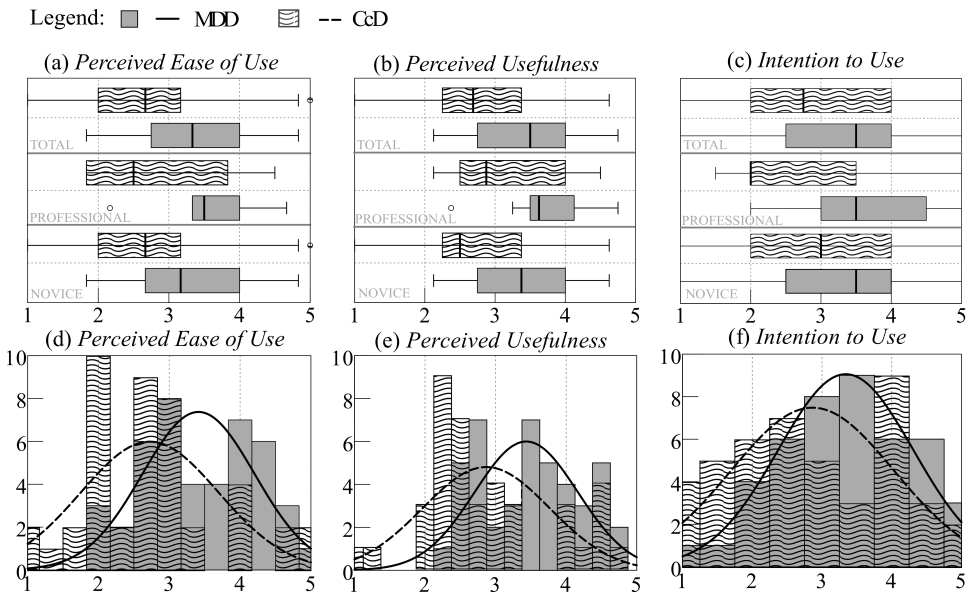

Figure 1.4: Box plots and histograms for satisfaction: (a) and (d) for *Perceived Ease of Use*; (b) and (e) for *Perceived Ease of Use*; and (c) and (f) for *Intention to Use*

out that a framework on its own is not enough to achieve the benefits of MDD.

In our focus group interviews, both the professionals and the novices agreed that the abstraction of models is a double-edged sword. On the one hand, the subjects stated that models empower them to focus on the task at hand. On the other hand, the subjects stated that they lose control of the code generated. The subjects acknowledge that this loss of control negatively influences their intention of use. A few subjects stated that this loss of control would be alleviated if the model transformation were considered as another developer of the team. This triggers an interesting new direction of research, exploring the social implications of MDD on development teams.

1.5 Threats To Validity

To describe the threats of validity of our work, we use the classification of [22].

Conclusion validity. The *low statistical power* was minimized because the confidence interval is 95 %. To minimize the *fishing and the error rate* threat, the tasks and corrections were designed by a video game software engineer. Furthermore, this engineer corrected the tasks. The *Reliability of measures* threat was mitigated because the measurements were obtained from the digital artefacts generated by the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the treatment implementation was identical in the two sessions. Also, the tasks were designed with similar difficulty. Finally, the experiment was affected by the *random heterogeneity of subjects* threat. The heterogeneity of subjects allowed us to increase the number of subjects in the experiment.

Internal validity. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience in software development. To mitigate this threat, the treatment was applied randomly. Other threat was *compensatory rivalry*: the subjects may have been motivated to perform the task with a higher level of quality by using the treatment that was more familiar to them.

Construct validity. *Mono-method bias* occurs due to the use of a single type of measure [17]. All of the measurements were affected by this threat. To mitigate this threat for the correctness and efficiency measurements, an instructor checked that the subjects performed the tasks, and we mechanized these measurements as much as possible by means of correction templates. We mitigated the threat to satisfaction by using a widely applied model (TAM) [5]. The *hypothesis guessing* threat appears when the subject thinks about the objective and the results of the experiment. To mitigate this threat, we did not explain the research questions or the experiment design to the subjects. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To weaken this threat, at the beginning of the experiment the instructor explained to the subjects that the experiment was not a test about their abilities.

Author bias occurs when the people involved in the process of creating the experiment artifacts subjectively influence the results. In order to mitigate this threat, the tasks were balanced, i.e., their sizes and difficulty were the same for all treatments. Furthermore, the tasks were extracted from a commercial video game. Finally, the *mono-operation bias* threat occurs when the treatments depend on a single operationalization. The experiment was affected by this threat since we worked with a single treatment.

External validity. The *interaction of selection and treatment* threat is an effect of having a subject that is not representative of the population that we want to generalize. However, using students as subjects instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts [10]. It would be necessary to replicate the experiment with different subject roles in order to mitigate this threat. The *domain* threat appears because the experiment has been conducted in a specific domain, i.e., video games development. We think that the generalizability of findings should be undertaken with caution. Other experiments in different domains should be performed to validate our findings.

1.6 Conclusion

In this work, we present an experiment that compares MDD and CcD in terms of correctness, efficiency, and satisfaction. Our experiment goes beyond the state of the art in terms of real-world fidelity and statistical power. A higher fidelity to real-world software development is achieved by means of the use of domain frameworks as they are used in real-world developments. Statistical power is enhanced by increasing the sample size. Our results challenge previous ideas that limit the benefits of MDD to academic exercises and to developers without experience. Furthermore, our results also suggest a new research direction that should include the social aspect of software development in order to better understand the benefits of MDD.

Bibliografía

[1] Bente Anda y Kai Hansen. «A case study on the application of UML in legacy development». En: *ISESE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering*. 2006. ISBN: 1595932186 (vid. págs. 36, 38, 40).

[2] Paul Baker, Shiou Loh y Frank Weil. «Model-Driven Engineering in a Large Industrial Context — Motorola Case Study». En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2005. ISBN: 3540290109 (vid. págs. 36, 38, 40).

[3] Victor R. Basili y H. Dieter Rombach. «The TAME Project: Towards Improvement-Oriented Software Environments». En: *IEEE Transactions on Software Engineering* (1988). ISSN: 00985589 (vid. pág. 39).

[4] Jacob Cohen. «Statistical power for the social sciences». En: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (vid. pág. 46).

[5] Fred D. Davis. «Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology». En: *MIS Q.* 13.3 (sep. de 1989), págs. 319-340. ISSN: 0276-7783 (vid. pág. 52).

[6] Werner Heijstek y Michel R V Chaudron. «Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process». En: *Conference Proceedings of the EUROMICRO*. 2009. ISBN: 9780769537849 (vid. págs. 36, 38, 40).

[7] John Hutchinson, Mark Rouncefield y Jon Whittle. «Model-driven engineering practices in industry». En: *Proceedings - International Conference on Software Engineering*. 2011. ISBN: 9781450304450 (vid. págs. 36, 38, 40).

[8] T Kapteijns y col. «A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare». En: *From code centric to model centric software engineering Practices Implications and ROI* (2009) (vid. págs. 36, 38, 40).

[9] Evrim Itir Karac, Burak Turhan y Natalia Juristo. «A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development». En: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589 (vid. pág. 46).

[10] Barbara A. Kitchenham y col. «Preliminary guidelines for empirical research in software engineering». En: *IEEE Transactions on Software Engineering* 28.8 (ago. de 2002), págs. 721-734. ISSN: 0098-5589 (vid. pág. 53).

[11] K Krogmann y S Becker. «A Case Study on Model-Driven and Conventional Software Development : The Palladio Editor». En: *Software Engineering* (2007) (vid. págs. 36, 38, 40).

[12] Yulkeidi Martínez, Cristina Cachero y Santiago Meliá. «MDD vs. traditional software development: A practitioner's subjective perspective». En: *Information and Software Technology*. 2013 (vid. págs. 36, 39, 40).

[13] Niklas Mellegård y Miroslaw Staron. «Distribution of effort among software development artefacts: An initial case study». En: *Lecture Notes in Business Information Processing*. 2010. ISBN: 9783642130502 (vid. págs. 36, 38, 40).

[14] Daniel L Moody. «The method evaluation model: a theoretical model for validating information systems design methods». En: *ECIS 2003 proceedings* (2003), pág. 79 (vid. pág. 41).

[15] Shinichi Nakagawa y Holger Schielzeth. «A general and simple method for obtaining R2 from generalized linear mixed-effects models». En: *Methods in ecology and evolution* 4.2 (2013), págs. 133-142 (vid. pág. 46).

[16] Jose Ignacio Panach Navarrete y col. «Evaluating Model-Driven Development Claims with respect to Quality: A Family of Experiments». En: *IEEE Transactions on Software Engineering* (2018) (vid. págs. 36, 39, 40).

[17] Jose Ignacio Panach y col. «In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and

satisfaction». En: *Information and Software Technology* (2015). ISSN: 09505849 (vid. págs. 36, 39, 40, 52).

[18] Paulo Eduardo Papotti y col. «A quantitative analysis of model -driven code generation through software experimentation». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2013, págs. 321-337 (vid. págs. 36, 39, 40).

[19] Bran Selic. «The pragmatics of model-driven development». En: *IEEE software* 20.5 (2003), págs. 19-25 (vid. pág. 36).

[20] Brady T West, Kathleen B Welch y Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman y Hall/CRC, 2014 (vid. pág. 46).

[21] Norman Wilde y col. «A comparison of methods for locating features in legacy software». En: *Journal of Systems and Software* (2003). ISSN: 01641212 (vid. pág. 41).

[22] Claes Wohlin y col. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (vid. págs. 39, 42, 52).

Capítulo 2

Comparing UML-based and DSL-based Modeling from Subjective and Objective Perspectives

In the last two decades, researchers have conducted several empirical evaluations, involving thousands of subjects, to understand the use of models in software development. The results of these evaluations show that most of the subjects make informal use of the models, which is known as 'modeling as sketch'. In this paper, we present an experiment that compares UML-based and DSL-based modeling when subjects model a part of a commercial video game. In the comparison, we have used objective and subjective measures, in contrast to other works that focus either on objective measures to evaluate modeling performance or on subjective measures to analyze modeling styles. Our results reveal that subjects underestimate the potential of their own models. Our finding is relevant for the design of future evaluations and for the teaching and adoption of modeling. If users correctly assess their models, they might leverage their potential as programs.

Versión del autor del artículo: Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives", in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2021, vol. 12751 LNCS, pp. 483-498, doi: 10.1007/978-3-030-79382-1_29.

2.1 Introduction

The software engineering process goes from the initial idea for building a particular software (problem space) to the code that implements the software (solution space) through different transformations. The idea is first conceptualized through models, and then the models are transformed into fully functional code. To understand the use of modeling languages in software development, in the last two decades, researchers have conducted surveys [17, 13, 1, 16, 20, 27, 2, 8], case studies [4, 26, 8], experiments [31, 23, 25, 11, 5, 8], and interviews [8] involving more than 5,355 subjects. The researchers classified the styles of modeling into three categories: *sketch* (informal models to aid in communication, see Figure 2.1.a), *blueprint* (models as the basis for programmers to create code, see Figure 2.1.b), and *programs* (models that include all of the details needed to generate code, see Figure 2.1.c). All of the works that evaluated the style of modeling [17, 8, 16, 27, 2] concur that most of the subjects make an informal use of the models, which is known as 'modeling as sketch'. Despite the benefits of using models as programs [14], models are still mainly used as sketch.

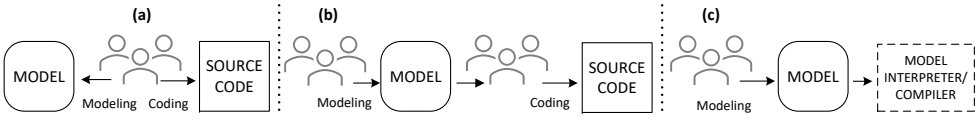

Figure 2.1: (a) Models as sketch, (b) Models as blueprint, (c) Models as programs

The Unified Modeling Language (UML) is a general-purpose modeling language that has become the 'de facto' standard for modeling software

systems [17, 7, 8]. Despite the available evidence about the efficiency of UML modeling, UML shortcomings have been identified by several authors [17, 4, 13, 26, 7, 8, 1]. In addition, to reduce the complexity of software development, the use of domain-specific languages (DSLs), languages closer to the problem domain, has proven to outperform the use of general-purpose programming languages [19]. We can also find empirical works that indicate that the use of a DSL can reduce the effort of developers when working in modeling performance testing tasks [5].

In empirical research, comparing UML against DSL is an open problem towards understanding software modeling [22, 7]. Previous works [19, 5] suggest that the usage of DSLs outperforms the usage of other software artifacts. Our own previous work in the field [14] points in the same direction. Thus far, works that have compared UML against other modeling languages [5, 11, 25, 23, 31] have focused solely on objective measures. In this work, we combine objective measures (correctness and efficiency) with the subjective assessment of subjects (model style classification and satisfaction) through a crossover experiment that compares UML vs DSL when subjects model an element of a commercial video game. Game software engineering has been identified as a knowledge area that needs more fundamental research [3]. Moreover, despite video-game development being one of the fastest growing industries, there are few works in the field, which makes it an original domain for exploring modeling adoption.

Building upon previous work we hypothesize that the usage of a DSL will outperform the usage of UML models in the video-games domain. The findings of this empirical study not only confirm this hypothesis, but in addition, reveal a problem that was not found in evaluations performed by other authors: no matter how correct the models are, and their ability to actually generate software, the subjects think of them more as sketches than as programs. Our results are useful for teaching modeling and model adoption because they put the focus on the problem of subjects underestimating the potential of their own models.

The rest of the paper is organized as follows: Section 2.2 reviews the related work. Section 2.3 describes our experiment, and Section 2.4 shows the results. Section 2.5 describes the threats to validity. Finally, Section 2.6 concludes the paper.

2.2 Related Work

Modeling languages have been identified as a key challenge to increase the adoption of models as programs [22], and the use of DSLs has been identified as more effective and efficient than general-purpose programming languages [19]. However, it is difficult to find empirical works that compare UML with an alternative modeling language. Budgen [7] explained in 2011 that this was because of the underlying assumption that UML did not require testing because it was a de facto standard. The oldest comparison work we have found dates from 2001. Zendler et al. [31] compared three object-oriented approaches (UML, Open modeling language [OML], and taxonomic object system [TOS]) in two different application systems with respect to coarse-grained modeling concepts. They conclude that the coarse-grained concepts of the object-oriented approaches OML and TOS were superior to those of UML when modeling a database-oriented application.

In 2004, Otero and Dolado [23] presented the results of a controlled experiment comparing the comprehension of UML and OML diagrams in the design of a real-time embedded system. They found that the specification of dynamic data is faster and easier to comprehend in OML than in UML. A year later, Iris Reinhartz-Berger and Dov Dori [25] compared UML with Object-Process Methodology (OPM) with respect to the level of comprehension and the quality of Web application models. The results of their experiment suggest that OPM is better than UML when modeling web applications, especially in the dynamics aspects and in the quality of the models.

In 2010, De Lucia et al. [11] presented the results of three sets of controlled experiments comparing UML class diagrams and Entity-Relationship (ER) diagrams with respect to the comprehension and the interpretation of data models during maintenance activities. The results demonstrated that the two notations gave the same support, except during verification activities when UML class diagrams provided better support than ER diagrams. In 2016, Bernardino et al. [5] presented the results of an experiment about the benefits and drawbacks when using UML or a DSL for

modeling performance testing in an IT company. Their results indicate that the effort using a DSL was lower than using UML.

Table 2.1: Empirical studies on modeling adoption

Work Year	Empirical strategy	Sample size	Context	Modeling language	Main problems identified
Grossman et al. [17] 2004	Survey	131 UML users	Industry	UML	UML as sketch is the most used style of UML modeling (P1) Lack of adequate UML understanding (P2)
Anda et al. [4] 2006	Case Study	16 System developers and project manager	Industry	UML	Inadequate level of UML training (P3) Inadequate modeling tools (P4)
Dobing and Parson [13] 2006	Survey	171 annalist using UML	Industry	UML	Lack of adequate UML understanding (P2) Inadequate offer of UML training (P3)
Staron [26] 2006	Case Study	8 Professionals	Industry	UML	Lack of well-integrated tools (P4)
Chaudron et al. [8] 2012	1 Experiment, 1 Case Study, 2 Surveys, 20+ Interviews	200+ Professionals and students	Industry Academia	UML	Modeling as sketch and for communication are the most used styles of modeling (P1) Lack of well-integrated tools (P4)
Agner et al. [1] 2013	Survey	209 Embedded-industry developers	Industry	UML	Lack of adequate UML understanding (P2) Lack of specialized professionals (P3) Inappropriate tool support (P4)
Gorschek et al. [16] 2014	Survey	3785 Developers	Industry	All	Models are used primarily in communication and collaboration (P1) Inadequate modeling tools (P4)
Marko et al. [20] 2014	Survey	112 Embedded-industry developers	Industry	All	Lack of well-integrated tools (P4)
Störrle [27] 2017	Survey	96 Professionals	Industry	All	Modeling as sketch is the most used style of modeling (P1) Cultural differences in modeling usage (P6)
Akdur et al. [2] 2018	Survey	627 Embedded-industry developers	Industry	All	Modeling as sketch is the most used style of modeling (P1) Lack of modeling expertise(P3) Inappropriate tool support (P4)

Table 2.1 shows the works on adopting models in the industrial context. Modeling as sketch (also known as informal modeling) is the second problem that has been identified most frequently in the literature [17, 8, 16, 27, 2] (see P1 in Table 2.1). Other studies have identified other problems such as the lack of understanding [17, 13, 1] (see P2) or training [4, 13, 1, 2] (see P3), or the lack of integration between modeling tools [4, 26, 8, 1, 20, 2] (see P4 in Table 2.1).

Through a survey, Grossman et al. [17] investigated if individuals who use UML perceive it to be beneficial and which characteristics affect the use of UML. They used the Task Technology Fit index to evaluate the

respondents' perceptions. They affirmed that most of the respondents of the survey were using what Fowler [15] calls 'UML by sketch', an informal approach to modeling and the values of TTF index indicated a slightly positive perception of UML. Chaudron et al. [8] synthesized a selection of empirical evidence (one experiment, one Case Study, two Surveys, and more than 20 Interviews) about the efficiency of UML modeling in software development. They concluded that especially for larger and distributed projects, UML is used to understand a problem at an abstract level and to share information with other team members. In these cases, UML was used without rigor and specialized tools were not used for the modeling.

Gorschek et al. [16] presented the results of a survey summarizing the answers of 3785 developers to a simple question: Which design models are used before coding? The answer was that design models were not used very extensively in industry and that when they were used, their use was informal and without tool support, and the notation was often not UML. Again, they found that models were used primarily as a communication and collaboration mechanism, where there is a need to solve problems or to obtain a joint understanding of the overall design by a team. Störrle [27] presented the results of an online survey, with 96 industry participants from all over the world. The questions in this case were 'How and what are the models used for?' He found that models were widely used in industry and that UML was indeed the leading language. This directly contradicts the results of Gorschek et al. [16]. He reported three distinct usage modes of models, the most frequent of which was informal usage for communication and understanding, and program-style [24] usage was rare. A year later, in 2018, Akdur et al. [2] conducted another online survey with opinions of 627 practicing embedded software engineers from 27 different countries. The survey addressed the state of software modeling and MDE practices in the worldwide embedded software industry. Their results match those of H. Störrle [27]:the majority of participants were using UML, and its use was informal.

Our work includes a quantitative perspective to discuss the differences in performance of the modeling languages, but we also take into account the modeling style in the analysis to reach a more complete understanding

of modeling usage. Furthermore, the classification of models in previous works [17, 8, 16, 27, 2] is taxonomic; the use that subjects made of models is classified into one style or another in accordance with their responses to certain questions in surveys or interviews. In our experiment, the subjects evaluate their models according to the usefulness of the model for each one of the modeling styles (sketch, blueprint or programs). This offers more complete information on the perception that the subjects have of the usefulness of their models.

2.3 Experiment design

2.3.1 Objectives

According to the guidelines for reporting software engineering experiments [30], we have organized our research objectives using the Goal Question Metric template for goal definition. Our goal is to **analyze** modeling languages and their perceived usefulness, **for the purpose of** comparison, **with respect to** correctness of the models constructed, efficiency, and user satisfaction, **from the point of view of** novice and professional developers, **in the context of** modeling for a video-game company.

2.3.2 Variables

In this study, the factor under investigation is the *Modeling Language*. There are two alternatives, UML and DSL, which are the modeling languages used by subjects to model an enemy of a commercial video game.

Since the goal of this experiment is to evaluate the effects of the use of different modeling languages, we selected *Correctness* and *Efficiency* as the objective response variables, which are related to modeling performance. We measured *Correctness* using a correction template, which was applied to the models developed by the subjects after the experiment. To calculate *Efficiency*, we measured the time employed by each subject to finish the task, using the start and end time of each task. *Efficiency* is the ratio of *Correctness* to time spent (in minutes) to perform a task.

We also compared UML and the DSL with respect to *Satisfaction* using a 5-point Likert-scale questionnaire based on the Technology Acceptance Model (TAM) [21].

We decompose *Satisfaction* into three subjective response variables as follows: *Perceived Ease of Use* (PEOU), the degree to which a person believes that learning and using a particular language would require less effort. *Perceived Usefulness* (PU), the degree to which a person believes that using a particular language will increase performance, and *Intention to Use* (ITU), the degree to which a person intends to use a modeling language. Each of these variables corresponds to specific items in the TAM questionnaire. We average the scores obtained for these items to obtain the value for each variable.

To analyze the subjective perception of the models, the subjects evaluated the usefulness of their models using a 5-point Likert-scale for each style of modeling: to understand and communicate (sketch), for developers to create code (blueprint) and to automatically generate code (programs).

2.3.3 Design

We chose a factorial crossover design with two periods using two different tasks, T1 and T2, one for each period. All of the subjects used the two modeling languages, each one of which was used in a different task. The subjects had been randomly divided into two groups (G1 and G2). In the first period of the experiment, all of the subjects solved T1 with G1 using UML and G2 using DSL. Afterwards, all of the subjects solved T2, G1 using DSL and G2 using UML.

This repeated measures design increases the sensitivity of the experiment [28]: the observation of the same subject using the two alternatives controls between-subject differences, improving experiment robustness regarding variation among subjects. By using two different sequences for each group (G1 used UML first and DSL afterwards, and G2 used DSL first and UML afterwards) and different tasks, the design counterbalances some of the effects caused by using the alternatives of the factor in a specific order (i.e., learning effect, fatigue).

To verify the experiment design, we conducted a pilot study with two subjects. The subjects in the pilot study did not participate in the experiment.

2.3.4 Research questions and hypotheses

The research questions and null hypotheses are formulated as follows:

RQ1 Does the modeling language used for modeling software impact the *Correctness* of the models? The corresponding null hypothesis is $H_{0,C}$: The modeling language does not have an effect on *Correctness*.

RQ2 Does the modeling language used for modeling software impact the *Efficiency* of developers to model? The null hypothesis for *Efficiency* is $H_{0,E}$: The modeling language does not have an effect on *Efficiency*.

RQ3 Is the user satisfaction different when developers use different modeling languages? To answer this question, we formulated three hypotheses based on the variables *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to Use*, with their corresponding null hypotheses. These are: $H_{0,PEOU}$, The modeling language does not have an effect on *Perceived Ease of Use*; $H_{0,PU}$, The modeling langu age does not have an effect on *Perceived Usefulness*; $H_{0,ITU}$, The modeling language does not have an effect on *Intention to Use*.

The hypotheses are formulated as two-tailed hypotheses.

2.3.5 Participants

The subjects were selected using convenience sampling [30]. We invited 26 third-year undergraduate students (novices) from a technology program who had passed previous courses where UML modeling was analyzed and used. Of these subjects, 25 decided to participate and 22 completed the tasks and forms. We also invite 17 professionals who are linked to modeling or video-game development to participate in the experiment. Nine of them decided to participate and completed the experiment. A total of 31 subjects with different knowledge about modeling performed the experiment.

The subjects filled out a demographic questionnaire that was used for characterizing the sample. Table 2.2 shows the mean and standard deviation of age, hours per day developing software (Developing time), and hours per day working with models (Modeling time). We used a 5-point Likert-scale to measure the subjects' knowledge of programming languages (Programming knowledge), modeling languages (Modeling Knowledge) and domain-specific languages (DSL knowledge). The mean and standard deviation of their answers are also shown in Table 2.2. The subjects recognized having a medium-high knowledge about software modeling or specific domain languages. All of them spent more time coding than modeling and evaluated their programming language knowledge or their ability with models higher than their knowledge about domain-specific languages.

Table 2.2: Results of the demographic questionnaire

	Age $\mu \pm \sigma$	**Developing time**$\pm\sigma$	**Modeling time**$\pm\sigma$	**Programming known**$\pm\sigma$	**Modeling known**$\pm\sigma$	**DSL known**$\pm\sigma$
All subjects	23.1±4.5	2.5±2.4	0.9±1.1	3.8±0.9	3.0±1.1	2.6±1.1
Novices	21.1±1.8	1.6±1.7	0.7± 0.8	3.5±0.9	2.5±0.8	2.2±0.9
Professionals	27,9±5.4	4.8±2.2	1.3±1.5	4.4±0.5	4.0±0.8	3.7±0.9

The experiment was conducted by two instructors and one expert in the video-game software domain. The expert provided information about the domain and about the Kromaia DSL. This expert was not the same person who was responsible for designing the tasks. During the experiment, one of the instructors gave the instructions and managed the focus groups. The other instructor clarified doubts about the experiment and took notes during the focus group.

2.3.6 Experimental objects and procedure

The tasks of our experiment were extracted from a real-world software development, Kromaia, which is a commercial video game released on PlayStation 4 and Steam. In Kromaia, the models are interpreted at run-time to create the C++ games' objects [6]. For modeling, the subjects used Shooter Definition Modeling Language, which is the DSL

used in Kromaia[1] and UML. A video-game software engineer, involved in the development of Kromaia, designed the two tasks of similar difficulty and prepared the correction template for the DSL task. An expert on modeling prepared the correction template for the UML task. The experimental objects used in this experiment (which includes the training material, the tasks and the forms used for the questionnaires) as well as the results and the statistical analysis are available at `http://svit.usj.es/UMLvsDSL-experiment`.

The experiment was conducted on-line due to the COVID19 pandemic restrictions. During the experiment, all of the participants joined the same video-conference via Microsoft Teams, and the chat-session was used to clarify doubts or share information. Two forms were prepared on Microsoft Forms for data collection, one for each experimental sequence. The experiment, scheduled for 2 hours, was conducted on two different days with different groups. On the first day, it was performed by novices and on the second day it was performed by professionals. The experimental procedure is described as follows:

1. An instructor explained the parts in the session, and he advised the subjects that it was not a test of their abilities.
2. The subjects attended a tutorial about the video-game enemies to be modeled and about the DSL used in the experiment. The information used in the tutorial and a UML usage guide were available to the subjects during the experiment. The time spent on this tutorial was less than 10 minutes.
3. The subjects received clear instructions on where to find the links to access the forms for the experiment. They were also told about the structure of these forms and where they could find information about UML and the DSL if they needed to. The subjects were randomly divided into two groups (G1 and G2); the subjects from G1 received the links to access one form and the subjects from G2 received a link to another form.

[1]Learn more of Kromaia DSL at: https://youtu.be/Vp3Zt4qXkoY

4. The subjects accessed to the on-line form and read and confirmed having read the information about the experiment, the data treatment of their personal information, and the voluntary nature of their participation before accessing the questionnaires and tasks of the experiment.

5. The subjects completed a demographic questionnaire.

6. The subjects performed the first task. The subjects from G1 had to use DSL to model a video-game enemy, and the subjects from G2 had to use UML to model the same enemy. After submitting their solution, the subjects classified the model they had built according to its usefulness, and they completed a satisfaction questionnaire about the modeling language used.

7. The subjects performed the second task. The subjects from G1 modeled another video-game enemy using UML, and the subjects from G2 modeled the same enemy using the DSL. Then, the subjects classified the model they had built and completed the satisfaction questionnaire.

8. A focus group interview about the tasks (with average duration of 15 minutes) was conducted by one instructor while the other instructor took notes.

9. Finally, the tasks were corrected and a researcher analyzed the results.

2.3.7 Analysis procedure

We have chosen the Linear Mixed Model (LMM) [29] for the statistical data analysis. LMM handles correlated data resulting from repeated measurements, and it allows us to study the effects of factors that intervene in a crossover design (period, sequence, or subject) and effects of other blocking variables (e.g., in our experiment, professional experience) [28].

In this study, we apply the Type III test of fixed effects with unstructured repeated covariance. The *Modeling Language* (ML) was defined as a fixed-repeated factor to identify the differences between using UML or DSL,

and the subjects were defined as random factor $(1|Subject)$ to reflect the repeated measures design. The response variables (RV) for this test were *Correctness* and *Efficiency*, and the three other variables correspond to *Satisfaction*: *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU) and *Intention to use* (ITU).

The assumption for applying LMM is normality of the residuals of the response variables. To verify this normality, we used Kolmogorov-Smirnov tests as well as visual inspections of the histogram and normal Q-Q plots.

The starting statistical model (Model 0) to be tested reflects the principal factors used in this experiment and is described as:

$$RV \sim ML + (1|\,subject) \quad (Model\ 0) \tag{2.1}$$

In order to take into account the potential effects of factors that intervene in a crossover design in determining the main effect of *Modeling Language*, we considered *Period* and *Sequence* to be fixed effects. We also considered fixed factors that are related to the subject's experience in the statistical model in order to explore the potential effects of *Experience* or the effects of the sequence *Modeling Language* and *Experience* $(ML * Experience)$ to determine the variability in the response variables.

We tested different statistical models like the ones used in the following formulas in order to find out which factors, in addition to *Modeling Language*, could best explain the changes in the response variables:

$$\begin{array}{ll} RV \sim ML + Experience + (1|\,Subject) & (Model\ 1) \\ RV \sim ML + Experience + ML * Experience + (1|\,Subject) & (Model\ 2) \\ RV \sim ML + Experience + Period + (1|\,Subject) & (Model\ 3) \end{array} \tag{2.2}$$

The statistical model fit of the tested models was evaluated based on goodness of fit measures such as Akaike's information criterion (AIC) and Schwarz's Bayesian Information Criterion (BIC). The model with the smallest AIC or BIC is considered to be the best fitting model [18]. To describes the changes in each response variable we selected the statistical model that satisfied the normality of residuals and also obtained the smallest AIC or BIC value.

To quantify the differences in the response variables due to significant fixed factors, we calculated the Cohen d value [9] between the alternatives

of these factors. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around of 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. We selected histograms and box plots to graphically describe the data and the results.

2.4 Results

2.4.1 Changes in the response variables

There were differences in the means of all of the response variables depending on which *Modeling Language* was used to model a video-game enemy. Table 2.3 shows the values for the mean and standard deviation of the dependent variables *Correction*, *Efficiency*, *Perceived Ease of Use* (PEOU), *Perceived Usefulness* (PU), and *Intention to use* (ITU) for each one of the *Modeling Languages* compared: UML and DSL.

Table 2.3: Values for the mean and standard deviation of the response variables

	Correctness	*Effectiveness*	*Satisfaction* $\mu \pm \sigma$		
	μ % $\pm\sigma$	μ %/*min* $\pm\sigma$	*PEOU*	*PU*	*ITU*
DSL	90.32 % ± 12.36	6.16 %/*min* ± 2.91	4.27 ± 0.61	4.11 ± 0.63	3.9 ± 0.99
UML	71.87 % ± 25.16	4.10 %/*min* ± 3.51	2.85 ± 0.95	2.83 ± 1.02	2.45 ± 1.23

According to the Cohen d values of the response variables, we can affirm that the effect of the *Modeling Language* on *Correctness* is large, with a Cohen d value of 0.930 and that the effect on *Efficiency* is medium-large, with a Cohen d value of 0.639. The effect size for *Satisfaction* is very large, with Cohen d values of 1.786, 1.515, and 1.327 for *Perceived Ease of Use*, *Perceived Usefulness*, and *Intention to use*, respectively. These values are related to the percentage of non-overlap between the distributions of the response variables for each modeling language. Higher values correspond with greater percentages of non-overlap and larger differences. The histograms of Figure 2.2 illustrate the differences in the response variables. In the histograms, the non-overlapping parts have a single pattern (either dots or shaded), while the overlapping parts have both patterns (dots and shaded).

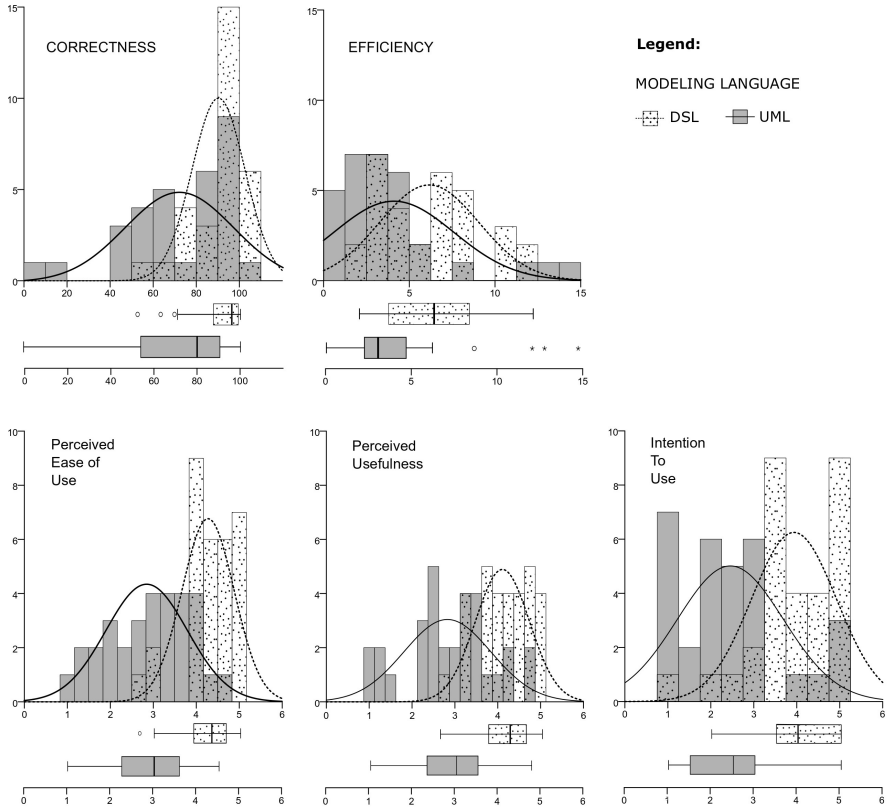

Figure 2.2: Histograms with normal distributions and box plots for the response variables

For all of the variables, the factor *Modeling Language* obtained p-values of less than 0.05, regardless of the statistical model used for its calculation. Therefore, all of the null hypotheses are rejected. Thus, the answers to the research questions **RQ1**, **RQ2**, and **RQ3** are affirmative. The Modeling Language used for developing software has a significant impact on the correctness of the model, efficiency, and the satisfaction of software developers.

For *Correctness*, the chosen statistical model was Model 1 (See formula (2)), which was the best fitting model that satisfied normality in the residuals of the response variable. The fixed factors, *Modeling Language* (F=15.156, p=0.001) and *Experience* (F=4.393, p=0.045), were conside-

red to be statistically significant to explain the changes in correctness. The Cohen d value of -0.654 calculated with the standardized difference between the means of *Correctness* for novices and professionals indicates a medium-large effect in favor of the professionals. The models made by professionals are more correct than the ones made by novices.

For *Efficiency* the chosen statistical model was Model 3 (See formula (2)). The fixed factors *Modeling Language* (F=12.337, p=0.001), *Experience* (F=5.275, p=0.029), and *Period* (F=16.451, p=0.000) were considered to be statistically significant to explain the changes in correctness. The Cohen d value of -0.592 for *Efficiency* between novices and professionals indicates a medium-large effect in favor of the professionals. The professionals are more efficient than novices when modeling. The Cohen d value of -0.788 for *Efficiency* between the first period and the second period, indicates a large effect in favor of the second period. The subjects were more efficient when modeling the second task.

For *Satisfaction* the *Modeling Language* factor was the one fixed factor that was found to be significant for all the statistical models tested. For *Perceived Ease of Use* and *Perceived Usefulness*, the chosen model was the starting statistical model, Model 0 (See formula (1)) and the *Modeling Language* factor obtained p-values of 0.000 for *Perceived Ease of Use* (F=963.137, p=0.000) and *Perceived Usefulness* (F=892.660, p=0.000). For *Intention to use*, the chosen model was Model 2 (See formula (2)); however, only the *Modeling Language* factor (F=13.846, p=0.001) was statistically significant in explaining the changes in this response variable. The changes in *Intention to use* due to *Experience* (F=0.604, p=0.443) or the sequence *Modeling Language* and *Experience* ($ML * Experience$) (F=0.115, p=0.737) were not considered to be statistically significant.

2.4.2 Model assessment by subjects

All of the subjects evaluated their DSL models better than their UML models. The novices considered their models to be more useful as sketch than as blueprint or as programs. The professionals evaluated the usefulness of their DSL model better as programs than as sketch or blueprint, and they evaluated their UML model to be more useful as sketch than as

blueprint or as programs. Table 2.4 shows the values for the mean and standard deviation of the subjects' evaluation of their own models for each modeling style.

Table 2.4: Values for the mean and standard deviation for models usefulness

		As sketch $\mu \pm \sigma$	*As blueprint* $\mu \pm \sigma$	*As a program* $\mu \pm \sigma$
DSL	Novices	4.1 ± 1.0	3.9 ± 1.1	3.5 ± 1.4
	Professionals	4.2 ± 0.8	3.7 ± 0.8	4.3 ± 0.5
UML	Novices	3.0 ± 1.0	3.3 ± 0.8	2.6 ± 1.1
	Professionals	3.8 ± 1.0	3.0 ± 1.1	2.9 ± 1.2

During the focus group, the novices declared that they found the DSL to be useful for communication, but they believed that UML would be necessary to generate a video-game enemy with all of its characteristics. The professionals (especially those linked to video games) said that UML could be useful to define a video-game enemy in general, at the beginning of the development, but that they would choose a DSL to define a specific video-game enemy.

2.4.3 Interpretation of the results

During the training of the experiment, the subjects were introduced to Kromaia, and the use of models made in Kromaia was explained to them. In Kromaia, models are used as programs (see Figure 2.1.c). It is very striking that, despite the fact that a successful case of models as programs was used and that the subjects were explicitly told that Kromaia developers use models as programs, most of the subjects continue to view their models as sketch rather than as programs.

Furthermore, the results also show that models are not far from scoring a value of 100 % correct. On average, only 28 % would have to be corrected in UML models and 10 % in DSL models to be used as programs. Actually, more than five subjects produced models with a value of 100 % correct. These models can be run on top of the Kromaia model interpreter. Nevertheless, they did not give a higher score to programs than to sketch. This is a problem that was not covered in evaluations by other authors: no matter how correct the models are, the subjects value them

more as sketches than as programs. We suggest that this is a major problem: it is not that the models are not ready to be used as programs because we must improve tools and teaching, it is that subjects think of models more as sketches than as programs.

2.5 Threats To Validity

To describe the threats to validity of our work, we use the classification of [30]:

Conclusion validity. The *low statistical power* threat was minimized because the confidence interval is 95 %. The *fishing and the error rate* threat was minimized using tasks and fixes designed by a video-game software engineer. The *Reliability of measures* threat was mitigated because the objective measurements were obtained from the digital artifacts generated by the subjects when they performed the tasks. The *reliability of treatment implementation* threat was alleviated because the procedure was identical in the two sessions. Also, the tasks were designed with similar difficulty.

Internal validity. The *compensatory rivalry* threat affected the experiment; the subjects may have been motivated to perform the task with a higher level of quality by using the modeling language that was the most familiar to them. The *interactions with selection* threat affected the experiment because of the voluntary nature of participation. To avoid student demotivation, we selected students of a course whose contents fit the design of the experiment. In addition, the subjects had different levels of modeling language knowledge and different levels of knowledge of the video-game domain. To mitigate this threat, the treatment was applied randomly. However, we found two outliers during the analysis of correctness and efficiency. The extreme values that were found correspond to subjects with scores of less than 25 % in correctness in the UML task. Following the recommendations of Dean et al. [12], we repeated the statistical analysis by excluding the data of these two subjects, and we found that the language factor remained statistically significant to explain the changes in all of the response variables for all of the models tested. Hence, our conclusions were not sensitive to the responses of subjects with low

scores in correctness. Even though the tasks were designed with similar complexity, the effect of the task (period) was significant for efficiency. The effect of the language being the same for both tasks suggests a learning effect; the subjects spent less time performing the second task. We minimized this *maturation* threat by using a crossover design. **Construct validity.** All of the measurements were affected by *Mono-method bias*. To mitigate this threat for the correctness and efficiency measurements, we mechanized these measurements as much as possible by means of correction templates. We mitigated the threat to satisfaction by using a widely applied model (TAM) [10]. The *hypothesis guessing* threat was mitigated because we did not explain the research questions to the subjects. To weaken the *evaluation apprehension* threat, at the beginning of the experiment, the instructor told the subjects that the experiment was not a test of their abilities. To mitigate the *Author bias* threat, the tasks were extracted from a commercial video game and designed with similar difficulty. Finally, the experiment was affected by the *mono-operation bias* threat because we worked with a single treatment.

External validity. The *domain* threat occurs because the experiment has been conducted in a specific domain, i.e., video-game development. We think that the generalizability of the findings should be undertaken with caution. Other experiments in different domains should be performed to validate our findings.

2.6 Conclusion

In this work, we present an experiment that compares UML and a DSL when subjects model in the video-game domain. To that extent, we combined objective measures and subjective measures. Our results reveal that the subjects underestimate the potential of their own models. This problem was not discovered by previous works that focus either on objective measures to evaluate modeling performance or on subjective measures to classify modeling styles. Our findings suggest that future evaluations should take into account both objective and subjective measures to better understand modeling languages. Our results are also relevant for teaching

modeling and model adoption. If users were able to assess their models correctly, they might leverage their latent potential as programs.

Bibliografía

[1] Luciane Telinski Wiedermann Agner y col. «A Brazilian survey on UML and model-driven practices for embedded software development». En: *Journal of Systems and Software* 86.4 (2013), págs. 997-1005 (vid. págs. 58, 59, 61).

[2] Deniz Akdur, Vahid Garousi y Onur Demirörs. «A survey on modeling and model-driven engineering practices in the embedded software industry». En: *Journal of Systems Architecture* 91 (2018), págs. 62-82 (vid. págs. 58, 61-63).

[3] Apostolos Ampatzoglou y Ioannis Stamelos. «Software engineering research for computer games: A systematic review». En: *Information and Software Technology* 52.9 (2010), págs. 888-901 (vid. pág. 59).

[4] Bente Anda y col. «Experiences from introducing UML-based development in a large safety-critical project». En: *Empirical Software Engineering* 11.4 (2006), págs. 555-581 (vid. págs. 58, 59, 61).

[5] Maicon Bernardino, Elder M Rodrigues y Avelino F Zorzo. «Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches». En: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, págs. 1660-1665 (vid. págs. 58-60).

[6] Daniel Blasco y col. «An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering». En: *Journal of Systems and Software* 171 (2021), pág. 110804 (vid. pág. 66).

[7] David Budgen y col. «Empirical evidence about the UML: a systematic literature review». En: *Software: Practice and Experience* 41.4 (2011), págs. 363-392 (vid. págs. 59, 60).

[8] Michel RV Chaudron, Werner Heijstek y Ariadi Nugroho. «How effective is UML modeling?» En: *Software & Systems Modeling* 11.4 (2012), págs. 571-580 (vid. págs. 58, 59, 61-63).

[9] Jacob Cohen. «Statistical power for the social sciences». En: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (vid. pág. 69).

[10] Fred D. Davis. «Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology». En: *MIS Q.* 13.3 (sep. de 1989), págs. 319-340. ISSN: 0276-7783 (vid. pág. 75).

[11] Andrea De Lucia y col. «An experimental comparison of ER and UML class diagrams for data modelling». En: *Empirical Software Engineering* 15.5 (2010), págs. 455-492 (vid. págs. 58-60).

[12] Angela Dean, Daniel Voss, Danel Draguljić y col. *Design and analysis of experiments*. Vol. 1. Springer, 1999 (vid. pág. 74).

[13] Brian Dobing y Jeffrey Parsons. «How UML is used». En: *Communications of the ACM* 49.5 (2006), págs. 109-113 (vid. págs. 58, 59, 61).

[14] África Domingo y col. «Evaluating the benefits of model-driven development». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2020, págs. 353-367 (vid. págs. 58, 59).

[15] Martin Fowler. *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004 (vid. pág. 62).

[16] Tony Gorschek, Ewan Tempero y Lefteris Angelis. «On the use of software design models in software development practice: An empirical investigation». En: *Journal of Systems and Software* 95 (2014), págs. 176-193 (vid. págs. 58, 61-63).

[17] Martin Grossman, Jay E Aronson y Richard V McCarthy. «Does UML make the grade? Insights from the software development community». En: *Information and Software Technology* 47.6 (2005), págs. 383-397 (vid. págs. 58, 59, 61, 63).

[18] Evrim Itir Karac, Burak Turhan y Natalia Juristo. «A Controlled Experiment with Novice Developers on the Impact of Task Description Granularity on Software Quality in Test-Driven Development». En: *IEEE Transactions on Software Engineering* (2019). ISSN: 0098-5589 (vid. pág. 69).

[19] Tomaž Kosar y col. «Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments». En: *Empirical Software Engineering* 23.5 (2018), págs. 2734-2763 (vid. págs. 59, 60).

[20] Nadja Christiane Marko y col. «Model-based engineering for embedded systems in practice». En: *Research reports in software engineering and management* (2014), págs. 1-48 (vid. págs. 58, 61).

[21] Daniel L Moody. «The method evaluation model: a theoretical model for validating information systems design methods». En: *ECIS 2003 proceedings* (2003), pág. 79 (vid. pág. 64).

[22] Gunter Mussbacher y col. «The relevance of model-driven engineering thirty years from now». En: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2014, págs. 183-200 (vid. págs. 59, 60).

[23] Mari Carmen Otero y José Javier Dolado. «Evaluation of the comprehension of the dynamic modeling in UML». En: *Information and Software Technology* 46.1 (2004), págs. 35-53 (vid. págs. 58-60).

[24] Oscar Pastor y Juan Carlos Molina. *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Springer Science & Business Media, 2007 (vid. pág. 62).

[25] Iris Reinhartz-Berger y Dov Dori. «OPM vs. UML–Experimenting with Comprehension and Construction of Web Application Models». En: *Empirical Software Engineering* 10.1 (2005), págs. 57-80 (vid. págs. 58-60).

[26] Miroslaw Staron. «Adopting model driven software development in industry–a case study at two companies». En: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2006, págs. 57-72 (vid. págs. 58, 59, 61).

[27] Harald Störrle. «How are Conceptual Models used in Industrial Software Development? A Descriptive Survey». En: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017, págs. 160-169 (vid. págs. 58, 61-63).

[28] Sira Vegas, Cecilia Apa y Natalia Juristo. «Crossover designs in software engineering experiments: Benefits and perils». En: *IEEE Transactions on Software Engineering* 42.2 (2015), págs. 120-135 (vid. págs. 64, 68).

[29] Brady T West, Kathleen B Welch y Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman y Hall/CRC, 2014 (vid. pág. 68).

[30] Claes Wohlin y col. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (vid. págs. 63, 65, 74).

[31] Andreas Zendler y col. «Experimental comparison of coarse-grained concepts in UML, OML, and TOS». En: *Journal of systems and Software* 57.1 (2001), págs. 21-30 (vid. págs. 58-60).

Capítulo 3

Evaluating the Influence of the Scope on Feature Location

Context: *Feature Location (FL) is a widespread technique that is used to maintain and evolve a software product. FL is also helpful in reengineering a family of software products into a Software Product Line (SPL). Despite the popularity of FL, there is no study that evaluates the influence of scope (single product or product family) when engineers perform FL.* ***Objective:*** *The goal of this paper is to compare the performance, productivity, and perceived difficulty of manual FL when scope changes from a single product to a product family.* ***Method:*** *We conducted a crossover experiment to compare the performance, productivity, and perceived difficulty of manual FL when scope changes. The experimental objects are extracted from a real-world SPL that uses a Domain-Specific Language to generate the firmware of its products.* ***Results:*** *Performance and productivity decrease significantly when engineers locate features in a product family regardless of their experience. For these variables the impact of the FL Scope is medium-large. On contrast, for perceived difficulty, the magnitude of the difference is moderate and is not significant.* ***Conclusions:*** *While performance and productivity decrease significantly when engineers locate features in a product family, the difficulty they perceive does not predict the significant worsening of the results. Our work also identifies strengths and weaknesses in FL. This can help in developing better FL approaches and test cases for evaluation.*

Versión del autor del artículo: Á. Domingo, J. Echeverría, Ó. Pastor, and C. Cetina, "Evaluating the influence of scope on feature location", Information and Software Technology, vol. 140, p. 106674, Dec. 2021, doi: 10.1016/J.INFSOF.2021.106674.

3.1 Introduction

Feature Location (FL) can arguably be seen as one of the most frequent maintenance tasks undertaken by software engineers [20, 15, 40, 31, 6, 18] since software maintenance and evolution involves adding new features to programs, improving existing functionalities, and removing unwanted functionalities. To this end, it is essential that software engineers find the elements of software features.

FL can also help to reengineer a family of software products into a Software Product Line (SPL) [2, 23] since an SPL involves the formalization of features that are shared by the products. To do this, it is essential for SPL engineers to be able to find the elements of SPL features.

Both classic FL (cFL) and FL in SPL reengineering (rFL) target the elements of features, however, the scope is different, i.e., a single product versus a product family, respectively. One might think that rFL is a particular case of cFL where the first step is to select a product in the family that has the feature and then to perform cFL on that product. Thus, one might conclude that cFL and rFL have similar performance, productivity and difficulty. The engineers of our industrial partner actually believed this (see Section 2 Motivation of the experiment). However, to date, no experiment has been done to compare the scopes of cFL and rFL. This can help engineers to better understand the problems they encounter when locating features in a product family. Knowing these problems is relevant in order to be able to develop new automated or semi-automated FL approaches that could mitigate them.

In this work, we present an experiment that compares different FL scopes (single product and product families), when engineers locate features manually, in terms of performance, productivity, and perceived difficulty.

The experimental objects are extracted from a real-world SPL that uses a Domain-Specific Language (DSL) to generate the firmware of its products. A total of 18 subjects (classified in two groups based on their DSL experience) performed the FL tasks of the experiment.

In the FL tasks, the subjects had to locate elements of different features. We use F-measure to measure performance as the percentage of a FL task solved by a subject. F-measure takes into account both the elements that a subject includes in its solution and are correct and those that it includes and are not correct. We use productivity as the percentage of an FL task solved by a subject per minute. The subjects rate the perceived difficulty of each task using a 7-point Likert scale, ranging from *too easy* to *too difficult*.

Our results shows how performance and productivity decrease significantly when engineers locate features in a product family regardless of their DSL experience. When the FL is in a product family, it is more difficult to locate the feature elements and the errors are propagated by the models throughout the family.

A more in-depth analysis shows how, for example, features with a larger size get better performance. Dispersion is the feature characteristic that best explains the changes in performance that occur when the scope changes: the stronger the dispersion, the lower the performance and the greater the differences in performance when locating features in a single product or in a product family.

Our results also indicate a paradox: even though performance and productivity are significantly worst when the subjects locates features in a product family, the difficulty they perceive does not predict the significant worsening of the results. The initial inclination of subjects is to think that locating features in a product family only takes more time than locating features in a product. However, it turns out that, in the context of a product family, new challenges arise with regard to feature propagation, the elimination process, and feature dispersion. Apparently, the subjects are not aware of the tricky challenge that FL presents in SPL reengineering.

The paradox introduced in the experiment is important because it could help engineers to understand and better perform reengineering Feature Location (rFL). If engineers realized the real difficulty of rFL, they might leverage this information to apply and develop techniques that would improve the performance and productivity of rFL tasks. Engineers perform classical Feature Location (cFL) tasks frequently (i.e., to remove unwanted functionalities in a feature). When they decide to reengineer a family of software products into an SPL, they need to perform rFL, and, by inertia, they may think that cFL is similar to rFL (i.e., they just have to choose a product, perform cFL, and check other products). However, there might be thousands of products in industrial settings and an engineer cannot comprehensively check all of them. The products they choose to locate the feature in can influence the results. This makes rFL more of a challenge than cFL without the engineers realizing it.

We hope that our work will raise awareness of the challenge posed by FL in SPL reengineering and that what we have learned will help design new approaches that compensate for the weaknesses of the engineers.

The rest of the paper is organized as follows: Section 3.2 describes the context that motivates the experiment. Section 3.3 reviews the related work, and Section 3.4 provides the necessary background in FL in product models and the software product domain. Section 3.5 describes the design of our experiment. Section 3.6 reports the results that we discuss in Section 3.7. Section 3.8 describes the threats to validity. Finally, Section 3.9 concludes the paper.

3.2 Motivation of the experiment

For more than ten years, the engineers of our industrial partner, BSH, have developed software that controls its induction hobs. BSH is one of the largest manufacturers of home appliances in Europe. Its induction division produces induction hobs that are sold under the Bosch and Siemens brands, among others. During those years, BSH has developed a family of software products and regularly carried out tasks of classic Feature Location (cFL) to modify the functionalities of its products. In cFL, an engineer has a description of the feature to be located and the product

in the product family that contains this feature. For example, in such a product, there may be an unwanted functionality that must be located in order to eliminate it and replace it with the correct functionality.

Figure 3.1 illustrates a concrete example of cFL in BSH. To simplify, an induction hob (product) is composed of several hob plates (the place where the pots and pans get hot). These require: inverters, which generate the energy for the hob (triangles); inductors, which convert the energy into heat (circles); power channels, which transfer the energy from one element to other (lines); and power managers, which control the path followed by energy through the channels (rectangles).

In a concrete product (P1 in Figure 3.1), the users had reported that when cooking with a pot on the upper hob plate with medium power, the hob turns itself off after a while. To fix this, the engineers must first locate the relevant elements for this unwanted functionality. The engineers have the description of the unwanted functionality and the product where it is located as the input. To perform cFL, the engineers inspect the properties of the elements that they consider relevant to the description of the feature. Using the information in the description, the engineers identify the inverters and channels on the upper hob plate as being relevant to that unwanted functionality (a_1, a_2, c_1, and c_2 in Figure 3.1). After the cFL task, the engineers make the decision to replace these elements with a single inverter and a higher power channel (Figure 3.1, right).

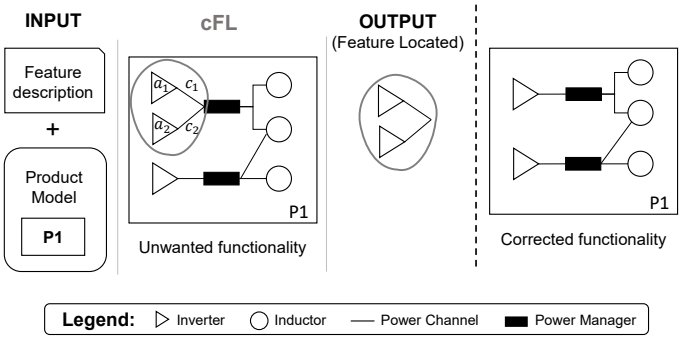

Figure 3.1: Example of classical Feature Location

At some point, the engineers of BSH decided that they want to start a Software Product Line (SPL) from the products that they already have. Software product lines are attractive because the goal is to reduce the development cost and time to market while improving the quality of the software systems by exploiting commonalities and managing the variability across a product family [30]. In an SPL, the common and variable features among the products are formalized to be reused in a systematic way in future developments.

To reengineer a family of software products into an SPL, engineers must identify the commonalities of their products and formalize them as reusable assets. Engineers must perform SPL reengineering Feature Location (rFL) tasks to formalize the features that capture the variability of the systems. In rFl, an engineer has the text description of the feature to be located and a set of products where the feature could be found. For example, in several products of a software family, there is a functionality that must be located in order to formalize it as a feature of the software product line.

When the engineers of our industrial partner (who have performed cFL regularly for years) have to perform rFL, their procedure is as follows: 1) they select a product in the family where they think the feature is located; 2) they perform cFL on that product; and 3) they search for that feature in the other products. The inertia of having performed cFL for so many years has influenced the way the engineers of our industrial partner perform rFL. For this reason, they argue that performing cFL and rFL have similar performance, productivity, and difficulty. After all, for them, the rFL task is essentially a cFL task where you must choose the product on which to perform cFL.

Figure 3.2 illustrates a concrete example in BSH where the engineers perform rFL in order to locate the feature of Double Hot Plate. The description of the Double Hot Plate feature is as follows: some induction hobs have a plate with an extra cooking space (called Double Hot Plate feature) that allows users to select a small or a large hob plate based on their needs. When a user selects the larger plate, two inductors are used, and more energy is needed to heat the plate without reducing the

temperature that the plate must reach or the time in which it must heat up.

To locate the Double Hot Plate feature, the first step is to select a product model from the family of products. In industrial environments such as BSH that have product ranges that are differentiated by brands and are adapted to different countries, the family has more than a thousand products. This makes a thorough review of all of the products a time-consuming or even impossible task. The output of the rFL may be different depending on the products the engineers select to locate the feature. The following are examples of this:

> **Example A:** The engineers select P2 to locate the feature (the fist step of Example A of Figure 3.2). When they locate the feature (the second step), they might think that the feature is composed of a power manager (d_1 in Example A), two power channels (c_1 and c_2 in Example A), and two inductors (b_1 and b_2 in Example A). In the third step, the engineers search for that feature in other products, and if they select a product that confirms their hypothesis (e.g., product P3 in Figure 3.2), then they conclude that the feature is composed of the five elements they had found in the second step.
>
> **Example B:** The engineers select P1 in the first step (Example B of Figure 3.2). When locating the feature in the second step, they might think that the feature is composed of a power manager (d_1 in Example B), three power channels (c_1, c_2, and c_3 in Example B), and two inductors (b_1 and b_2 in Example B). In the third step, if they select a product that confirms their hypothesis (P4 in Example B), they conclude that the feature has six elements as they had hypothesized in the second step.
>
> **Example C:** The engineers select P1 in the first step. They might think that the feature is composed of six elements as in Example B. In this example, the engineers select other products to confirm their hypothesis (P2 and P3 in Example C, step 3), which could make them change their first selection of elements to conclude that the feature is composed of five elements as in Example A.

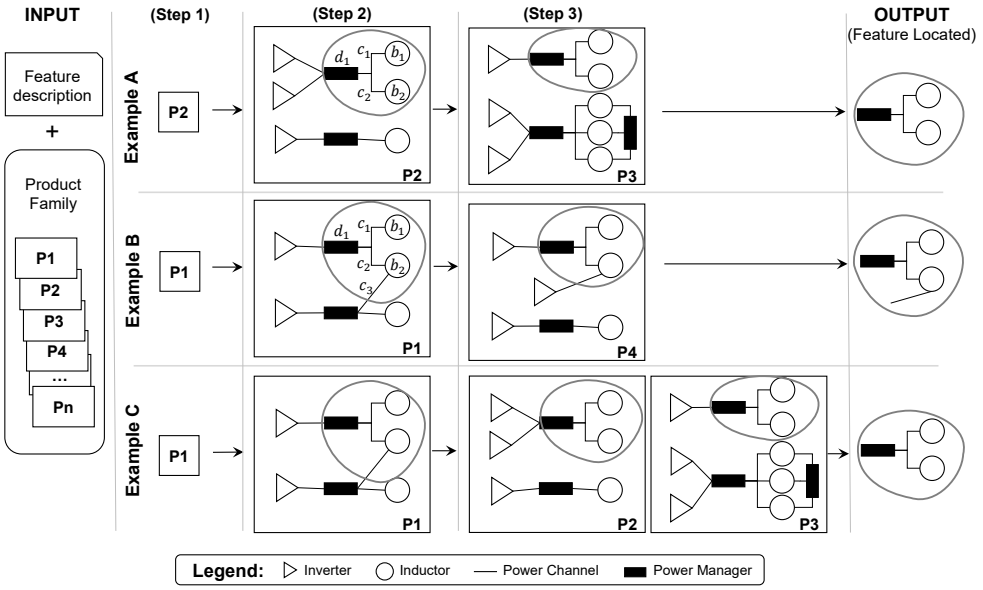

Figure 3.2: Example of reengineering Feature Location

Actually, it is true that rFL and cFL may look similar. In fact, once the product is known, the core of the localization task is the same: the engineers inspect the properties of the model elements to identify those model elements that are the most relevant to the feature description. That is why BSH engineers think that the difficulty of rFL and cFL is the same. However, the selection of products for both starting to locate a feature and for checking the results of the first search affects the performance and the productivity of the rFL tasks with respect to the cFL tasks without the engineers being aware of this. As Table 3.9 shows, in rFL, the engineers can fail in selecting the product, and they can propagate errors in the location of features. Also, the elimination process works worse in rFL than in cFL, and the size and dispersion of the features influence the result of rFL.

This SPL initialization is not only limited to the industrial partner's engineers. According to the results of a survey by Berger et al. [5], starting an SPL from existing products is widespread in industry. Those engineers who have previously maintained and evolved a family of products

(and thus performed cFL) will have to perform rFl to initialize an SPL from this family of products. Thus, they have also been influenced by their experience of performing cFL before rFL. Consequently, they may also think that cFL and rFL have similar performance, productivity, and difficulty.

Our experiment has revealed that the subjects, BSH engineers (experts) and developers without domain-specific knowledge (non-experts), underestimate the rFL task. On the one hand, the subjects state that they do not perceive differences in difficulty between cFL and rFL, which aligns with what the engineers of our industrial partner stated before the experiment. On the other hand, the results for performance and productivity are significantly worse in rFL than in cFL, contrary to what BSH engineers believed. This misperception regarding the rFL task makes the already difficult initialization of an SPL in an industrial environment even more complicated

3.3 Related work

Researchers have conducted several studies to improve the understanding of features and to develop effective techniques to locate features. Some focus on automated techniques to support developers when locating features. Others focus on how developers locate features. The first are referred to as automated FL and the second ones are referred to as manual FL. We also find the term semi-automated FL because features are being located by developers that use different kinds of tools to automatize the process. Not only it is necessary to develop effective FL automated techniques, but it is also important to improve the way of using these techniques by the developers.

A survey of manual FL by Krüger et al. [19] reviews the literature about developers performing manual feature location. They compiled case studies and field studies from 2003 to 2018 that show how manual feature location is performed in industry. Krüger et al. argue that regardless of whether automated or semi-automated techniques are used, manual feature location is relevant. Most of these techniques require a seed to be able to locate the feature and this seed must be located manually.

Table 3.1: Empirical studies on Feature Location

Work Year	Focus	FL Scope	Systems	Variables	Strategy Context	Sample size
Wilde et al. [42] 2003	Manual	Single Product	FORTRAN	Effort to adopt FL Techniques	Case study Industry	4 subjects with different experience
Revelle et al. [33] 2005	Manual	Single Product	C and Java	Concern overlap and Spread	Case study Academia	2 researchers
Wang et al. [39] 2011	Manual	Single Product	Java	Frequencies of physical actions	Experiment Academia	2 developers 18 students
Wang et al. [40] 2013	Manual	Single Product	Java	Frequencies of physical actions Precision, Recall, and F-measure	Experiment Academia	2 developers 54 students
Jordan et al. [16] 2015	Manual	Single Product	COBOL DSLs	Search relevance and success	Case study Industry	2 software engineers
Damevski et al. [11] 2016	Manual	Single Product	Visual Studio	Number of queries, terms in queries, and navigation command events	Field study Industry	667 developers
Krüger et al. [20] 2018	Manual	Single Product	C	LOC, Scattering Degree, and Tangling Degree	Case study Academia	2 researchers
Martinez et al. [23] 2015	Automated	Product Family	UML	Precision, Recall, and Variability safety	Case study Academia	No subjects
Assunção et al. [2] 2020	Automated	Product Family	UML	Number of common model elements	Case study Academia	No subjects
Font et al. [14] 2016	Automated	Product Family	DSL IHs	Precision, Recall, and F-measure	Case study Industry	No subjects
Marcén et al. [22] 2017	Automated	Single Product	DSL	Precision, Recall, and F-measure	Case study Industry	No subjects
Pérez et al. [29] 2018	Automated	Product Family	DSL	Precision, Recall, and F-measure	Case study Industry	No subjects
Font et al. [13] 2017	Automated	Product Family	DSL IHs	Matthew's Correlation Coefficient (MCC) Precision, Recall, and F-measure	Case study Industry	No subjects
Arcega et al. [1] 2015	Automated	Product Family	DSL PervML	Accuracy	Case study Academia	Masters' student as software engineer
Cetina et al. [9] 2017	Automated	Product Family	DSL	Recall, Precision, Area Under the Receiver Operating Characteristics curve, and MCC	Case study Academia	No subjects
Ballarín et al. [3] 2018	Automated	Product Family	DSL	Precision, Recall, F-measure, and MCC (Size, volume, density, multiplicity, and dispersion to report the location problem)	Case study Industry	No subjects
Pérez et al. [27] 2019	Semi-automated	Product Family	DSL	Precision, Recall, and F-measure	Case study Industry	19 Domain Experts
Pérez et al. [28] 2020	Manual, Automated	Single Product	DSL	Performance, Productivity, and Satisfaction	Experiment Industry	18 subjects
This work 2020	Manual	Single Product / Product Family	DSL	Performance (F-measure), Productivity, and Perceived difficulty	Experiment Industry	18 subjects 13 developers 5 Domain Experts

The seven works analyzed in the survey,[42, 33, 39, 40, 16, 11, 20] use a single software product for the feature location tasks and the software products are code developed. Table 1.1 includes some characteristics of these works.

The case study of Wilde et al. [42] analyzes the advantages and disadvantages of three techniques for locating features in C code when using them to locate features in FORTRAN Code. Two of the techniques are automated, and the third one is considered manual (grep text search method). An experienced academic programmer and a graduate student use software reconnaissance to locate a set of features. The experienced programmer, with knowledge about the software system used dependency graph method and the experienced FORTRAN programmer, used the grep text search method to locate the set of features.

The paper of Revelle, Broadbent, and Coppit [33] analyze the identification of the code associated to concerns. A concern is a notion that is more flexible than the notion of feature, but it does include features. They propose guidelines to identify concerns and locate the code associated to them. They compare the percentage of overlap between the sets of code that each subject associates to the same concern. They also adapt Lai and Murphy's spread metric [21] to take into account the number of different files that contain code from the same concern. They conclude that the percentage of overlapping between the sets of code associated to a concern given by different researchers decreases when the number of files containing the code increases, when the spread increases.

The two papers of Wang et al. [39, 40] collect the results of three FL controlled experiments. Two full-time developers and 54 students participate in their experiments. In their first experiment they establish a heuristic to describe phases, patterns, and actions in a feature location task that they present in their first paper. In this second and third experiments, they refine their heuristic and analyze factors such as specific knowledge about feature location, task properties, or experience, which may influence the performance of developers when locating features. Their statistical analysis shows how specific knowledge about feature location phases, patterns and actions improve the developers' performance locating features.

The case study of Jordan et al. [16] has an industrial context. They analyze the search actions and the tools used by nomads, who are experienced software engineers that work on large software systems. They measure the percentage of relevant search results that are viewed by the subjects and use a search success indicator for the searches. A non-empty set of

results, with manageable size, and containing relevant results is considered a successful search. They affirm that nomads are twice as effective locating features as the non-experts reported in other studies [18, 36].

Damevsky et al. [11] report a field study of how developers perform feature location during their daily activity. They report what type of queries, retrieval strategies, and patterns are used by developers when locating features. They also report the search tools used by developers to locate features and how often developers use them. They use the heuristic defined by Wang et al. [40] and analyze the tracking data of developers when locating features.

The case study on the Marlin 3D Printer of Kruger et al. [20] is focused on identifying and locating optional and mandatory features. They explore and compare the characteristics of these features, describing them through metrics such as Lines of Cod (LOC), Scattering Degree (the number of locations where the feature is implemented), or Tangling Degree (the number of features contained in a feature). The authors also provide a set of feature fact sheets that could be used in empirical studies about feature location.

The survey elaborated by Julia Rubin and Marsha Chechink [34] analyzes 24 automated FL techniques and explores how to adapt them to feature location in family products. They affirm that none of the existing FL techniques consider families of related products explicitly. To adapt the FL techniques to a product family, each one of the products of the family is considered an independent entity. They propose that by considering the relations of the products in a family, product line commonalities and variations can provide additional input to improve the accuracy of FL techniques.

Other works ([23, 2]) propose automated approaches for the feature identification and subsequent generation of SPLs from a set of existing software variants. Martínez et al. [23] use two real-word systems, with seven and three model variants, respectively, to evaluate their approach. They compare the number of model elements of the primitive system with the elements of the blocks identified in the model-based SPL generated. They affirm that the generated SPL can regenerate the previous variants and

generate new valid variants. Assunção et al. [2] presents an automated approach to aid the generation of SPLs from existing UML class diagrams and the list of associated features. They generate a Feature Model, which expresses common and variable characteristics of a product family, and a Product Line Architecture, which allows developers to generate, maintain, and evolve system families. To evaluate their approach, they analyze the results obtained with 10 different applications. In each one of these applications, they use a set of well-defined features and apply a Genetic Algorithm to locate features in the Feature Model generated. Optimal values of precision, recall, and variability safety are obtained when the search results on the Feature Model and the input features are compared. Both studies use UML models as input instead of files of code, but neither of the two studies considers real developers using their approaches.

Feature location in models at the industrial scale is a central topic in previous works from our SVIT research group [14, 22, 29, 13, 1, 9, 3, 27, 28]. Given a feature description as input, these works [14, 22, 29] rank the model fragments that are relevant for the feature and explore different approaches to guide the automated feature localization: clustering (through Formal Concept Analysis) [14], empirical learning (through Learning to Rank) [22], and combinations of Similitude, Understandability, and Timing (through Latent Semantic Indexing, Model Size, and Defect Principle, respectively) [29]. In [13], the fitness function is fixed (Similitude), and five search strategies are evaluated (the Evolutionary Algorithm, Random Search, steepest Hill Climbing, Iterated Local Search with restarts, and a hybrid between the Evolutionary algorithm and Hill Climbing). In [1], models at run-time are proposed to be used for increasing the information for feature location. In [9], the sustainability of long-living software systems is exploited to guide feature location. The study of Ballarín et al. [3] provides measures to describe model fragments such as the ones for feature location in models. Collaborative FL is introduced in [27] when the FL task is complex and significantly exceeds the knowledge of a single software engineer. In [28], manual FL and automated FL are compared. The work shows that the subjects are equally satisfied with the results of the two approaches. Hence, we face the possibility that, if subjects consider the results to be good enough, they may

lack the motivation to iterate on the query or the results through the usage of guided techniques.

The works about manual FL use one single software product to locate features in code. Most of the studies about automated FL also focus their attention on locating features in single software products (mainly in code), and the studies about feature location that take into account system variants or product model families do not take into account the developer who locates the features.

Our work addresses the research gap in manual FL in model products, comparing *Performance, Productivity, and Perceived difficulty* between locating features in a Single Product Model and locating features in a Product Model Family.

3.4 Background

3.4.1 Feature Location

Let us consider a software product that is composed of elements from a specific universe:

$$U = \{e_1, e_2, .., e_i, ..., e_j, ..., e_k, ..., e_n\}$$

These elements could be different depending on which artifacts we analyze (e.g., code, requirements or models). If we analyze code lines, the elements could be each word, or each line, or specific groupings of lines, depending on the granularity of the problem. If we analyze requirements or models, these elements also change.

Assuming the artifact and the granularity are fixed, we can now consider a software feature that is represented by a subset of elements of the universe that implements some functionality, for example:

$$F = \{e_i, e_j, e_k\}$$

Thus, we can understand a software product to be a set of elements of U:

$$P = \{F_1 = \{e_1, e_2\}, ..., F = \{e_i, e_j, e_k\}, ...F_N = \{e_i, ..., e_j\}\}$$

In this case, it is not very difficult to find the feature in the software product. The problem begins when the following occur: a software product changes, the features that compose a software product evolve, and the information about them is not as structured. Sometimes the software product is more similar to the one below:

$$P = \{e_1, \{e_2, e_i\}, ..., e_j, e_1, ..., e_k, e_i, e_j, e_k, ..., e_m, ..., e_r\}$$

We do not always have all of the information about the feature and we look for sets like $F = \{e_i, x, e_k, y\}$, where x and y are elements from the specific universe, but they are unknown elements in the feature location. This set is determined from a feature description, which may be more or less precise. Feature Location is the process of finding subsets of elements (not all of which are known), in a collection of elements that represents a software product, given some artifacts and a determined granularity of the problem. Artifacts and granularity specify the universe, U, to which the elements belong. The way the different elements of the universe are composed to build a software product, P, or features, F, depends on them and also on the domain of the problem.

FL can be classified as manual, semi-automated or automated depending on the degree of intervention required by a person to determine and to locate the set of elements that represents a feature. This experiment focuses on manual FL.

3.4.2 Feature Location in Product Models

When models are used to develop software, as in Model Driven Development (MDD) [26], the main development artifact is the product model. In MDD contexts, engineers specify the system to be built with the models and then obtain the source code through model transformations. When models are the main development artifact, FL is performed on product models. This is the case of our industrial partner, which uses a DSL to specify the firmware of its induction hobs. The C++ code that controls its induction hobs is obtained with a transformation from model to code. There are also MDD contexts in which the models are interpreted.

An example of interpreted models is the case of Kromaia, a commercial video game where all of its elements are specified with a DSL that is interpreted at run-time [7]. In these cases, no code is generated from the models, so FL is performed on a product model.

Since a product model is a kind of software product, it can be represented by a collection of elements from a universe set as we represented a software product above. The model elements are the elements of a universe, and a feature is a set of model elements. Locating a feature in a product model means determining the set of model elements that compose the feature and finding these elements in the set of product model elements.

A Product Model Family is composed of several Single Product Models that are similar but different in order to meet certain market needs, i.e., a family of collections composed of model elements from the same universe. The set of elements that represent a feature can be in a single model or in several models. To locate a feature in a Product Model Family, it is necessary to check the product models in the family.

Induction Hobs constitute a Product Model Family for our partner. There are thousands of product models in this family, depending on the brand and the induction hob characteristics (i.e., the number or size of the hob plates, type of hob plates, etc). All of these product models are composed of elements from the same DSL that represent inverters, inductors, power channels, and power managers. The set of elements that represents a feature (i.e., a hob plate with extra coocking space) is in several models but not in all of them.

The Induction Hob universe of elements in BSH would be something like this:

$$U = \{a_1, a_2, .., a_{n_a}, b_1..., b_{n_b}, c_1..., c_{n_c}, d_1..., d_{n_d}\}$$

where a_{i_a}, b_{i_b}, c_{i_c}, and d_{i_d} represent specific inverters, inductors, power channels, and power managers, respectively: n_a being the number of inverters with $1 \leq i_a \leq n_a$; n_b being the number of inductors with $1 \leq i_b \leq n_b$; n_c being the number of power channels with $1 \leq i_c \leq n_c$; and n_d being the number of all of the power managers with $1 \leq i_d \leq n_d$.

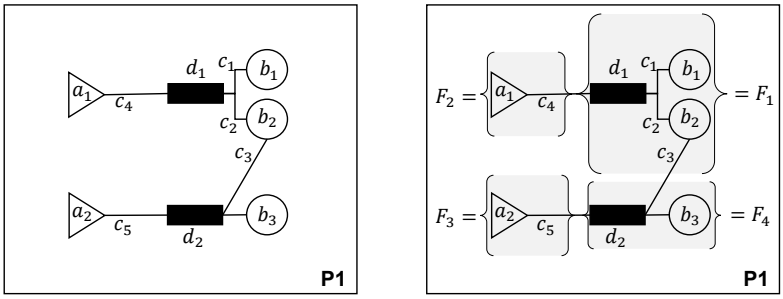

Figure 3.3: Abstract representation of a Product Model in Induction Hobs Family

Figure 3.3 shows the abstract representation of the product model P1 used in Section 2 (See Figures 3.1 and 3.2). The product model P1 could be represented as a a subset of elements of the universe U as follows:

$$\text{P1} = \{a_1, a_2, b_1, b_2, b_3, c_1, c_2, c_3, c_4, c_5, c_6, d_1, d_2\}$$

The Double Hot Plate feature of the rFL example (a hob plate with extra cooking space) could also be represented as a subset of the universe U:

$$F_1 = \{b_1, b_2, c_1, c_2, c_3, d_1\}$$

Therefore, the product model could be represented as a set of features as follows:

$$\text{P1}=\{F_1=\{b_1, b_2, c_1, c_2, c_3, d_1\}, F_2=\{a_1, c_4\}, F_3=\{a_2, c_5\}, F_4=\{b_3, c_6, d_2\}\}$$

where F_2 and F_3 represent different kinds of Simple Inverter features and F_4 represents a Simple Hob Plate feature.

3.4.3 Software Product Model Domain

In this experiment, we have used a subset of the Induction Hob Product Model Family from our industrial partner, BSH, which uses a Software Product Line (SPL) and a Domain-Specific Language (DSL) to generate the firmware of its induction hobs. This DSL is composed of 46 meta-classes, 74 references among them, and more than 180 properties. In order to gain legibility and due to intellectual property rights concerns, a simplified subset of the DSL is used in this experiment. We considered

four kinds of model elements. Each one of them is characterized by three or four properties with different ranges of values depending on their functionality and the requirements of the induction hob. The metamodel and the graphical syntax of the DSL elements are shown in Figure 3.4.

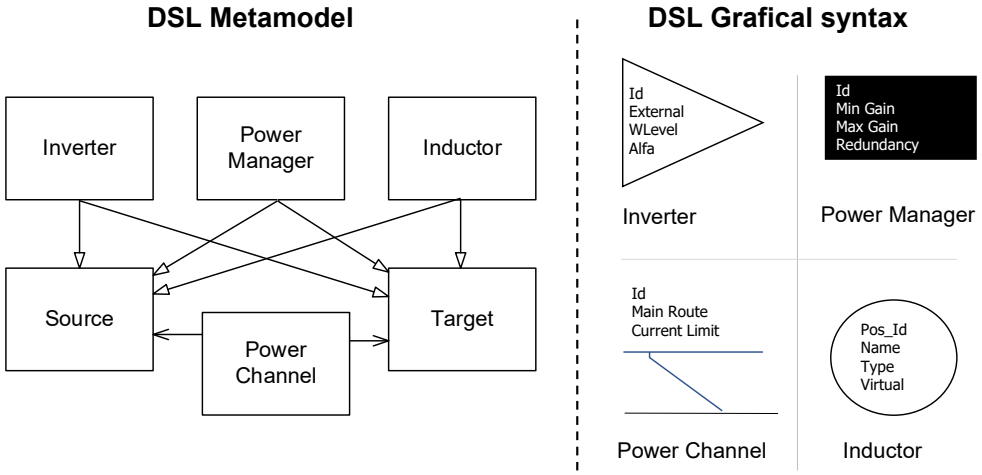

Figure 3.4: Metamodel and Graphical syntax of the DSL

The triangle represents an inverter. The inverter is in charge of converting input electric supply to the specific requirements of the induction hob. The power channels are represented as lines and they are in charge of transferring the energy from the inverter to the inductor. The power manager is represented by a rectangle and controls the path followed by the energy through the channels. The inductor is represented by a circle and is in charge of transforming energy into an electromagnetic field. These elements allow different characteristics included in an induction hob to be expressed. Each induction hob is represented by a set of elements, a product model. The elements and subsets of elements also allow different features included in an induction hob to be expressed.

Similarly to other software artifacts, the granularity can vary depending on the nature of the models and the features being located. Taking into account the MOF standard from the Object Management Group (OMG) to define a universal meta-model for describing modeling languages [24], we divide the relevant elements of a model into a set of atomic elements

(*Class*, *Reference*, and *Property*), and we do not consider further subdivisions of those units in this work. *Class* is the core element. It holds a set of properties and associations (e.g., see the Inductor class element in the metamodel in Figure 3.4 (top-left)). *Reference* relates two class elements. It includes a source class and a target class, a multiplicity for the target class and the source class, and a name. Associations can also be distinguished by whether or not they are containment associations. For instance, the Inductor class reference in the metamodel of Figure 3.4 (top-left) is a containment reference whose source is the Induction Hob class (multiplicity 1) and whose target is the Inductor class (multiplicity any), while the source reference is a reference (non-containment) whose source is the Provider Channel class (multiplicity 1) and whose target is the Inverter class (multiplicity 1). *Property* gives information about a class. It includes the property meta-name, the type, and the value. For instance, the Inverter class element in the metamodel in Figure 3.4 (top-left) contains a property named WLevel whose type is a String. Based on this division, a feature is a subset of the model elements that are present in the model, with the granularity of the elements being classes, references, or properties.

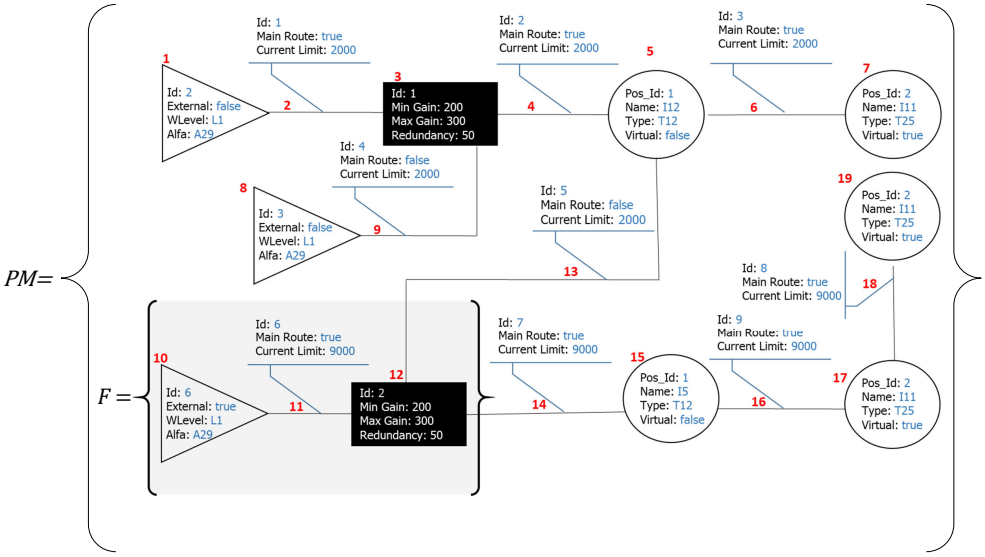

Figure 3.5: Graphical representation of a Product Model and a Feature in the DSL

The features and the product models used in this experiment are collections of model elements with specific properties. Figure 3.5 shows the graphical representation of the realization of the feature (F) described by the text 'High power external supply for induction chain' in a Product Model (PM) from the Product Model Family used in the experiment. Ordered numbers have been assigned to each element in the product models to easily identify each model element during the experiment.

3.5 Experiment design

3.5.1 Objectives

We have organized our research objectives using the Goal Question Metric template for goal definition following the guidelines for reporting software engineering experiments in [43]. Our goal is to:

Analyze Different *FL Scopes* (Single Product Models and Product Model Families) **for the purpose of** comparison, **with respect to** *Performance*, *Productivity*, and *Perceived difficulty*, **from the point of view of** experts and non-experts in the specific domain, **in the context of** manual FL in models in a company.

3.5.2 Research questions and hypotheses

Through this experiment, we seek to answer the following three research questions about manual FL in models:

RQ1 Does the *FL Scope* used for locating features impact *Performance* in manual FL? The corresponding null hypothesis is $H_0 1$: The *FL Scope* does not have an effect on *Performance*.

RQ2 Does the *FL Scope* used for locating features impact *Productivity* in manual FL? The null hypothesis for *Productivity* is $H_0 2$: The *FL Scope* does not have an effect on *Productivity*.

RQ3 Is the *Perceived Difficulty* different when subjects use different *FL Scopes* for locating features? To answer this question we formulated the

null hypothesis $H_0$3: The *FL Scope* does not have an effect on *Perceived Difficulty*.

The hypotheses are formulated as two-tailed hypotheses since we do not know of empirical or theoretical studies that support a specific direction for the effect of scope.

3.5.3 Experimental objects

In this experiment we have used a Product Model Family made up of five product models, that are characterized by the number of model elements and the number of total properties in each one. Table 3.2 shows this information. These Single Product Models are the collections of model elements where the subjects had to locate features during the experiment. Figure 3.5 shows the graphical representation of Product Model 4 (PM4) from the Product Model Family used in the experiment.

Table 3.2: Product Model Description

	Nº elements	**Nº properties**
Product Model 1 (PM_1)	18	64
Product Model 2 (PM_2)	14	49
Product Model 3 (PM_3)	15	53
Product Model 4 (PM_4)	19	67
Product Model 5 (PM_5)	15	53
Product Model Family (PMF)	81	286

The subjects who participated in the experiment had to locate features in a Single Product Model (SPM) and in a Product Model Family (PMF), which represent two different *Feature Location Scopes*. When the subjects located features in a SPM, a feature text description and the number of the model was given. To locate features in the PMF only the feature text description was given to the subjects, and they had to find the model elements of the feature in the collection of elements of five product models.

Table 3.3 shows the characteristics of the features used in the experiment in terms of the size and the percentage of model elements from the feature in the collection of elements of the SPM or the PMF. Figure 3.5 shows the realization of the feature described by the text 'High power external

Table 3.3: Description of experimental objects

		FL Scope			
		SPM		**PMF**	
	$Size\,(F)$	$\frac{Size(F)}{Size(PM_i)}$ %	PM_i	$\frac{Size(F)}{Size(PMF)}$ %	$\|MF\|$
F01	1	6 %	PM_1	1 %	1
F02	2	11 %	PM_1	2 %	1
F03	3	13 %	PM_3	4 %	2
F04	8	53 %	PM_5	10 %	1
F05	2	13 %	PM_5	2 %	1
F06	2	11 %	PM_1	2 %	1
F07	2	7 %	PM_2	2 %	2
F08	3	16 %	PM_4	4 %	1
F09	2	13 %	PM_5	2 %	1
F10	4	21 %	PM_4	5 %	1
F11	2	7 %	PM_2	2 %	2
F12	1	5 %	PM_4	1 %	1

supply for induction chain' (F08 Table 3.3). Since it is composed of three elements, its size is three. The percentage of model elements from the feature increases when the FL focuses on one single product model and also when the number of model elements in the feature increases. The model elements of the features 03, 07, and 11 appear in two different product models when the feature is located in the Product Model Family. The table includes information about the model used when FL is in SPM and the number of product models containing the feature ($|MF|$) when FL is in a PMF.

Each solution output by the subjects is a subset of model elements in the *FL Scope* where the feature is being located. To describe performance, the confusion matrix classifies the solutions output by the subjects [37, 13].

The confusion matrix distinguishes between two values: true or present and false or absent. The predicted values (model elements in the subject's solution) and the model elements of the feature to be located are compared, and the confusion matrix arranges the results of the comparison into four categories. These are: True Positive (TP), predicted values that are in the feature to be located; False Positive (FP), predicted values that are not in the feature to be located; True Negative (TN), model elements

that are neither predicted values nor elements in the feature to be located; and False Negative (FN), elements in the feature to be located that are not predicted values.

Recall, Precision, and F-measure are defined from these predicted values of the confusion matrix. Recall is the ratio between TP and $TP{+}FN$. It represents the percentage of the feature elements that appear in the subject's solution. Precision is the ratio between TP and ($TP{+}FP$). It represents the percentage of the elements in the subject's solution that are in the set that represents the feature. The F-measure is the harmonic mean of Recall and Precision: $F-measure = \left(\frac{(Recall)^{-1}+(Precision)^{-1}}{2}\right)^{-1}$.

The feature realizations of the Family Product Model are known beforehand. For each feature text description given to the subjects, the set of product model elements that are implemented in the feature is known. It is the ground truth (the oracle) that allows us to build the confusion matrix for the subjects' solutions when locating features.

With the following example, we illustrate the potential of Recall, Precision, and F-measure values to describe performance when locating features and how the F-measure could be interpreted as the percentage of problem solved by a subject. Figure 3.5 shows the graphical representation of the realization of the feature (F) described by the text 'High power external supply for induction chain' in a Product Model. Let us assume that the solutions provided by three subjects (A, B, and C) for the feature location of the feature F, were $S_A = \{10, 11\}$, $S_B = \{10, 11, 12, 14, 15\}$, and $S_C = \{9, 3, 11\}$, respectively. The real value of the feature given by the oracle is $F = \{10, 11, 12\}$.

Table 3.4: Example of performance measurements

	TP	FP	FN	Precision	Recall	F-Measure
Oracle	3	0	0	100 %	100 %	100 %
S_A	2	0	1	100 %	67 %	80 %
S_B	2	3	1	40 %	67 %	50 %
S_C	1	2	2	33 %	33 %	33 %

Table 3.4 shows the values of TP, FP, and FN in the confusion matrix and the correspondent values of Recall, Precision, and F-measure for the

oracle and for the solutions S_A, S_B, and S_C. The solution of A (S_A) has a precision of 100 %, but its recall value is 67 % since it does not contain all of the model elements of the oracle. For the solution S_A, the F-measure is 80 %. The recall value for the solution of B (S_B) is also 67 %, but the precision value is only 40 % which is the percentage of elements of this solution that are in the oracle. The F-measure of S_B is only 50 %. The solution S_C obtains values of 33 % for Precision, Recall, and F-measure since the solution only contains one element of the oracle and it contains two elements that are not in the oracle.

3.5.4 Variables

The independent variable in this study is the *FL Scope*. It has two values, Single Product Model (SPM) and Product Model Family (PMF), which are two different kinds of sets of model elements used by subjects to solve the FL tasks.

We consider three dependent variables: *Performance*, *Productivity*, and *Perceived Difficulty*.

To measure *Performance*, we built the confusion matrix of each one of the six features that the subjects must locate for each task to calculate its recall, precision, and F- measure. To represent the *Performance* of the subjects in each task, we calculated the arithmetic mean of the F-measures obtained for each one of the six features to be located in the task.

We measured the time used by each subject to finish each task to calculate *Productivity* as the ratio of *Performance* to time spent (in minutes) to perform a task.

At the end of each task, each subject rated the level of difficulty of the task using a 7-point Likert-scale, ranging from *too easy* to *very difficult*. This value was used to measure *Perceived Difficulty*.

3.5.5 Design

In order to improve experiment robustness regarding variation among subjects [42], we chose a repeated measurement using the largest possible sample size. To avoid the order effect, we chose a crossover design, and we grouped the features to be located in two different tasks, T1 and T2. The subjects had to locate six different features in each one of the tasks. All of the subjects located six features in a Single Product Model in one task, and they located another six different features in a Product Model Family in the other task.

The subjects had been randomly divided into two groups (G1 and G2). In the first part of the experiment, all of the subjects solved T1, G1 located features in the Product Model Family, and G2 located the same features in a Single Product Model. Afterwards, all of the subjects solved T2, G1 located features in a single Product Model, and G2 located features in a Product Model Family. Table 3.5 shows a summary of the experiment crossover design.

Table 3.5: Experiment Design

	Task	
Group	*Task 1*	*Task 2*
G1	Product Model Family	Single Product Model
G2	Single Product Model	Product Model Family

The two tasks were designed by the same instructor. Both were similar yet independent enough to avoid the learning effect. To verify the experiment design, we conducted a pilot study with one subject. This pilot study allowed us to detect typographical and semantic mistakes in some expressions, which were corrected for the experiment. The subject in the pilot study did not participate in the experiment described above.

This design avoids the learning problem effect and the variability due to differences in the average response capacity of the subjects.

3.5.6 Participants

The subjects were selected according to convenience sampling [43]. A total of 18 subjects performed the experiment. All of the subjects completed a demographic questionnaire before entering the experiment, which was used to characterize the sample. Five subjects were software developers and researchers in the induction hob domain. They stated spending an average of 4 hours per day developing software, and an average of 4.2 hours per day working with modeling languages. They were the experts in the experiment. There were also 13 Master's students from Universidad San Jorge (Zaragoza, Spain). They were not experts in the induction hob domain. They stated spending an average of 4.9 hours per day developing software, and 1.34 hours per day working with modeling languages. They were the non-experts in the experiment.

The experiment was conducted by two instructors. The instructors are senior software engineers from the company, and they are also responsible for training newcomer engineers after a hiring process. One instructor designed the tasks for the experiment and generated the correction templates. This instructor clarified general doubts during the experiment and took notes during the focus group. The other instructor explained the experiment, provided information, clarified doubts about the DSL used in the experiment, managed the focus groups, and corrected the tasks.

3.5.7 Experiment procedure

The diagram in Figure 3.6 summarizes the experimental procedure, which is described as follows:

1. The subjects received information about the experiment. An instructor explained the parts in the session. He advised that it was not a test of their abilities.
2. The subjects attended a tutorial about FL in software models, and about the DSL used in the experiment. The information used in the tutorial was available to the subjects during the experiment. The average time spent on this tutorial was 15 minutes.

3. The subjects completed a demographic questionnaire. One of the instructors distributed and collected the questionnaires, verifying that all of the fields had been answered and that the subject had signed the voluntary participation form in the experiment.

4. The subjects received clear instructions on where to find the statements for each task, how to submit their work, and how to complete the task sheet at the end of each task.

5. The subjects performed the first task. The subjects were randomly divided into two groups (G1 and G2) to locate features from the first task. The subjects from G1 had to locate six features in the Product Model Family, and the subjects from G2 had to locate the same six features in a Single Product Model.

6. The subjects assessed the difficulty of the task, taking into account where the feature was located, in a Single Product Model or in the Product Model Family.

7. The subjects added comments about the process followed to locate the feature elements of the first task.

8. An instructor checked that each subject had filled in all of the fields on the task sheet.

9. The subjects performed the second task. The subjects from G1 located six features in a Single Product Model, and the subjects from G2 located the same six features in the Product Model Family. Then, the subjects assessed the difficulty of the second task and added comments about the process followed to locate the feature elements of the second task.

10. A focus group interview about the tasks was conducted by one instructor, while the other instructor took notes.

11. Finally, an instructor corrected the tasks, and a researcher analyzed the results.

The experiment was conducted on two different days at Universidad San Jorge (Zaragoza, Spain). On the first day, it was performed by a group

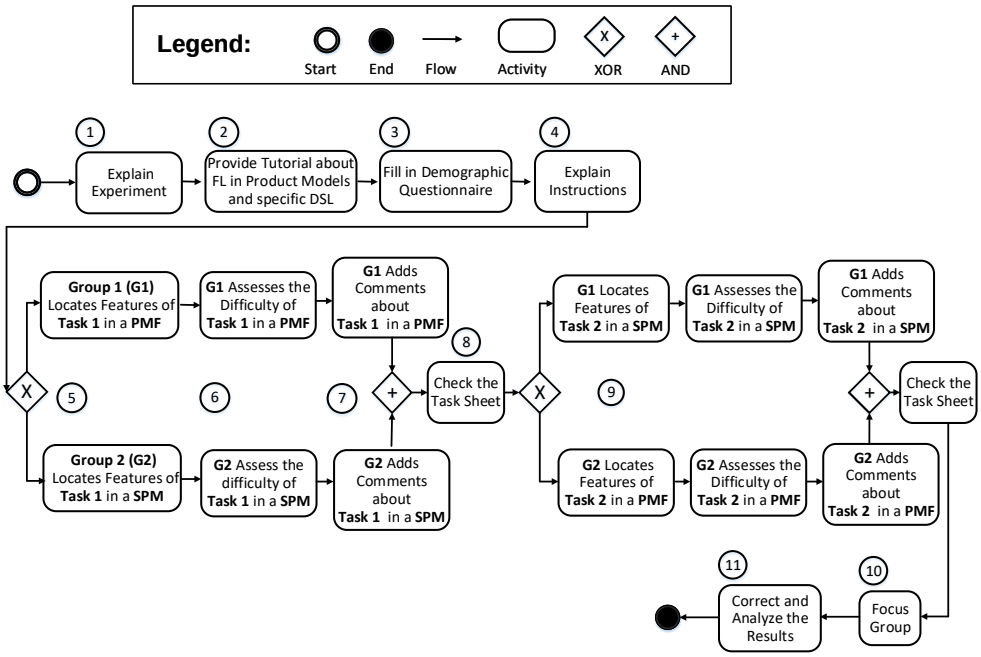

Figure 3.6: Experimental Procedure

of thirteen Master's students (non-experts) in a subject about advanced software modeling. On the second day, the same experiment was performed by the five experts divided into two groups based on their schedule availability (Resp., 2experts, 3 experts).

The instructor responsible for designing the tasks was not the same one who explain the tutorial about FL in software models and the DSL used in the experiment. The materials used in this experiment, which includes the training material, the consent to process the data, the demographic questionnaire, the task sheets, and the results are available at `http://svit.usj.es/ManualFL-experiment`.

3.5.8 Analysis procedure

For the data analysis, we have chosen the Linear Mixed Model (LMM) [41]. LMM handles correlated data resulting from repeated measurements. It is one of the analysis method recommended for crossover experiments in software engineering [38]. The dependent variables for this test are *Performance*, *Productivity*, and *Perceived Difficulty*.

In this study, the subjects who participated are consider random factors and the *FL Scope* is the variable that is repeated to identify the differences between locating features in a Single Product Model or in a Product Model Family. The experimental factors, which are considered fixed effects in our statistical model, are the *FL Scope* (FLS), the *Experience* (E), and the sequence *FL Scope* and *Experience* ($FLS * E$). The subjects are the random effects. The statistical model is expressed in the following formula, where DV represents the dependent variable:

$$DV \sim \quad FLS + E + FLS * E + (1|\, Subject) \tag{3.1}$$

Since the primary focus in our investigation is the *FL Scope*, it is a fixed effect. However, we add *Experience* and the sequence *FL Scope* and *Experience* as be fixed effects for their potential in determining the variability in the dependent variables due to *FL Scope*. There are studies on manual feature location [16, 40] which affirm that human factors such as experience affect feature location patterns and actions, that are improving the effectiveness of the FL. Adding fixed factors related to the subject's experience in the statistical model improves the quality of the model in order to explain the variance of the dependent variables.

To quantify the difference between FL in a Single Product Model and FL in a Product Model Family, we have calculated the effect size using the means and the standard deviations of the values of *Performance*, *Productivity*, and *Perceived Difficulty* for each one of the *FL Scopes* where the subjects locate features. With these values, we calculated Cohen's d Value [10], which is the standardized difference between the two means. Values of Cohen d between 0.2 and 0.3 indicate a small effect, values around 0.5 indicate a medium effect, and values greater than 0.8 indicate a large effect. This value also allows us to measure the percentage of overlap, the percentage between the distributions of the dependent variables

Table 3.6: Values for the mean of the dependent variables

		Performance %	*Productivity* %/*min*	*Perceived Difficulty*
Experts	SPM	65 %	8.3 %/min	2.2
	PMF	47 %	3.6 %/min	3.0
Non-experts	SPM	35 %	3.9 %/min	3.0
	PMF	23 %	2.2 %/min	3.3
All subjects	SPM	43 %	5.1 %/min	2.8
	PMF	29 %	2.6 %/min	3.2

for FL in a Single Product Model and the distributions of the dependent variables for FL in a Product Model Family.

3.6 Results

Table 3.6 shows the values for the mean of the dependent variables *Performance*, *Productivity*, and *Perceived Difficulty* for each one of the *FL Scopes* in which the subjects locate features: Single Product Model and Product Model Family.

There are differences in the means of all the dependent variables depending on which *FL Scope* was used to locate features. These differences are found in both, the experts and the non-experts. The hypothesis testing will allow us to confirm whether or not these differences are significant.

3.6.1 Hypothesis testing

The use of the Linear Mixed Model test assumed that dependent variables residuals must be normally distributed. The normality of these residuals had been verified by the Shapiro-Wilk and Kolmogorov-Smirnov tests, in addition to visual inspections of the histogram and normal Q-Q plots. We obtained normally distributed residuals for all of the dependent variables. All of the residuals obtained a p-value greater than 0.05 with the normality tests.

The results of the Type III test of fixed effects for all of the dependent variables are shown in Table 3.7.

Table 3.7: Results of fixed effects for each variable

	Performance	*Productivity*	*Perceived Difficulty*
FLS	(F=12.0, p=.003)	(F=10.3, p=.003)	(F=3.0, p=.102)
E	(F=15.7, p=.001)	(F=8.2, p=.007)	(F=1.1, p=.307)
$FLS * E$	(F=.5, p=.490)	(F=2.3, p=.143)	(F=.6, p=.452)

For the variables *Performance* and *Productivity*, the factor *FL Scope* obtained p-values of less than 0.05. Therefore, our first two null hypotheses are rejected. Thus, the answers to research questions **RQ1** and **RQ2** are affirmative. The *FL Scope* used for locating features has a significant impact on *Performance* and *Productivity*.

However, since *FL Scope* obtained p-values greater than 0.05 for the variable *Perceived Difficulty*, we can not reject our third null hypothesis. Thus, the answer to research question **RQ3** is negative. The *FL Scope* used for locating features does not have significant impact on the difficulty perceived by the subjects.

The factor *Experience* obtained p-values of less than 0.05 for *Performance* and *Productivity* but greater than 0.05 for *Perceived Difficulty*. This factor also explains the changes in the first two dependent variables, but not in *Perceived Difficulty*.

Moreover, the fixed factor *FL Scope*Experience* obtained p-values greater than 0.05 for all of the dependent variables, which implies that the combination of *FL Scope* and *Experience* had no significant influence on the changes in *Performance*, *Productivity*, or *Perceived Difficulty*. By changing the *FL Scope*, the effects on the dependent variables occur in the same direction and with a similar intensity for both groups of subjects (experts and non-experts).

3.6.2 Effect size

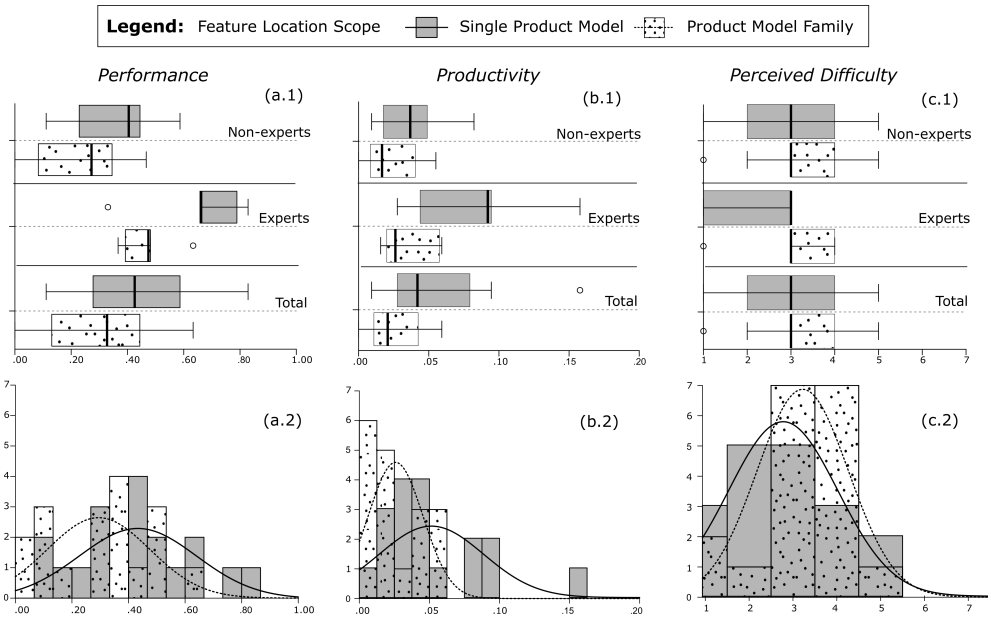

Figure 3.7: Box plots and histograms with normal distribution for *Performance* ((a.1) and (a.2), respectively); for *Productivity* ((b.1) and (b.2)); and for *Perceived Difficulty* ((c.1) and (c.2))

The effect size of the differences based on manual FL in a Single Product Model or in a Product Model Family for *Performance* is medium-large, with a Cohen d value of 0.703. The effect size for *Productivity* is large, with a Cohen d value of 0.847. These values indicate that the percentage of overlap between the distribution of the variables, when the FL is in a Single Product Model and when the FL is in a Family Product, is less than 60 %. Taking into account the area covered by each of the two distributions, the percentage of area shared by both distributions is less than 60 %. The percentage of area covered by one of the distributions, but not the other, is more than 40 % (percentage of non-overlap)

The distribution of *Performance* when the FL is in a Single Product Model with the distribution of *Performance* when the FL is in a Family Product model is more than 43 %. The percentage of non-overlap of *Productivity* distributions for FL in a Single Model and FL in a Family

Product Model is more than 47 %. The box plots and the histograms of Figure 3.7 illustrate the differences in *Performance* ((a.1) and (a.2)) and in *Productivity* ((b.1) and (b.2)). In histograms of Figure 3.7, the non-overlapping parts have a single pattern (either dots or shaded), while the overlapping parts have both patterns (dots and shaded).

The box plots and the histograms of Figure 3.7 for *Perceived Difficulty*,(c.1) and (c.2), respectively, illustrate that the difference when the *FL Scope* changes is not large. A Cohen d value of -0.381 suggested a moderate effect in favor of FL in a Product Model Family. The subjects perceived slightly more difficult FL in a Product Model Family than in a Single Product Model. The percentage of non-overlap of *Perceived Difficulty* distributions for FL in a Single Model and FL in a Family Product Model is less than 29 %. Its histogram has about 70 % overlapping.

This data allows us to give more precise answers to **RQ1**, **RQ2**, and **RQ3**. For *Performance* and *Productivity*, the impact of the *FL Scope* used for locating features is medium-large, and large and the difference is statistically significant. On contrast, for *Perceived Difficulty*, the magnitude of the difference is moderate and is not significant.

3.7 Discussion

To better explain the results, we analyzed the performance values for each feature. For 89 % of the features, 100 % of Precision, Recall, and F-measure was not reached (87 % for SPM, 92 % for PMF). All of the subjects had errors in some of the features they located. Only two subjects (experts) found elements for all of the features to be located in a SPM. No subject located the elements for all of the features when they located features in the PMF.

We analyzed the confusion matrix to measure the subjects' errors in each feature when the 100 % performance was not reached. False Positive was the error that most frequently appeared, and the number of errors increased when FL was in a PMF. When FL was in a PMF, there were 25 % more False Positives. Some subjects acknowledged that after locating feature elements, they reviewed the other product models looking for

these elements. When the subjects select a non-feature element, if that element is shared by other product models in the family, then the error is propagated. We found that 8 % of the subjects' solutions (9 of 108) were affected by fault propagation. This fault propagation was detected in the solutions of four non-experts (30 % of the non-experts). The solutions of the experts were not affected by fault propagation.

PMF also brings another type of error to the table. When the FL was in a PMF, there were 67 % more False Negative elements than when FL was in a SPM. Both the experts and non-experts described a process of elimination to locate features. The elimination process was not taught in the training session. One expert and one non-expert explicitly described an elimination process on their task sheets. Two experts and three non-experts stated that they used something that was unique in the feature description as a filter to discard elements or models during the focus group. As the number of elements of the FL scope increases, the more difficult it is to find the correct feature elements using a process of elimination.

When the subjects were locating features in the PMF, they did not always select the correct product model where the feature elements were located. Some subjects indicated that the feature was not in any product model. On average, the experts selected the product model where the feature elements were located for 60 % of the features to be located. The non-experts selected the correct product model for the 50 % of the features, on average. The only time that all the subjects selected the correct product model was for the feature F04. This is the largest feature in the set of features. This feature also had the best performance: 85 % when the feature is being located in a SPM, and 78 % when the feature is locating in a PMF. Performance is also related to the number of feature elements; features with more elements obtained better performance.

We also analyzed the changes in performance based on other characteristics of the features to be located. We used the five measurements (Size, volume, density, multiplicity, and dispersion) proposed by Ballarín et al. [3]. We found that dispersion was the measurement that best explains the difference in performance when the FL scope changes, based on the characteristics of the location problem. The dispersion measures the ratio

of connected elements in the feature. If a feature is composed by three elements but only two of them are connected it is considered that the feature has two groups of elements. Dispersion is computed as the ratio between the number of groups and the number of elements of a feature [3]. For example, if a feature is composed of three elements in two groups, the dispersion is 2/3.

Table 3.8: Performance Measures by Dispersion

Dispersion Range	Precision	Recall	F-measure	FL Scope
0-0.3	65 %	70 %	63 %	SPM
	64 %	56 %	57 %	PMF
0.5-0.7	56 %	61 %	55 %	SPM
	34 %	40 %	35 %	PMF
1	12 %	44 %	36 %	SPM
	6 %	18 %	9 %	PMF

We classified the features using dispersion. In our experiment, there are two features with a dispersion values of less than 0.3, six features with dispersion values between 0.5 and 0.7, and four features with a dispersion value of one. Table 3.8 shows the average values of performance for the features of each range. As can be observed, the stronger the dispersion, the lower the performance and the greater the differences in performance between locating features in a SPM and locating features in a PMF.

Table 3.9 summarizes the results, the answers, and the findings related to each research question. The above findings (the largest features, feature dispersion, feature propagation, and the elimination process) can help the research community to develop better approaches in the context of feature location.

- **Largest features**: We recommend that the scientific community conduct experiments to study the influence of feature size on FL tasks in detail. To date, no related work has identified the influence of this factor on the performance of FL tasks. The feature size may have the potential to help determine when to use manual, automated, or semi-automated FL. So far, there are no concrete recommendations on when to use each of these approaches. Engineers may think that it is not worth using automated FL to locate a small

Table 3.9: Summary of Results, Answers, and Findings by Research Questions

RQ1	**Results**	For *Performance*, the factor *FL Scope* obtained a p-value of less than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is rejected. The effect size for *Performance* is medium-large in favor of FL in a SPM, with a Cohen d value of 0.703.
	Answer	The *FL Scope* used for locating features has a significant impact on *Performance*. The *Performance* is worse when FL is in a PMF, with the impact being medium-large.
	Findings	· The number of errors increased when FL was in a PMF. · When FL is in a PMF, error propagation affects 30 % of non-experts but does not affect experts. · When subjects locate features in PMF, they can fail in selecting the product model where the feature is actually located. · The process of elimination as a FL Technique works worse in PMF than in SPM. · Features with a larger size (more elements) get better performance in SMP and in PMF. · Dispersion was the measurement that best explains the difference in performance when the *FL Scope* changes. · The stronger the dispersion, the lower the performance and the greater the differences between locating features in a SPM and locating features in a PMF.
RQ2	**Results**	For *Productivity*, the factor *FL Scope* obtained a p-value of less than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is rejected. The effect size for *Productivity* is large in favor of SPM, with a Cohen d value of 0.847.
	Answer	The *FL Scope* used for locating features has a significant impact on *Productivity*. The *Productivity* is worse when FL is in a PMF, with the impact being large.
	Findings	· The initial inclination of subjects is to think that locating features in a PMF only takes more time than locating features in a SPM. · Features with a larger size (more elements) get better productivity in SMP and in PMF. · Dispersion was the measurement that best explains the difference in productivity when the *FL Scope* changes. · The stronger the dispersion, the lower the productivity and the greater the differences between locating features in a SPM and locating features in a PMF.
RQ3	**Results**	For *Perceived Difficulty*, the factor *FL Scope* obtained a p-value of more than 0.05 in the hypothesis contrast test; therefore, the null hypothesis of equal distributions is not rejected. The effect size of the differences for *Perceived Difficulty* is moderate in favor of FL in a PMF, with a Cohen d value of -0.381.
	Answer	*FL Scope* used for locating features does not have a significant impact on *Perceived Difficulty*.
	Findings	· The subjects (experts and non-experts) underestimate the difficulty of FL in a Product Model Family. · The difficulty that subjects perceive does not predict the significant worsening of the results in performance and productivity. · The subjects are not aware of the tricky challenge that FL presents in SPL reengineering.

feature, so they locate it manually. This idea can backfire, and according to our findings, the size of the feature could help engineers to select the best approaches.

- **Feature propagation**: As Krüger et al. [19] stated, many automated and semi-automated FL techniques require a seed for the feature to be located, and this seed must be located manually. When these seeds are obtained by manual FL, they may contain wrong elements and this error could be propagated, as we have shown in our work.

Acknowledging this weakness is the first step in designing input filters that improve the quality of the seeds.

- **Elimination process**: Automated and semi-automated FL techniques explore different approaches to guide feature location (e.g., latent semantic analysis [13] or empirical learning [22]). These techniques search for the relevant elements of the feature. However, the process of elimination could provide inspiration to complement these techniques. For example, these techniques could add a phase in which the model elements were scored based on their differences with the description of a certain feature. The implementation of this process of elimination in FL techniques is out of the scope of this work; however, we want to emphasize that this explicit process of elimination could be a new direction in research to improve FL techniques.

- **Feature dispersion**: This factor, like the ones above, can also contribute to defining feature profiles to recommend which FL approach to use. Therefore, we recommend that experiments delve into how this factor can influence the performance of FL tasks. This factor may also be relevant in rethinking the way in which the results of feature location techniques are presented to engineers. Most of the FL techniques use feature rankings in order to present the results by relevance. Since the dispersion is related to the performance of the engineers, the rankings should offer the possibility of also presenting the results by dispersion ranges and groups of elements. This more complete information could help engineers decide which result in the feature ranking should be chosen.

Furthermore, understanding the difficulties in manual FL is also helpful in order to not miss relevant test cases when FL is evaluated. Our results identify cases where the performance of the engineers in this respect is worse, and these cases can be used as test-cases to evaluate whether an automated or a semi-automated technique could compensate for this.

With regard to generalization, the design of our experiment not only considers the specific characteristics of the domain used by our industrial partner. In fact, it could be applied to any domain that uses the wides-

pread MOF modeling standard. This is why the tasks and the results are expressed in terms of model elements (classes, references, and properties) and not in terms of domain-dependent concepts (inverter, power channel, power manager, or inductor). For example, the five measurements (size, volume, density, multiplicity, and dispersion) that we have used in the discussion were defined in another work [3] using examples from a completely different domain (train control software) and are applicable to any domain.

3.8 Threats to validity

To describe the threats of validity of our work, we use the classification of [43]:

Conclusion validity. The *low statistical power* was minimized because the confidence interval is 95 %. To minimize the *fishing and the error rate* threat, the tasks and corrections were designed by an instructor, with experience in the DSL used, who did not participate in the correction process. The *Reliability of measures* threat was mitigated because two instructors tested the data coherence of the subjects task sheets at the end of each task. The *reliability of treatment implementation* threat was alleviated because the treatment implementation was identical in the two sessions.

Internal validity. To avoid the *instrumentation* threat, we conducted a pilot study to verify the design and the instrumentation. The *interactions with selection* threat affected the experiment because there were subjects who had different levels of experience. To mitigate this threat, the treatment was applied randomly.

Construct validity. *Mono-method bias* occurs due to the use of a single type of measure [25]. All of the measurements were affected by this threat. To mitigate this threat, an instructor checked that the subjects performed the tasks, and we mechanized these measurements as much as possible by means of correction templates. The *hypothesis guessing* threat appears when the subject thinks about the objective and the results of the experiment. To mitigate this threat, we did not explain the

research questions or the experiment design to the subjects. The *evaluation apprehension* threat appears when the subjects are afraid of being evaluated. To weaken this threat, at the beginning of the experiment, the instructor explained to the subjects that the experiment was not a test about their abilities. *Author bias* occurs when the people involved in the process of creating the experiment artifacts subjectively influence the results. In order to mitigate this threat, the tasks were balanced, i.e., their sizes and difficulty were the same for the two tasks. Finally, the *mono-operation bias* threat occurs when the treatments depend on a single operationalization. The experiment was affected by this threat since we worked with a specific DSL.

External validity. The *interaction of selection and treatment* threat is an effect of having a subject that is not representative of the population that we want to generalize. The experiment was performed by non-experts and experts. The participation of non-experts can be a source of experiment weakness; nevertheless, using students instead of software engineers is not a major issue as long as the research questions are not specifically focused on experts [17, 32]. Our results are similar to those of other studies in which the performance of experts and students is compared when locating features in code: experts perform better than non-experts on FL tasks [39, 40, 16]. In our experiment, we also found that regardless of experience, when the scope changes and the FL tasks are performed on a family of products, the performance and productivity worsens. The performance and productivity of experts and non-experts is equally impaired when FL tasks are performed in a product family. The *interaction of setting and treatment* threat is an effect of not having material that is representative of the industrial context of study. The domain used in the experiment is sufficient to perform FL tasks that are analogous to those commonly performed by the engineers of our industrial partner. The values achieved for performance show that all of the subjects had errors in some of the features they located. Only two subjects (experts) found elements for all of the features to be located and only in a single product model. These facts suggest that the tasks are non-trivial. Furthermore, the domain used in this experiment has already been used in previous works [28][12]. The *domain* threat appears since we only analyzed the induction hob domain. We think that the generaliza-

bility of findings should be undertaken with caution. Other experiments in different domains should be performed to validate our findings.

3.9 Conclusion

In this work, we present an experiment that compares performance, productivity, and perceived difficulty in manual FL when the scope changes from a single product to a product family. The experimental objects are extracted from a real-world SPL that uses a DSL. While the performance and the productivity decrease significantly when engineers locate features in a product family, the perceived difficulty does not have very significant changes.

A more thorough analysis seems to indicate how the feature size or the feature dispersion can explain the changes in performance when the scope changes. These results suggest a new research direction that can increase the understanding of the feature problem in order to take advantage of strengths and compensate for weaknesses when developing or evaluating new approaches to FL.

In this work the subjects located features in isolation, but in feature-based product lines when features are added to a product model, interactions may appear between the features of that product or the features of other products in the family. Interactions can be severely damaging to system development and to user expectations [8]. The analysis of interactions between features could also be used to determine the original feature module definitions [4, 35]. None of the authors referenced in the Related Work section has explicitly analyzed the effects of feature interaction on the performance of FL tasks. With the aim of understanding the feature problem in an SPL, feature interactions must also be considered, which is what we plan to do in future works.

Bibliografía

[1] Lorena Arcega y col. «Leveraging Models at Run-Time to Retrieve Information for Feature Location.» En: *MoDELS@ Run. time.* 2015, págs. 51-60 (vid. págs. 90, 93).

[2] Wesley KG Assunção, Silvia R Vergilio y Roberto E Lopez-Herrejon. «Automatic extraction of product line architecture and feature models from UML class diagram variants». En: *Information and Software Technology* 117 (2020), pág. 106198 (vid. págs. 82, 90, 92, 93).

[3] Manuel Ballarín y col. «Measures to report the location problem of model fragment location». En: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems.* 2018, págs. 189-199 (vid. págs. 90, 93, 114, 115, 118).

[4] Don Batory, Peter Höfner y Jongwook Kim. «Feature interactions, products, and composition». En: *Proceedings of the 10th ACM international conference on Generative programming and component engineering.* 2011, págs. 13-22 (vid. pág. 120).

[5] Thorsten Berger y col. «A survey of variability modeling in industrial practice». En: *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems.* 2013, págs. 1-8 (vid. pág. 88).

[6] Ted J Biggerstaff, Bharat G Mitbander y Dallas Webster. «The concept assignment problem in program understanding». En: *[1993] Proceedings Working Conference on Reverse Engineering.* IEEE. 1993, págs. 27-43 (vid. pág. 82).

[7] Daniel Blasco y col. «An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering». En: *Journal of Systems and Software* 171 (2021), pág. 110804 (vid. pág. 96).

[8] Muffy Calder y col. «Feature interaction: a critical review and considered forecast». En: *Computer Networks* 41.1 (2003), págs. 115-141 (vid. pág. 120).

[9] Carlos Cetina y col. «Improving feature location in long-living model-based product families designed with sustainability goals». En: *Journal of Systems and Software* 134 (2017), págs. 261-278 (vid. págs. 90, 93).

[10] Jacob Cohen. «Statistical power for the social sciences». En: *Hillsdale, NJ: Laurence Erlbaum and Associates* (1988) (vid. pág. 109).

[11] Kostadin Damevski, David Shepherd y Lori Pollock. «A field study of how developers locate features in source code». En: *Empirical Software Engineering* 21.2 (2016), págs. 724-747 (vid. págs. 90, 92).

[12] Jorge Echeverría y col. «An empirical study of performance using Clone & Own and Software Product Lines in an industrial context». En: *Information and Software Technology* 130 (2021), pág. 106444 (vid. pág. 119).

[13] Jaime Font y col. «Achieving feature location in families of models through the use of search-based software engineering». En: *IEEE Transactions on Evolutionary Computation* 22.3 (2017), págs. 363-377 (vid. págs. 90, 93, 102, 117).

[14] Jaime Font y col. «Feature location in models through a genetic algorithm driven by information retrieval techniques». En: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. 2016, págs. 272-282 (vid. págs. 90, 93).

[15] Wenbin Ji y col. «Maintaining feature traceability with embedded annotations». En: *Proceedings of the 19th International Conference on Software Product Line*. 2015, págs. 61-70 (vid. pág. 82).

[16] Howell Jordan y col. «Manually locating features in industrial source code: the search actions of software nomads». En: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, págs. 174-177 (vid. págs. 90, 91, 109, 119).

[17] Barbara A. Kitchenham y col. «Preliminary guidelines for empirical research in software engineering». En: *IEEE Transactions on Software Engineering* 28.8 (ago. de 2002), págs. 721-734. ISSN: 0098-5589 (vid. pág. 119).

[18] Andrew J Ko y col. «An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks». En: *IEEE Transactions on software engineering* 32.12 (2006), págs. 971-987 (vid. págs. 82, 92).

[19] Jacob Krüger, Thorsten Berger y Thomas Leich. «Features and How to Find Them: A Survey of Manual Feature Location». En: *Software Engineering for Variability Intensive Systems* (2019), págs. 153-172 (vid. págs. 89, 116).

[20] Jacob Krüger y col. «Towards a better understanding of software features and their characteristics: a case study of marlin». En: *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. 2018, págs. 105-112 (vid. págs. 82, 90, 92).

[21] Albert Lai y Gail C Murphy. «The structure of features in Java code: An exploratory investigation». En: *Position Paper for Multi-Dimensional Separation of Concerns Workshop, OOPSLA*. 1999 (vid. pág. 91).

[22] Ana C Marcén y col. «Towards feature location in models through a learning to rank approach». En: *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*. 2017, págs. 57-64 (vid. págs. 90, 93, 117).

[23] Jabier Martinez y col. «Automating the extraction of model-based software product lines from model variants (T)». En: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2015, págs. 396-406 (vid. págs. 82, 90, 92).

[24] *O.M.G.: Meta object facility (mof) version 2.4.1*. `http://www.omg.org/spec/MOF/2.4.1/`. Online; accessed 12 December 2020. 2013 (vid. pág. 98).

[25] Jose Ignacio Panach y col. «In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction». En: *Information and Software Technology* (2015). ISSN: 09505849 (vid. pág. 118).

[26] Oscar Pastor y Juan Carlos Molina. *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Springer Science & Business Media, 2007 (vid. pág. 95).

[27] Francisca Pérez y col. «Collaborative feature location in models through automatic query expansion». En: *Automated Software Engineering* 26.1 (2019), págs. 161-202 (vid. págs. 90, 93).

[28] Francisca Pérez y col. «Comparing manual and automated feature location in conceptual models: A Controlled experiment». En: *Information and Software Technology* 125 (2020), pág. 106337 (vid. págs. 90, 93, 119).

[29] Francisca Pérez y col. «Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study». En: *Information and Software Technology* 103 (2018), págs. 188-201 (vid. págs. 90, 93).

[30] Klaus Pohl, Günter Böckle y Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005 (vid. pág. 86).

[31] Denys Poshyvanyk y col. «Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval». En: *IEEE Transactions on Software Engineering* 33.6 (2007), págs. 420-432 (vid. pág. 82).

[32] Iris Reinhartz-Berger y Dov Dori. «OPM vs. UML–Experimenting with Comprehension and Construction of Web Application Models». En: *Empirical Software Engineering* 10.1 (2005), págs. 57-80 (vid. pág. 119).

[33] Meghan Revelle, Tiffany Broadbent y David Coppit. «Understanding concerns in software: insights gained from two case studies». En: *13th International Workshop on Program Comprehension (IWPC'05)*. IEEE. 2005, págs. 23-32 (vid. págs. 90, 91).

[34] Julia Rubin y Marsha Chechik. «A survey of feature location techniques». En: *Domain Engineering*. Springer, 2013, págs. 29-58 (vid. pág. 92).

[35] Norbert Siegmund y col. «Predicting performance via automated feature-interaction detection». En: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, págs. 167-177 (vid. pág. 120).

[36] Jamie Starke, Chris Luce y Jonathan Sillito. «Searching and skimming: An exploratory study». En: *2009 IEEE International Conference on Software Maintenance*. IEEE. 2009, págs. 157-166 (vid. pág. 92).

[37] Stephen V Stehman. «Selecting and interpreting measures of thematic classification accuracy». En: *Remote sensing of Environment* 62.1 (1997), págs. 77-89 (vid. pág. 102).

[38] Sira Vegas, Cecilia Apa y Natalia Juristo. «Crossover designs in software engineering experiments: Benefits and perils». En: *IEEE Transactions on Software Engineering* 42.2 (2015), págs. 120-135 (vid. pág. 109).

[39] Jinshui Wang y col. «An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions». En: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. 2011, págs. 213-222 (vid. págs. 90, 91, 119).

[40] Jinshui Wang y col. «How developers perform feature location tasks: a human-centric and process-oriented exploratory study». En: *Journal of Software: Evolution and Process* 25.11 (2013), págs. 1193-1224 (vid. págs. 82, 90-92, 109, 119).

[41] Brady T West, Kathleen B Welch y Andrzej T Galecki. *Linear mixed models: a practical guide using statistical software*. Chapman y Hall/CRC, 2014 (vid. pág. 109).

[42] Norman Wilde y col. «A comparison of methods for locating features in legacy software». En: *Journal of Systems and Software* 65.2 (2003), págs. 105-114 (vid. págs. 90, 91, 105).

[43] Claes Wohlin y col. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (vid. págs. 100, 106, 118).

Parte III

Discusión de resultados

Discusión de resultados

En esta parte de la tesis se discuten los resultados de este trabajo, conectando las preguntas de investigación planteadas al inicio del trabajo, con los resultados plasmados en los artículos de investigación recogidos en los tres capítulos anteriores.

Para entender las causas por las que MDD no ha tenido el éxito pronosticado por la comunidad, nos planteamos tres preguntas de investigación. Cada una de estas preguntas motiva uno de los experimentos que se recogen en las publicaciones de este compendio. En esta sección, discutimos cómo los resultados obtenidos en los experimentos realizados nos ayudan a responder a las preguntas de investigación de este trabajo, y a comprender las causas por las que MDD no ha reemplazado a otros enfoques en ingeniería del software.

RQ_1: ¿Son los beneficios de MDD frente a otros enfoques como CcD fieles a la realidad del desarrollo software actual?

Para responder a la primera pregunta de investigación, realizamos un experimento en el que consideraremos tareas extraídas de un entorno de desarrollo software real y se utilizan frameworks de dominio, tanto en el contexto MDD, como en el contexto CcD. Hoy en día, los frameworks de dominio se utilizan ampliamente en el desarrollo de software. Estos frameworks ahorran tiempo a los desarrolladores al utilizar implementaciones que han sido desarrolladas previamente. En las comparaciones anteriores entre MDD y CcD, se podría argumentar que los beneficios de MDD podrían provenir de la falta de frameworks de dominio para CcD. Este no fue el caso en nuestro experimento, que se publica con el título *Evaluating the Benefits of Model-Driven Development* [10], y es el primer resultado que se presenta en el compendio de artículos que componen esta tesis. A partir de los resultados y hallazgos de este experimento, afirmamos que los beneficios de MDD respecto CcD siguen vigentes aumentando la fidelidad de la comparación con entornos de desarrollo real.

En este experimento participaron 44 sujetos, clasificados en desarrolladores con experiencia y desarrolladores sin experiencia, que debían desarrollar el enemigo final de un videojuego real (Kromaia). Consideramos como variables dependientes objetivas: la *Corrección* de los desarrollos de los sujetos y la *Eficiencia* de los sujetos al realizarlos. Como variable dependiente subjetiva utilizamos *Satisfacción*, definida a través de la *Facilidad de Uso Percibida*, *Utilidad Percibida* e *Intención de Uso*. En nuestro experimento, cuando los sujetos utilizan MDD mejoran sus resultados de *Corrección* y *Eficiencia*, en un 41 % y un 54 % respectivamente, independientemente de su experiencia. Tanto los desarrolladores con experiencia

como los desarrolladores sin experiencia mejoraron significativamente sus resultados con MDD. El factor experiencia no afecta significativamente a los cambios en las variables dependientes. El tamaño del efecto entre desarrollar utilizando MDD en lugar de CcD es muy grande en las variables *Corrección* y *Eficiencia* en favor de MDD. Sin embargo, el tamaño del efecto en *Satisfacción* es medio, especialmente en *Intención de Uso*, donde el efecto podría considerarse pequeño en comparación con el resto.

Aún no sabemos si la clave de los beneficios de MDD se debe a la abstracción de los modelos, a la automatización de las transformaciones de modelo a código, o en una combinación de ambas. Sin embargo, resulta que un framework de dominio por sí solo no es suficiente para lograr los beneficios de MDD. Nuestros resultados también contradicen las afirmaciones que limitan los beneficios de MDD a desarrolladores sin experiencia. En nuestro experimento la experiencia no afecta significativamente a los cambios *Corrección*, *Eficiencia* o *Satisfacción*. Además, nuestros resultados revelan una paradoja:

> A pesar de las significativas mejoras en *Corrección* y *Eficiencia*, la *Intención de Uso* de MDD no alcanza los valores máximos

Durante el experimento, tanto los sujetos con experiencia, como los sujetos sin experiencia, coincidieron en que la abstracción de los modelos es un arma de doble filo. Por un lado, les permite centrarse en la tarea que tienen entre manos. Por otro lado, los sujetos sienten que pierden el control del código generado. Los sujetos reconocen que esta pérdida de control influye negativamente en su intención de uso. Algunos sujetos afirmaron que esta pérdida de control se aliviaría si la transformación del modelo se considerara como un desarrollador más del equipo. Esto desencadena una nueva e interesante dirección de investigación. Creemos que los experimentos futuros deberían ir más allá de la faceta técnica y, por qué no, explorar las implicaciones sociales y culturales de MDD en los equipos de desarrollo.

RQ_2: ¿Los ingenieros software valoran adecuadamente los modelos que utilizan para el desarrollo de software?

Para responder a la segunda pregunta de investigación, realizamos un experimento en el que los sujetos debían modelar el enemigo final de un videojuego profesional (Kromaia), utilizando dos lenguajes de modelado diferentes: un lenguaje de propósito general (UML) y un lenguaje de dominio específico (DSL). Este experimento se publica con el título *Comparing UML-based and DSL-based Modeling from Subjective and Objective Perspectives* [9], y es el segundo resultado que se presenta en el compendio de artículos que componen esta tesis. A partir de los resultados y hallazgos de este experimento, afirmamos que los profesionales software no valoran correctamente la utilidad de los modelos que ellos mismos elaboran.

En este experimento participaron 31 sujetos, clasificados en desarrolladores con experiencia y desarrolladores sin experiencia, que debían modelar el enemigo final de un videojuego. Consideramos como variables dependientes objetivas la *Corrección* de los modelos y la *Eficiencia* de los sujetos al realizarlos. También utilizamos *Satisfacción*, definida a través de la *Facilidad de Uso Percibida*, *Utilidad Percibida* e *Intención de Uso*. Los resultados de este experimento muestran que la *Corrección* y la *Eficiencia* mejoran significativamente cuando los sujetos modelan utilizando el DSL en lugar de UML, y que el tamaño de las diferencias es grande para ambas variables a favor del DSL. En *Satisfacción* las diferencias también son significativas, y el tamaño efecto es muy grande a favor del modelado con el DSL. En este experimento, la experiencia es un factor que afecta significativamente a *Corrección* y la *Eficiencia*, no a la *Satisfacción*. Los sujetos más experimentados generan modelos más correctos de forma más eficiente.

Los modelos elaborados por los sujetos en el experimento no están lejos de obtener un 100 % de corrección para ser utilizados como programas. Por término medio, sólo habría que corregir un 28 % en los modelos UML y un 10 % en los modelos DSL. Sin embargo, al igual que en otros trabajos empíricos previos [14, 6, 13, 26, 2], los sujetos de nuestro experimento también afirman que el uso principal de sus modelos es como bocetos

para asistir en la comunicación del equipo de desarrollo (como 'sketch'). Nuestros resultados revelan que:

> Los sujetos subestiman el potencial de sus propios modelos, y probablemente, el problema de adopción de MDD no se limita a elegir el lenguaje de modelado que mejor funciona en un determinado dominio.

Durante el entrenamiento del experimento, se introdujo a los sujetos en el DSL utilizado en Kromaia, y se les explicó el uso de los modelos realizados con el. En Kromaia, los modelos se utilizan como programas (como 'programs'). Resulta muy llamativo que, a pesar de que se utilizó un caso exitoso de modelos como 'programs' y de que se explicó explícitamente a los sujetos que Kromaia se desarrolla utilizando MDD, la mayoría de los sujetos seguían viendo sus modelos más como 'sketch' que como 'programs'. Nuestro trabajo revela un problema que no se había encontrado en las evaluaciones de otros autores: por muy correctos que sean los modelos, los sujetos consideran bocetos los modelos que podrían ser utilizados como programas en contextos MDD.

Nuestros resultados sugieren que el problema de adopción de MDD es más profundo que comparar qué lenguaje de modelado funciona mejor en un determinado dominio, como era el objetivo de algunos trabajos anteriores [5, 29, 19, 22, 7]. Aunque un lenguaje tenga un rendimiento significativamente mejor que otro, los sujetos podrían seguir considerando sus modelos como bocetos. No es sólo que los desarrolladores sean inexpertos o que no entiendan los modelos, como se identifica en otros trabajos [14, 8, 1, 4, 2, 25, 6, 18]. Es que, aunque sus modelos no están lejos de la solución, los desarrolladores no los valoran correctamente. Para que los modelos sean más útiles en el desarrollo de software, debemos conseguir que los desarrolladores aprendan a valorar mejor lo que ellos mismos modelan.

RQ_3: ¿Cómo es el desempeño de los profesionales software en tareas de mantenimiento en contextos MDD?

Para dar respuesta a la tercera pregunta de investigación de esta tesis, realizamos un experimento en el que analizamos el desempeño de los

ingenieros en dos tipos de tareas de localización de características en un contexto MDD. Los modelos utilizados en el experimento se extraen de una línea de producto software del entorno industrial, que utiliza un lenguaje específico de dominio (DSL) para generar el firmware de sus productos a partir de modelos. Los sujetos realizan FL clásica (cFL), cuando la localización de la característica se realiza en único modelo de producto, y FL para la reingeniería (rFL), cuando el objetivo es localizar características comunes en una familia de productos. Este experimento se publica con el título *Evaluating the Influence of the Scope on Feature Location* [11], y es el tercer resultado que se presenta en el compendio de artículos que componen esta tesis. A partir de los resultados y hallazgos de este experimento, afirmamos que los profesionales software con experiencia previa en el DSL obtienen mejor desempeño en las tareas de FL que los sujetos sin experiencia. Sin embargo, ambos grupos empeoran de igual forma cuando realizan rFL en lugar de cFL sin que perciban el aumento de la dificultad.

En este experimento participaron 18 sujetos, clasificados en dos grupos en función de su experiencia con el DSL utilizado en las tareas (desarrolladores con experiencia y sin experiencia). Los sujetos debían localizar características en diferentes contextos: en un único producto (cFL) o en una familia de productos (rFL). Consideramos como variables dependientes objetivas *Rendimiento* y *Productividad* de los sujetos al realizar tareas de localización. Como variable subjetiva utilizamos la *Dificultad Percibida* por los sujetos en las tareas de localización. Los resultados de este experimento muestran cómo, independientemente de la experiencia de los sujetos, el *Rendimiento* y la *Productividad* disminuyen significativamente cuando localizan características en una familia de productos (rFL). Los ingenieros con experiencia con el DSL realizaron las tareas de feature location significativamente mejor que los sujetos sin experiencia, pero tanto unos como otros, redujeron en un 40 % su rendimiento y a la mitad su productividad cuando en lugar de localizar características en un producto las tenían que localizar en una familia de productos. En cambio, la *Dificultad Percibida* no cambia significativamente entre las tareas cFL y las tareas rFL en ninguno de los grupos de sujetos.

Aparentemente, los sujetos no son conscientes del reto que supone la FL para reingeniería y subestiman la dificultad de hacer mantenimiento en contextos MDD.

En este trabajo [11], también se presenta un análisis más profundo de los resultados, que permite identificar factores que afectan a la correcta localización de características. Se muestra cómo el tamaño de las características a localizar, o su dispersión en la familia de productos, explican algunos de los cambios en el rendimiento cuando se realiza localización de características para reingeniería (rFL). Cuanto mayor es el tamaño y menor es la dispersión de las características, mejores son el rendimiento y la productividad en las tareas de localización. Además, se muestra como la rFL supone nuevos retos como la propagación de errores de localización en los diferentes productos de una familia. También identificamos el uso de un proceso de eliminación para las tareas de localización, lo que puede inspirar nuevas técnicas automáticas, semiautomáticas o manuales. Conocer las razones que dificultan o facilitan las tareas de localización de características, puede ayudar a los ingenieros a aplicar y desarrollar técnicas que mejoren su desempeño en tareas de mantenimiento en contextos MDD.

Cumplimiento del objetivo

El objetivo fundamental de esta tesis consiste en estudiar empíricamente las causas por las que MDD no ha tenido el éxito pronosticado por la comunidad. En esta sección se reflexiona sobre el cumplimiento de este objetivo a partir de los aprendizajes extraídos durante el desarrollo de la investigación. La Tabla 2 recoge un resumen de las respuestas a las preguntas de investigación de esta tesis, relacionándolas con los resultados y hallazgos de nuestros experimentos.

Guiados por las herramientas del ciclo empírico de *design science* [28], hemos procurado desarrollar artículos rigurosos, que satisfagan los estándares de calidad planteados por la comunidad científica [21]. En nuestros experimentos han participado un total de 93 sujetos: 23 expertos y 70 no expertos, lo que nos ha permitido aumentar el poder estadístico de nuestros resultados. Hemos utilizado dominios novedosos del contexto

Tabla 2: Resumen de respuestas, experimentos, resultados y hallazgos por pregunta de investigación

RQ_1	**¿Son los beneficios de MDD fieles a la realidad del desarrollo software actual?**		
Respuesta	Los beneficios de MDD respecto CcD siguen vigentes aumentando la fidelidad de la comparación con entornos de desarrollo real.		
Experimento	**Nº de sujetos**	**Factores analizados**	**Variables respuesta**
	44 9 expertos 35 no expertos	Contexto de desarrollo (MDD/CcD) Experiencia	**Objetivas:** Corrección y Eficiencia **Subjetivas:** Satisfacción (Utilidad Percibida, Facilidad de Uso Percibida, Intención de Uso)
Resultados	· La Corrección, la Eficiencia y la Satisfacción mejoran significativamente cuando los sujetos desarrollan utilizando MDD en lugar de CcD. · A pesar de las grandes mejoras en Corrección y Eficiencia, la Intención de Uso de MDD no alcanza valores máximos. · El factor experiencia no afecta significativamente a los cambios en las variables respuesta		
Hallazgos	· Un framework de dominio por sí solo no es suficiente para lograr los beneficios de MDD. · Tanto los desarrolladores con experiencia como los desarrolladores sin experiencia se benefician del uso de MDD. · La pérdida de control sobre el código automáticamente generado en contextos MDD influye negativamente en la Intención de Uso		
RQ_2	**¿Los profesionales software valoran adecuadamente los modelos que utilizan en sus desarrollos?**		
Respuesta	Los profesionales software no valoran correctamente la utilidad de los modelos que desarrollan.		
Experimento	**Nº de sujetos**	**Factores analizados**	**Variables respuesta**
	31 9 expertos 22 no expertos	Lenguaje de modelado (UML/DSL) Experiencia, Periodo y Secuencia	**Objetivas:** Corrección y Eficiencia **Subjetivas:** Satisfacción (Utilidad Percibida, Facilidad de Uso Percibida, Intención de Uso) Utilidad percibida del modelo realizado
Resultados	· La Corrección, la Eficiencia y la Satisfacción mejoran significativamente cuando los sujetos modelan utilizando el DSL en lugar de UML. · A pesar de la corrección de los modelos para ser utilizados en contextos MDD, los sujetos los clasifican como 'bocetos'.		
Hallazgos	· Los sujetos más experimentados generan modelos más correctos de forma más eficiente. · Los sujetos subestiman el potencial de sus propios modelos. · El problema de adopción de MDD no se limita a elegir el lenguaje de modelado que mejor funciona en un determinado contexto.		
RQ_3	**¿Cómo es el desempeño de los profesionales software en tareas de mantenimiento en contextos MDD?**		
Respuesta	Los sujetos con experiencia obtiene un mejor desempeño en las tareas de FL que los sujetos sin experiencia. Sin embargo, ambos grupos empeoran de igual forma cuando realizan rFL en lugar de cFL sin que perciban el aumento de la dificultad.		
Experimento	**Nº de sujetos**	**Factores analizados**	**Variables respuesta**
	18 5 expertos 13 no expertos	Alcance de la FL (cFL/rFL) Experiencia, Periodo y Secuencia	**Objetivas:** Rendimiento y productividad **Subjetivas:** Dificultad percibida
Resultados	· El rendimiento y la productividad empeoran significativamente cuando se realiza rFL en lugar de cFL · A pesar de que el rendimiento y la productividad disminuyen en casi un 50 %, la dificultad percibida por los sujetos no sufre variaciones significativas.		
Hallazgos	· Los sujetos subestiman la dificultad de las tareas de rFL en contextos MDD. · El tamaño y la dispersión de las características permiten explicar el rendimiento y la productividad de la FL: A mayor tamaño y menor dispersión de las características mejores son el rendimiento y la productividad de la FL. · La propagación de errores afecta negativamente al rendimiento y a la productividad de la rFL. · El proceso de eliminación, de elementos no significativos para la característica en los modelos, puede favorecer el rendimiento de la FL.		

industrial, en particular, la ingeniería del software para videojuegos. A pesar de ser una de las industrias con un crecimiento más rápido, la ingeniería del software para videojuegos ha sido identificada como un área de conocimiento que necesita más investigación fundamental [3], lo que la convierte en un dominio original para explorar la adopción de MDD.

En los modelos estadísticos utilizados en el análisis de datos, hemos considerados factores adicionales como los derivados de los diseños cruzados (secuencia y periodo) o la experiencia. Los primeros nos han permitido considerar en nuestras conclusiones los efectos de los factores de diseño. El factor experiencia nos ha permitido descartar que sólo los sujetos sin experiencia se benefician de MDD, como conjeturaban autores previos [17, 20]. Tanto los profesionales con experiencia como aquellos sin experiencia se benefician de igual forma de MDD [10]. Por otra parte, el factor experiencia también indica que los sujetos con experiencia en modelado modelan mejor (de forma más correcta y eficiente) y valoran mejor la utilidad de los modelos que realizan, que los sujetos sin experiencia [9]. También en nuestro tercer trabajo [11], los sujetos con conocimientos en el DSL utilizado tuvieron mayor desempeño en las tareas. Sin embargo, ni los sujetos con experiencia ni los sujetos con experiencia percibieron la dificultad de la localización de características para la reingeniería, y su rendimiento y productividad disminuyó significativamente independientemente de su experiencia.

En el diseño de nuestros experimentos hemos incorporado medidas objetivas y subjetivas. Esto nos ha permitido analizar desde una dimensión adicional la utilización de los modelos en los procesos software. Contrastar ambos tipos de medidas nos has permitido concluir que la percepción que los sujetos tienen de los modelos afecta negativamente en la adopción de enfoques MDD. Para entender el modelado en la ingeniería del software, es necesario contrastar la opinión de los sujetos con datos objetivos sobre su desempeño.

En trabajos previos, se atribuye la falta de éxito de MDD a carencias como la falta de herramientas adecuadas [14, 4, 8, 25, 24, 6, 1, 13, 18, 2, 16], la complejidad de los lenguajes y estándares de modelado [23, 12, 15, 24], la falta de profesionales con conocimientos específicos sobre la práctica MDD [15, 24, 1, 27, 2] o al elevado esfuerzo en la formación de

los profesionales [18, 16]. En nuestros experimentos, no todos los sujetos tenían formación previa sobre los lenguajes de modelado específicos utilizados, y a lo sumo se ha invertido media hora durante los experimentos a explicarles la utilización de estos lenguajes. Todos ellos fueron capaces de desarrollar las tareas asociadas a los experimentos con rendimientos aceptables. A pesar de que los modelos realizados por los sujetos sean capaces de generar código automáticamente [9] y con mayor rendimiento al que obtienen desarrollando el código con otros enfoques [10], los sujetos subestimaron la utilidad [9] y la complejidad [11] de los modelos, y no mostraron predisposición a utilizarlos [10, 9]. A partir de estos resultados, consideramos que las causas que hacen que MDD no haya tenido el éxito que se esperaba pueden tener más que ver con la percepción que los ingenieros tienen del modelado, que con otras carencias descritas en los trabajos previos.

Esta investigación nos lleva a concluir que, para entender con profundidad las causas por las que MDD todavía no ha tenido el éxito esperado, es necesario poner el foco de atención en los factores humanos. Hasta ahora las conferencias de ingeniería del software y, por tanto, la comunidad de ingeniería del software, han tenido una perspectiva principalmente técnica, donde los factores humanos pueden no haber estado suficientemente bien representados. Posiblemente sea necesario explorar la parte social y cultural del desarrollo software desde otras perspectivas como las propuestas por la comunidad de investigadores de *The ACM CHI Conference on Human Factors in Computing Systems*. Esta comunidad invierte la importancia asignada a las componentes técnicas y a los factores humanos respecto a la realizada por la comunidad de ingeniería de software. Una futura línea de investigación sobre factores humanos en MDD puede ser clave para que MDD alcance el éxito pronosticado. De hecho, ese es el trabajo que continua esta tesis.

> Teniendo en cuenta el estado actual de madurez de MDD, las evidencias de esta tesis apuntan a que su éxito depende de factores humanos, y no de factores técnicos. Es necesaria una línea de investigación centrada en el análisis de los factores humanos en contextos MDD.

Consideramos que esta tesis también es relevante para formar a los profesionales software en modelado y, por ende, para la adopción de MDD. Si los ingenieros fueran capaces de evaluar correctamente la utilidad y la complejidad de los modelos que desarrollan y utilizan, podrían explotar el potencial de MDD. Desde nuestro punto de vista, si queremos que MDD tenga el éxito pronosticado hace ya casi 20 años, deberíamos ser capaces de modificar la percepción que los ingenieros tienen del modelado.

Bibliografía

[1] Luciane Telinski Wiedermann Agner y col. «A Brazilian survey on UML and model-driven practices for embedded software development». En: *Journal of Systems and Software* 86.4 (2013), págs. 997-1005 (vid. págs. 133, 137).

[2] Deniz Akdur, Vahid Garousi y Onur Demirörs. «A survey on modeling and model-driven engineering practices in the embedded software industry». En: *Journal of Systems Architecture* 91 (2018), págs. 62-82 (vid. págs. 132, 133, 137).

[3] Apostolos Ampatzoglou y Ioannis Stamelos. «Software engineering research for computer games: A systematic review». En: *Information and Software Technology* 52.9 (2010), págs. 888-901 (vid. pág. 137).

[4] Bente Anda y col. «Experiences from introducing UML-based development in a large safety-critical project». En: *Empirical Software Engineering* 11.4 (2006), págs. 555-581 (vid. págs. 133, 137).

[5] Maicon Bernardino, Elder M Rodrigues y Avelino F Zorzo. «Performance Testing Modeling: an empirical evaluation of DSL and UML-based approaches». En: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. 2016, págs. 1660-1665 (vid. pág. 133).

[6] Michel RV Chaudron, Werner Heijstek y Ariadi Nugroho. «How effective is UML modeling?» En: *Software & Systems Modeling* 11.4 (2012), págs. 571-580 (vid. págs. 132, 133, 137).

[7] Andrea De Lucia y col. «An experimental comparison of ER and UML class diagrams for data modelling». En: *Empirical Software Engineering* 15.5 (2010), págs. 455-492 (vid. pág. 133).

[8] Brian Dobing y Jeffrey Parsons. «How UML is used». En: *Communications of the ACM* 49.5 (2006), págs. 109-113 (vid. págs. 133, 137).

[9] África Domingo y col. «Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2021, págs. 483-498 (vid. págs. 132, 137, 138).

[10] África Domingo y col. «Evaluating the benefits of model-driven development». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2020, págs. 353-367 (vid. págs. 130, 137, 138).

[11] África Domingo y col. «Evaluating the influence of scope on feature location». En: *Information and Software Technology* 140 (2021), pág. 106674. DOI: `https://doi.org/10.1016/j.infsof.2021.106674` (vid. págs. 134, 135, 137, 138).

[12] Karl Frank. *Keeping your business relevant with Model Driven Architecture (MDA). The Borland advantage*. White Paper. Object Management Group (OMG), 2004 (vid. pág. 137).

[13] Tony Gorschek, Ewan Tempero y Lefteris Angelis. «On the use of software design models in software development practice: An empirical investigation». En: *Journal of Systems and Software* 95 (2014), págs. 176-193 (vid. págs. 132, 137).

[14] Martin Grossman, Jay E Aronson y Richard V McCarthy. «Does UML make the grade? Insights from the software development community». En: *Information and Software Technology* 47.6 (2005), págs. 383-397 (vid. págs. 132, 133, 137).

[15] Brent Hailpern y Peri Tarr. «Model-driven development: The good, the bad, and the ugly». En: *IBM systems journal* 45.3 (2006), págs. 451-461 (vid. pág. 137).

[16] Rodi Jolak y col. «Model-based software engineering: A multiple-case study on challenges and development efforts». En: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, págs. 213-223 (vid. págs. 137, 138).

[17] T Kapteijns y col. «A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare». En: *From code centric to model centric software engineering Practices Implications and ROI* (2009) (vid. pág. 137).

[18] Nadja Christiane Marko y col. «Model-based engineering for embedded systems in practice». En: *Research reports in software engineering and management* (2014), págs. 1-48 (vid. págs. 133, 137, 138).

[19] Mari Carmen Otero y José Javier Dolado. «Evaluation of the comprehension of the dynamic modeling in UML». En: *Information and Software Technology* 46.1 (2004), págs. 35-53 (vid. pág. 133).

[20] Jose Ignacio Panach y col. «In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction». En: *Information and Software Technology* (2015). ISSN: 09505849 (vid. pág. 137).

[21] Paul Ralph y col. «Empirical standards for software engineering research». En: *arXiv preprint arXiv:2010.03525* (2020) (vid. pág. 135).

[22] Iris Reinhartz-Berger y Dov Dori. «OPM vs. UML–Experimenting with Comprehension and Construction of Web Application Models». En: *Empirical Software Engineering* 10.1 (2005), págs. 57-80 (vid. pág. 133).

[23] Bran Selic. «The pragmatics of model-driven development». En: *IEEE software* 20.5 (2003), págs. 19-25 (vid. pág. 137).

[24] Yashwant Singh y Manu Sood. «Model driven architecture: A perspective». En: *2009 IEEE International Advance Computing Conference*. IEEE. 2009, págs. 1644-1652 (vid. pág. 137).

[25] Miroslaw Staron. «Adopting model driven software development in industry–a case study at two companies». En: *International Conference on Model*

Driven Engineering Languages and Systems. Springer. 2006, págs. 57-72 (vid. págs. 133, 137).

[26] Harald Störrle. «How are Conceptual Models used in Industrial Software Development? A Descriptive Survey». En: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017, págs. 160-169 (vid. pág. 132).

[27] Marco Torchiano y col. «Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry». En: *Journal of Systems and Software* 86.8 (2013), págs. 2110-2126 (vid. pág. 137).

[28] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014 (vid. pág. 135).

[29] Andreas Zendler y col. «Experimental comparison of coarse-grained concepts in UML, OML, and TOS». En: *Journal of systems and Software* 57.1 (2001), págs. 21-30 (vid. pág. 133).

Parte IV

Conclusiones

Conclusiones

Esta parte presenta las conclusiones finales de la tesis, resumiendo el objetivo, el estudio realizado y los resultados de nuestro trabajo. También se presentan futuras líneas de investigación que pueden contribuir a ampliar estos resultados.

En esta tesis se presenta el compendio de tres artículos a partir de los cuales se intenta ahondar en las causas por las que el Desarrollo Dirigido por Modelos (MDD) no ha tenido el éxito pronosticado hace ya casi 20 años. Cada uno de estos artículos trata de responder empíricamente a una pregunta de investigación: ¿Son los beneficios de MDD frente a otros enfoques fieles a la realidad del desarrollo software actual?, ¿Los profesionales software valoran adecuadamente los modelos que utilizan en el desarrollo de software?, ¿Cómo es el desempeño de los profesionales software en tareas de mantenimiento en entornos MDD?.

En el primer artículo que compone esta tesis [2], se presenta un experimento en el que se compara MDD y CcD, que va más allá del estado del arte en términos de fidelidad al mundo real y poder estadístico. Para aumentar la fidelidad con el desarrollo de software actual, utilizamos frameworks de dominio tal y como se utilizan en los desarrollos del contexto industrial. La potencia estadística se mejora aumentando el tamaño de la muestra. Nuestros resultados desafían las ideas anteriores que limitan los beneficios de MDD a los ejercicios académicos y a los desarrolladores sin experiencia. Sin embargo, a pesar de las grandes mejoras en el software generado, en términos de corrección y eficiencia, la intención de uso de MDD no alcanza los valores máximos. Los resultados de este trabajo nos permiten afirmar que los beneficios de MDD son fieles a la realidad del software, y sugieren que la percepción que los sujetos tienen de los modelos afecta negativamente a su intención de uso.

En el segundo artículo de esta tesis [1], presentamos un experimento que nos permite analizar la valoración que hacen los sujetos de los modelos que ellos mismos realizan. En el experimento se compara UML y un DSL cuando los sujetos modelan en el contexto de desarrollo de un videojuego comercial. Nuestros resultados revelan un problema importante: no es que los modelos no estén preparados para ser utilizados como programas porque hay que mejorar los lenguajes, las herramientas, o la formación asociada, es que, los sujetos piensan en los modelos más como bocetos que como programas. La combinación de medidas objetivas y subjetivas en el análisis de resultados, nos permiten afirmar que los sujetos no valoran correctamente los modelos que desarrollan y subestiman su potencial.

En el tercer artículo de esta tesis [3], presentamos un experimento que analiza el desempeño de los profesionales software en tareas de mantenimiento en contextos MDD. En el experimento se analiza el rendimiento, la productividad y la dificultad percibida de los ingenieros cuando realizan diferentes tareas de localización de características en un contexto MDD. Los objetos experimentales se extraen de un SPL del mundo real que utiliza un DSL para generar el software de sus productos. Los sujetos con experiencia en el DSL obtienen mejores resultados que los sujetos sin experiencia, sin embargo, para todos los sujetos, el rendimiento y la productividad disminuyen significativamente cuando las tareas de localización de características se realizan en una familia de productos en lugar de en un único producto. Por otra parte, la dificultad percibida entre unas tareas y otras no presenta cambios significativos. Los ingenieros infravaloran la dificultad de la localización de características en una familia de modelos de producto. De nuevo, las medidas subjetivas del experimento muestran como los sujetos tienden a subestimar el contexto MDD. Un análisis más exhaustivo de los resultados indica que, el tamaño de las características o la dispersión de las mismas pueden explicar los cambios en el rendimiento y la productividad cuando cambia el objetivo de la búsqueda. Estos resultados aumentan el conocimiento sobre las dificultades asociadas a las tareas de localización, y pueden ayudar a desarrollar y evaluar nuevas herramientas que aprovechen los puntos fuertes de los ingenieros, y que compensen sus puntos débiles y las dificultades intrínsecas a los problemas de localización de características.

En nuestros experimentos, hemos utilizado dominios novedosos del contexto industrial real y han participado casi cien sujetos, lo que nos ha permitido aumentar la fidelidad del trabajo al desarrollo de software actual y el poder estadístico de nuestros resultados. En todos nuestros trabajos hemos considerado medidas objetivas y subjetivas, lo que nos ha permitido analizar desde una dimensión adicional la utilización de modelos en los procesos software. A partir de nuestros resultados, podemos concluir que las causas que hacen que MDD no haya tenido el éxito que se esperaba pueden tener más que ver con la percepción que los ingenieros tienen del modelado, que con otras carencias descritas en trabajos previos. A pesar de que los modelos realizados por los sujetos sean capaces de generar código automáticamente [1] y con mayor rendi-

miento al que obtienen desarrollando el código con otros enfoques [2], los sujetos subestiman la utilidad [1] y la complejidad [3] de los modelos, y no muestran predisposición a utilizarlos [2, 1]. Este problema no ha sido descrito en trabajos anteriores, quizá por centrarse en medidas objetivas para evaluar el rendimiento del modelado, o en medidas subjetivas para clasificar los estilos de modelado o la percepción de los usuarios. Nuestros resultados sugieren que, para entender el modelado de software, es necesario contrastar la opinión de los sujetos con las mediciones. Las futuras evaluaciones empíricas en ingeniería del software deberían tener en cuenta tanto medidas objetivas como subjetivas.

El primer trabajo de este compendio apoya claramente la misma conclusión que, de una forma u otra, encontramos en la gran mayoría de los trabajos previos analizados: la práctica de MDD reporta notables beneficios a la ingeniería del software. Esta conclusión, refuerza la necesidad de desarrollar trabajos que, como esta tesis, ayuden a entender por qué MDD no es ampliamente utilizado en la práctica real del desarrollo software. Nuestros artículos pueden ayudar a formar a los profesionales software en modelado, ayudándoles a modificar la percepción que tienen de los modelos software. Para que los ingenieros software exploten el potencial del modelado y de los enfoques MDD, es necesario incrementar su capacidad para evaluar la utilidad y la complejidad de los modelos que desarrollan y utilizan.

Nuestros resultados nos llevan a responder explícitamente al objetivo de esta tesis, estudiar empíricamente las causas por las que MDD no ha tenido el éxito pronosticado por la comunidad. Concluimos que la percepción que los ingenieros tienen del modelado influye negativamente en su intención de uso, y que tienden a subestimar los enfoques MDD. Consideramos que, para entender con profundidad las causas por las que MDD todavía no ha tenido el éxito esperado, es necesario poner el foco en los factores humanos. Para que MDD alcance el éxito pronosticado por la comunidad, es necesario ampliar el conocimiento sobre los factores humanos que intervienen en la adopción de MDD.

Como primer paso en el análisis de factores humanos que pueden afectar al éxito de MDD, trabajamos en una nueva aproximación empírica en la que cambiamos la perspectiva de la investigación. Queremos explorar

si existe algún tipo de perfil específico dentro de los profesionales software que sea más propenso a tener éxito en contextos MDD. Para ello, caracterizaremos a los sujetos en función de diferentes factores como su experiencia profesional, su conocimiento previo sobre modelado, su habilidad en otros enfoques o su rendimiento académico, y analizaremos experimentalmente si existe alguna combinación de estos factores que favorezca el éxito de MDD. Por otra parte, algunos autores han insistido en la importancia de herramientas adecuadas para la correcta implantación de MDD. También nos proponemos comparar el rendimiento de profesionales software al utilizar diferentes herramientas que implementen el paradigma MDD. Este análisis puede ayudar a definir las características que han de tener este tipo de herramientas para favorecer la adopción de MDD en entornos reales de desarrollo software.

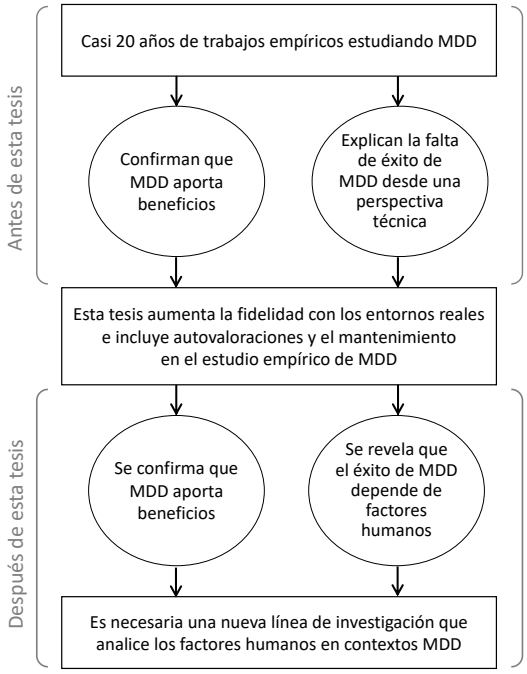

Figura 4: Conclusión de esta tesis

Finalizo este libro con la Figura 4, que sintetiza gráficamente las principales aportaciones de esta tesis y su conclusión.

Bibliografía

[1] África Domingo y col. «Comparing UML-Based and DSL-Based Modeling from Subjective and Objective Perspectives». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2021, págs. 483-498 (vid. págs. 146-148).

[2] África Domingo y col. «Evaluating the benefits of model-driven development». En: *International Conference on Advanced Information Systems Engineering*. Springer. 2020, págs. 353-367 (vid. págs. 146, 148).

[3] África Domingo y col. «Evaluating the influence of scope on feature location». En: *Information and Software Technology* 140 (2021), pág. 106674. DOI: https://doi.org/10.1016/j.infsof.2021.106674 (vid. págs. 147, 148).