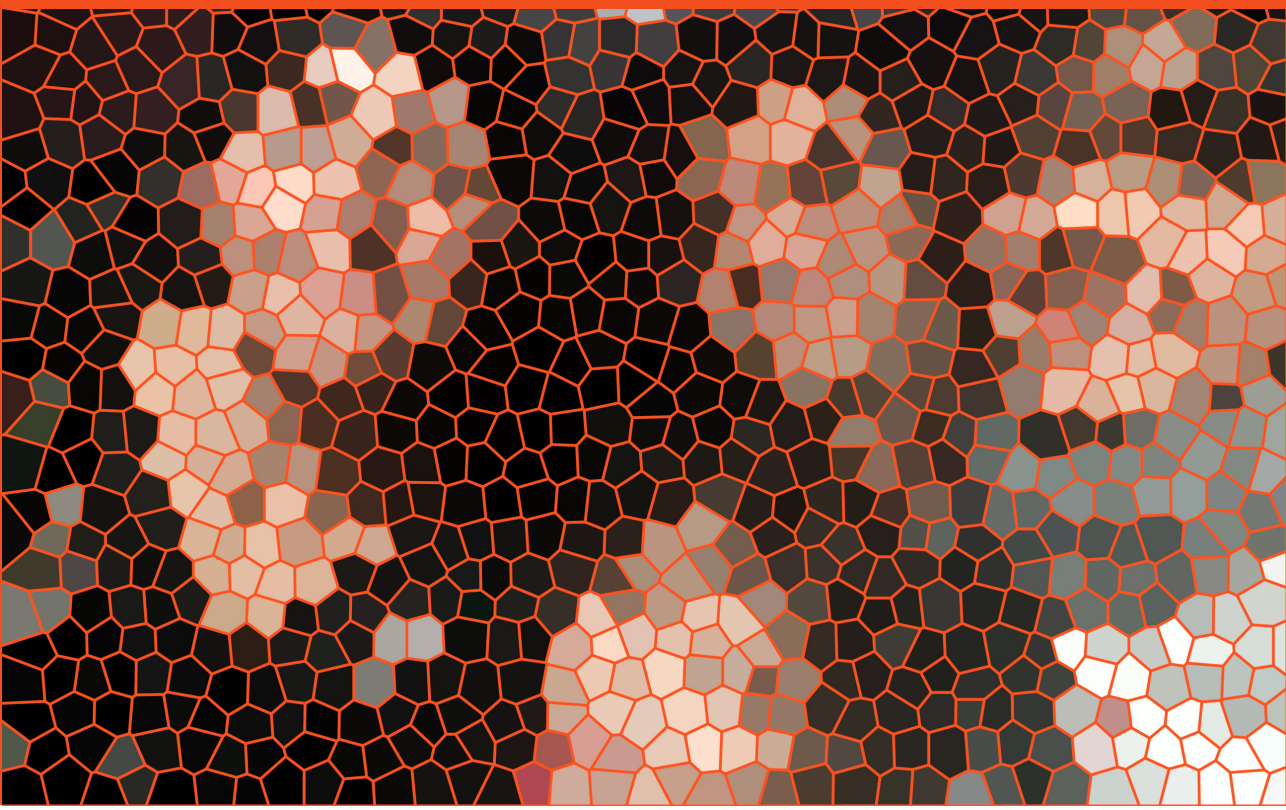
Analysis and Improvement of a Software Production Process based on the Combination of Model Driven Development and Software Product Lines

June 2018

Jorge **Echeverría Ochoa**



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Supervisors

Óscar **Pastor López**

Carlos **Cetina Englada**

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Analysis and Improvement of a Software Production Process based on the Combination of Model Driven Development and Software Product Lines

Junio de 2018

Autor: Jorge Echeverría Ochoa

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

Dedicatoria

A Judith,
la primera luz,
la mujer guerrera,
la mágica oportunidad.

A Candela y Aitana,
el brillo en la oscuridad,
la felicidad soñada.

A mis padres,
el origen y el cariño.

Agradecimientos

En primer lugar, agradecer a Carlos Cetina que me embarcara en esta aventura, me haya guiado y acompañado. Quiero dar las gracias a Óscar Pastor, he tenido la suerte de disfrutar de su compañía y sabiduría.

Esta tesis es fruto del trabajo de investigación realizado durante los últimos años dentro del grupo de investigación SVIT Research Group. No puedo olvidar a todos aquellos que han hecho posible y han colaborado en mis investigaciones: Francisca Pérez, Ignacio Panach, Jaime Font, Lorena Arcega, Manuel Ballarín y Raúl Lapeña; de todos ellos he aprendido valiosas enseñanzas.

Mi agradecimiento a todos mis compañeros de trabajo en la Universidad San Jorge, especialmente a África Domingo, Gabriel Marro, Jesús Carro, Violeta Monasterio, David Chinarro y Jesús Bergues; es un placer compartir vuestro tiempo. Un especial reconocimiento a Antonio Estepa por la estupenda portada que ha diseñado.

Gracias a las familias Joven-Zubiría y Caballero-Echeverría por su sustento, cariño y ayuda.

Finalmente, agradecer a todas aquellas personas que, tan generosamente, han participado en los estudios empíricos que son la base de esta tesis; muy especialmente, a la división de placas de inducción de BSH y la empresa CAF.

Resumen

La reutilización es un factor clave para reducir los costos y mejorar la calidad de las propiedades de productos software como la seguridad, fiabilidad o rendimiento. Siguiendo este factor surge la aproximación para el desarrollo de software de Líneas de Productos Software; esta aproximación promete, entre otras cosas, acortar el tiempo del desarrollo de los sistemas software y reducir significativamente los costes de desarrollo y mantenimiento. Por otro lado, el Desarrollo Dirigido por Modelos es un enfoque para el desarrollo de software que propone el uso de modelos en varios niveles de abstracción y transformaciones de modelo como artefactos principales. El uso de modelos como los principales artefactos en el desarrollo de software ofrece muchas ventajas a los desarrolladores, por ejemplo, las transformaciones de modelo permiten la conversión de un modelo fuente en otro modelo objetivo, el aumento del nivel de abstracción permite a los desarrolladores centrarse en el problema a resolver y restar importancia a los detalles de implementación. Ambos paradigmas, en la búsqueda de optimizar el tiempo de producción y calidad en el software generado, pueden reunir importantes ventajas en el proceso de producción de software.

La combinación de Desarrollo Dirigido por Modelos y Líneas de Producto Software para producir productos software requiere la identificación de nuevos retos y necesidades de los *stakeholders* involucrados. La investigación presentada en esta tesis tiene el objetivo, apoyada en varios estudios empíricos realizados en entornos industriales, de aumentar el conocimiento y realizar una serie de propuestas de mejora del proceso de desarrollo software fundamentado en la combinación de Desarrollo Dirigido por Modelos y Líneas de Producto Software. Para alcanzar este objetivo se han estudiado cuatro dimensiones: procesado de requisitos, usabilidad, comprensión (en la configuración de productos) y gestión de errores. Cada una de estas dimensiones ha sido abordada en un estudio empírico, estudios presentados en trabajos de investigación y que forman la parte nuclear de esta tesis.

Como resultado del trabajo realizado en esta tesis se han elaborado una serie de propuestas para mejorar el proceso de desarrollo software basado en la combinación de Desarrollo Dirigido por Modelos y Líneas de Producto Software y se han generado siete trabajos de investigación. Cinco de estos trabajos han sido presentados en conferencias de relevancia en el ámbito de la Ingeniería del Software: CAiSEForum'15, CAiSE'16, ESEM'16, ISD'17 y ESEM'17. Estos

resultados de investigación han sido aplicados en el proceso de desarrollo de software de placas de inducción de la división de electrodomésticos de BSH (Bosch, Siemens, Gaggenau, Neff y Balay) y están siendo utilizados para su aplicación en la actual implantación para el desarrollo de software del PLC que controla los trenes en la empresa Construcciones y Auxiliar de Ferrocarriles.

Abstract

Software reuse is a key factor in reducing costs and improving the quality of software product properties such as security, reliability, or performance. Taking this factor in account, the Software Product Line approach appears for software development. This approach promises to decrease the time spent in developing software systems and to significantly reduce the costs for development and maintenance of software systems, among other things. In addition, Model Driven Development is an approach for software development that proposes the use of models at various levels of abstraction and model transformations as main artifacts. The use of models as the main artifacts in software development offers many advantages for developers. For instance, model transformations allow the conversion of a source model into another target model. The increase in the level of abstraction allows the developers to focus on the problem to solve, subtracting importance to the implementation details.

Both paradigms search to optimize the production time and quality of the generated software, and can result in important advantages in the software production process. However, the combination of Model Driven Development and Software Product Lines to develop software products requires the identification of the challenges and needs of the involved stakeholders. The research presented in this dissertation, supported by several empirical studies carried out in industrial environments, aims to increase the knowledge in the field and to do a set of proposals to improve the software development process based on the combination of Model Driven Development and Software Product Lines. To achieve this objective, four dimensions have been studied: requirements processing, usability, comprehension (configuring software products), and error management. The dimensions have been addressed through empirical studies, presented in research papers. These papers conform the core of this dissertation.

As result of the work carried out for this dissertation, a set of proposals to improve the software development process based on the combination of Model Driven Development and Software Product Lines have been generated. Furthermore, seven research papers have been published. Five of these works have been presented at relevant conferences in the Software Engineering field: CAiSE Forum'15, CAiSE'16, ESEM'16, ISD'17 and ESEM'17. These research results have been applied in the software development process of the induction hobs from the electrical appliances division of BSH (under the brands

Bosch, Siemens, Gaggenau, Neff, and Balay), and are also being used in the current implementation of the PLC that controls the trains manufactured by the Construcciones y Auxiliar de Ferrocarriles company.

Resum

La reutilització és un factor clau per a reduir els costos i millorar la qualitat de les propietats de productes programari com la seguretat, fiabilitat o rendiment. Seguint aquest factor sorgeix l'aproximació per al desenvolupament de programari utilitzant Línies de Productes Programari; aquesta aproximació promet, entre altres coses, escurçar el temps del desenvolupament dels sistemes programari i reduir significativament els costos de desenvolupament i manteniment. D'altra banda, el Desenvolupament Dirigit per Models és un enfocament per al desenvolupament de programari que proposa l'ús de models en diversos nivells d'abstracció i transformacions de model com artefactes principals. L'ús de models com els principals artefactes en el desenvolupament de programari ofereix molts avantatges als desenvolupadors, per exemple, les transformacions de model permeten la conversió d'un model font en un altre model objectiu, l'augment del nivell d'abstracció permet als desenvolupadors centrar-se en el problema a resoldre i restar importància als detalls d'implementació. Tots dos paradigmes, en la cerca d'optimitzar el temps de producció i qualitat en el programari generat, poden reunir importants avantatges en el procés de producció de programari.La combinació de Desenvolupament Dirigit per Models i Línies de Producte Programari per a produir productes programari requereix la identificació de nous reptes i necessitats dels *stakeholders* involucrats. La recerca presentada en aquesta tesi té l'objectiu, recolzada en diversos estudis empírics realitzats en entorns industrials, d'augmentar el coneixement i realitzar una sèrie de propostes de millora del procés de desenvolupament de programari fonamentat en la combinació de Desenvolupament Dirigit per Models i Línies de Producte Programari. Per a aconseguir aquest objectiu s'han estudiat quatre dimensions: processament de requisits, usabilitat, comprensió (en la configuració de productes) i gestió d'errors. Cadascuna d'aquestes dimensions ha sigut abordada en un estudi empíric, estudis presentats en treballs de recerca i que formen la part nuclear d'aquesta tesi.

Com a resultat del treball realitzat en aquesta tesi s'han elaborat una sèrie de propostes per a millorar el procés de desenvolupament programari basat en la combinació de Desenvolupament Dirigit per Models i Línies de Producte Programari i s'han generat set treballs de recerca. Cinc d'aquests treballs han sigut presentats en conferències de rellevància en l'àmbit de l'Enginyeria del Programari: CAiSEForum'15, CAiSE'16, ESEM'16, ISD'17 i ESEM'17. Aquests resultats de recerca han sigut aplicats en el procés de desenvolupament de pro-

gramari de plaques d'inducció de la divisió d'electrodomèstics de BSH (Bosch, Siemens, Gaggenau, Neff i Balay) i estan sent utilitzats per a la seua aplicació en l'actual implantació per al desenvolupament de programari del PLC que controla els trens en l'empresa Construcciones y Auxiliar de Ferrocarriles.

Índice general

Índice de figuras

Índice de tablas

Parte I

Introducción

1

Introducción

Índice

1.1 Motivación

El Desarrollo Dirigido por Modelos (*Model Driven Development, MDD*) se ha mostrado como un paradigma en el desarrollo software que permite reducir costes y mejorar el tiempo de desarrollo [1]. Por otro lado, las Líneas de Producto Software (*Software Product Line, SPL*) se han mostrado como un sistema eficaz para mejorar la calidad del software fundamentándose en el modelado de la variabilidad [2].

Investigaciones recientes revelan que la mayoría de las SPLs se construyen cuando ya existen productos [3], por lo tanto, la configuración de los productos ya existentes dirigen la creación de la SPL [4]. Este método es conocido como extractivo, partiendo de un sistema existente se inicia una SPL, formalizando la variabilidad entre un conjunto de productos similares acorde a un modelo de variabilidad. La SPL resultante es capaz de generar los productos utilizados como entrada y generar nuevos productos con una reutilización sistemática.

Nuestros esfuerzos de investigación buscan, sustentado en varios estudios empíricos, profundizar en el conocimiento en el desarrollo software en entornos industriales donde el paradigma de desarrollo MDD ha sido complementado con una SPL mediante la formalización de la variabilidad. Este proceso, siguiendo la descripción del párrafo anterior, se ha dado en dos de nuestros socios industriales, convirtiéndose este contexto real en el origen de este trabajo de tesis. Estos socios industriales son Construcciones y Auxiliar de Ferrocarriles (CAF) y la división de grupos de electrodomésticos de BSH.

Nuestros socios industriales, CAF y la división de grupos de electrodomésticos de BSH, se encuentran en diferentes niveles de adopción del proceso de desarrollo software combinación de MDD y SPL, siendo el caso de BSH un nivel de adopción mayor; por esta razón, se han podido realizar un mayor número de estudios empíricos de los presentados en esta tesis en el contexto de la división de grupos de electrodomésticos de BSH.

1.2 Descripción del problema

Partiendo del contexto descrito en el anterior apartado (1.1), el objetivo de esta tesis es analizar en profundidad y realizar una serie de propuestas para mejorar los procesos de desarrollo software basados en MDD+SPL en entornos industriales. El desarrollo software consiste en administrar el proceso de construir un producto o sistema que cubra las necesidades de los usuarios, probarlo,

instalarlo en un ambiente productivo, mantenerlo y hacerlo evolucionar con los cambios de negocio [5].

Existe la posibilidad de abordar diferentes dimensiones en un desarrollo software basado en MDD+SPL y que son científicamente relevantes. Por ejemplo, en [6] se estudian flexibilidad y soporte para el usuario final, en [7] se aborda usabilidad, en [8] se intenta mejorar la productividad, en el sector del automóvil cobran especial importancia la tolerancia a fallos, disponibilidad, confiabilidad y rendimiento [9]. Involucramos a expertos de nuestros socios industriales para que estimaran que dimensiones eran más relevantes para ellos, considerando el impacto de las mismas en sus desarrollos. Por lo tanto, la elección de las dimensiones tratadas en esta tesis es por relevancia para la comunidad científica, por importancia para nuestros socios industriales y, por último, por ser viable abordarla en el tiempo disponible durante el desarrollo de la tesis. Como consecuencia de lo expresado anteriormente, en esta tesis se han estudiado cuatro dimensiones:

- **Procesado de requisitos**: La primera medida del éxito de un sistema software es el grado de cumplimiento del propósito para el cuál fue creado [10]. Este propósito está reflejado en la especificación de requisitos. En [11] se enumeran las causas de las que depende el éxito de un proyecto de desarrollo software, entre las razones principales se encuentra "un enunciado claro de requisitos". En esta misma dirección, en [12], se califica como fundamental para el éxito de cualquier proyecto software la correcta interpretación de los requisitos (en mucha ocasiones escritos en lenguaje natural). Desde la perspectiva de nuestro socio industrial CAF, y generalizable a entornos con similares características, es importante conocer como afecta la naturaleza de los requisitos en el proceso de generar modelos software a partir de los mismos.

- **Usabilidad**: La usabilidad es una cualidad que todo software debe optimizar para facilitar su utilización por parte de los usuarios finales. Desde los comienzos de la Ingeniería del Software se observó que la calidad está compuesta por un conjunto de cualidades, y una de ellas, es la usabilidad [13]. Nuestros socios industriales son conscientes de la importancia de la usabilidad dentro del proceso de desarrollo software: consideran que una mejora de la usabilidad puede mejorar parámetros como eficacia, eficiencia, satisfacción o facilidad de aprendizaje.

- **Comprensión**: Como queda reflejado en diversos modelos de calidad del software el entendimiento del software desarrollado es fundamental [14, 15] para desarrollar software de calidad. Por otro lado, como se describe

en [16] las herramientas de desarrollo son interesantes, pero es mucho más importante conocer el proceso y la técnica. Esta dimensión ha sido abordada porque, el análisis de la dimensión usabilidad en el proceso de desarrollo MDD+SPL, puso de manifiesto la dificultad de los usuarios para comprender las posibilidades de combinar fragmentos de modelo software, acorde a unas normas de variabilidad, para generar productos software.

- **Gestión de errores**: En todo proceso de desarrollo software "vivo" surgen errores que deben ser solucionados para permitir su correcto funcionamiento. Desde los primeros modelos de calidad de software, como por ejemplo el presentado por McCall [17] en su apartado correspondiente a mantenibilidad, se hace referencia a "el esfuerzo requerido para localizar y corregir un fallo en el programa con su entorno operacional". Las afirmaciones de la comunidad científica que recalcan la importancia de la gestión de errores, ligado al concepto de mantenibilidad del software en una SPL, han sido asumidas por nuestros socios industriales en el proceso de adopción de MDD+SPL. Esta circunstancia, unida a la consciencia de nuestros socios de la importancia de la gestión de errores en su desempeño diario, puso en valor la idoneidad de estudiar la gestión de errores en un proceso MDD+SPL.

La relevancia de las anteriores dimensiones (en algunos modelos de calidad denominadas características, subcaracterísticas o atributos) queda reflejada en trabajos como [18, 19, 20]. Otros estudios sobre modelos de calidad, como [21], concluye que los modelos de calidad deben ser: flexibles, para adaptarse a una organización y proyecto específicos; transparentes, para permitir una comprensión clara de su lógica, así como del significado de las dimensiones y las relaciones entre ellos; reutilizables, para ser reutilizado y mejorado en todas las SPLs. En este modelo se consideran enfoques de modelado de calidad existentes, pero concluyen que ninguno satisface completamente sus requisitos. Consideran que el modelo de calidad, incluyendo las dimensiones a evaluar, debe ser especificado en función del contexto. En [22] se presenta un modelo de calidad, se trata de una extensión del estándar ISO/IEC 25000 (SQuaRE) [23], que proporciona apoyo a las actividades de aseguramiento de la calidad y evaluación en desarrollos con el paradigma SPL. El primer paso es definir los atributos de calidad relevantes para el dominio. Se recorta el modelo genérico de calidad para SPL seleccionando solo los atributos de mayor interés en el dominio. Los atributos del dominio están identificados por revisión en la literatura del dominio, entrevistas con los usuarios y diseñadores de sistemas similares y mediante el uso de la industria correspondiente. En la misma línea,

[24] muestra una revisión sistemática sobre técnicas y métodos para evaluar la calidad de Líneas de Producto Software y, entre las conclusiones destaca la necesidad de realizar experimentos en el ámbito de las SPLs para la generación de evidencias y comprobar que métodos son aconsejables en función de cada situación.

Tomando como punto de partida el objetivo general de este trabajo de tesis enunciado en el inicio de este apartado y las dimensiones anteriormente descritas, se detallan los objetivos a alcanzar con este trabajo:

- Conocer como afecta la naturaleza de los requisitos para la generación de modelos software en un entorno industrial.
- Analizar la usabilidad en un proceso de desarrollo MDD+SPL en un entorno industrial.
- Estudiar la comprensión de la variabilidad de fragmentos de modelo software por parte de usuarios en el proceso de configuración de productos.
- Analizar como los ingenieros de software corrigen errores en un entorno industrial de desarrollo software donde se utiliza MDD+SPL.
- Proponer un conjunto de mejoras para el desarrollo de software MDD+SPL.

Para alcanzar los objetivos reseñados y, por lo tanto, estudiar cada una de las dimensiones citadas anteriormente, se han enunciado cuatro preguntas de investigación que forman el armazón de esta tesis. Cada dimensión tiene asociada una pregunta de investigación:

RQ1 Procesado de requisitos: ¿Cómo afecta la naturaleza de los requisitos para generar modelos software en un entorno de desarrollo MDD+SPL?

RQ2 Usabilidad: ¿Cuál es la usabilidad en un desarrollo software basado en MDD+SPL?

RQ3 Comprensión: ¿Cómo es la comprensión para configurar productos desde fragmentos de modelo?

RQ4 Gestión de errores: ¿Cómo se corrigen los errores en un entorno de desarrollo software basado en MDD+SPL?

La combinación de MDD y SPL parece tener una clara aportación a la producción de software sistemática; en entornos industriales, donde el software es

mantenido durante décadas, dimensiones como procesado de requisitos, usabilidad, comprensión (en la configuración) y gestión errores son esenciales. Por ello, queremos conocer como es el comportamiento de estas dimensiones en un entorno de desarrollo MDD+SPL. Cada una de las dimensiones es considerada en una de las preguntas de investigación, y las citadas preguntas han sido respondidas en un artículo de investigación, estos artículos forman la parte central de esta tesis (capítulos 5 a 8).

1.3 Resumen del trabajo

La Figura 1.1 muestra una descripción general del trabajo realizado como parte de esta tesis. Observando las distintas filas tenemos: (fila 1) identifica el objetivo que aborda la tesis; (fila 2) muestra las preguntas de investigación para alcanzar el citado objetivo; (fila 3) muestra las diferentes dimensiones que han sido estudiadas acorde a cada pregunta de investigación; (fila 4) enumera las publicaciones científicas generadas; (fila 5) enumera los proyectos de investigación en los que se ha contribuido; (fila 6) enumera los socios industriales donde se han realizado algunos de los estudios empíricos detallados posteriormente.

<table>
<tr><td>Objetivo</td><td colspan="4">Comprender y mejorar los procesos de desarrollo software basados en Desarrollo Dirigido por Modelos y Líneas de Producto Software</td></tr>
<tr><td>Preguntas de Investigación</td><td>Influencia de los requisitos en la generación de modelos</td><td>Usabilidad en un Desarrollo Dirigido por Modelos con Variabilidad</td><td>Comprensión cuando se trabaja con fragmentos de modelos</td><td>Gestión de errores en un Desarrollo Dirigido por Modelos con Variabilidad</td></tr>
<tr><td>Dimensiones</td><td>Procesado de requisitos</td><td>Usabilidad</td><td>Comprensión</td><td>Gestión de errores</td></tr>
<tr><td>Publicaciones</td><td>ESEM’17
ISD’17</td><td>CSIMQ’15
CAiSE Forum’15</td><td>CAiSE’16
IDoESE’16</td><td>ESEM’16</td></tr>
<tr><td>Proyectos de investigación</td><td colspan="4">VARIAMOS: Extracción de Variabilidad Dirigida por Modelos para la Adopción de Líneas de Producto Software Plan Nacional R+D+i y ERDF – TIN2015-64397-R
REVaMP$_2$: Round-trip Engineering and Variability Management Platform and Process Information Technology for European Advancement – ITEA 3 Call 2</td></tr>
<tr><td>Socios industriales</td><td colspan="4">BSH: Grupo de Electrodomésticos
Herramienta para gestión y extracción de variabilidad para el firmware de placas de inducción
CAF: Modelado de la variabilidad, evolución y generación de código para el software de sistemas ferroviarios</td></tr>
</table>

Figura 1.1: Resumen del trabajo para la realización de la tesis.

Se han publicado 7 trabajos de investigación que buscan responder a las preguntas de investigación citadas en el apartado 1.2. Los trabajos son los siguientes:

1. **The influence of Requirements in Model Software Development in an Industrial Environment (ESEM´17)** [25]: Estudia la dimensión *Procesado de requisitos*. Se trata de un experimento localizado en la fase de generación de modelos software desde requisitos. Aborda la diferencia al generar modelos software dependiendo de la naturaleza de los requisitos utilizados para describir las funcionalidad del software. Estos requisitos se clasifican en requisitos enriquecidos y no enriquecidos.

2. **Assessing the Performance of Automated Model Extraction Rules (ISD´17)** [26]: Estudia la dimensión *Procesado de requisitos*. Como en el caso anterior, este trabajo se localiza en la fase de generación de modelos software desde requisitos software. En este trabajo de investigación se hace una comparación entre los modelos software generados por expertos de dominio a partir de un conjunto de requisitos y los modelos software generados utilizando reglas automáticas de extracción de modelos a partir de los mismos requisitos.

3. **Usability Evaluation of Variability Modeling by means of Common Variability Language (CSiMQ´15)** [27]: Estudia la dimensión *Usabilidad*. En este estudio de caso se presenta una evaluación de usabilidad considerando los valores para la eficacia, eficiencia y satisfacción, dependiendo de tres diferentes tipos de tareas a realizar por ingenieros de software. Estos tres tipos de tareas son: configuración, ámbito y visualización. Las tareas estudiadas influyen en los fragmentos de modelo, la generación de modelos de productos y la especificación de la variabilidad.

4. **Usability Evaluation of Variability Modeling by means of Common Variability Language (CAiSEForum´15)** [28]: Se trata de una versión reducida del trabajo anterior y, por tanto, estudia la dimensión *Usabilidad*.

5. **Comprehensibility of Variability in Model Fragments for Product Configuration (CAiSE´16)** [29]: Aborda la dimensión *Comprensión*. Esta investigación afronta, con una perspectiva experimental, las dificultades que encuentran estudiantes de Ingeniería Informática y profesionales en desarrollo software para configurar productos software dependiendo de los tipos de fragmentos utilizados para generar el producto.

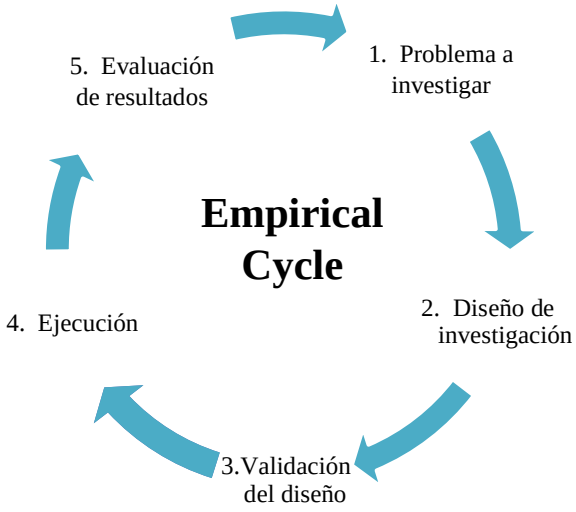


Figura 1.2: Ciclo de investigación empírica.

1. **Conocimiento del problema a investigar**: en esta fase, se buscan los temas principales a investigar y se formulan las preguntas de investigación.

2. **Diseño de investigación**: en esta fase, se ha diseñado cada uno de los estudios empíricos desarrollados para obtener respuestas a las anteriores preguntas de investigación.

3. **Validación del diseño**: La validación de los estudios ha sido realizada considerando tres ejes: posibilidad de repetir los estudios empíricos, consideraciones éticas hacia los participantes e inferencia de los estudios. Por otro lado, los estudios empíricos desarrollados han sido validados con la publicación de nuestros resultados.

4. **Ejecución de la investigación**: se han realizado 4 estudios empíricos, dando como resultado 7 trabajos de investigación.

5. **Evaluación de resultados**: Los estudios empíricos han mostrado resultados que han permitido obtener respuestas para nuestras preguntas de investigación e indican la dirección a seguir por las personas involucradas en procesos de desarrollo software dirigido por modelos con modelado de la variabilidad.

El proceso cíclico descrito anteriormente se ha desarrollado de forma iterativa: partiendo del estudio empírico inicial, el conocimiento producido (los resultados obtenidos) dirigen la investigación al estudio desde nuevas perspectivas. En esta disertación el ciclo ha sido aplicado 4 veces, uno para cada una de las dimensiones estudiadas.

Siguiendo el ciclo definido por Wieringa en [32], el primer ciclo comenzó con la conciencia del problema en un entorno industrial donde se desarrolla software bajo los paradigmas estudiados en este trabajo de tesis. Inicialmente se identificó el problema a resolver y se diseñó el estudio empírico para obtener respuestas a las preguntas de investigación asociadas. Los resultados obtenidos en este ciclo dispararon nuevos problemas a investigar que actuaron como puntos de inicio para posteriores investigaciones.

1.5 Estructura de la tesis

Esta tesis está estructurada en tres partes:

Parte I La primera parte es la introducción de la tesis, posteriormente se explica las bases tecnológicas de las tesis y se presentan algunos trabajos relacionados con los trabajos empíricos desarrollados en esta tesis y, finalmente, añade una pequeña *overview* sobre el trabajo desarrollado en esta tesis.

1 Introducción Este capítulo presenta la motivación para la tesis, los desafíos que se abordan, la contribución, la visión general de trabajo realizado, la metodología seguida y la estructura de la tesis.

2 Background Este capítulo presenta algunos antecedentes relacionados con los temas cubiertos en la tesis. Se centra en el Desarrollo Dirigido por Modelos, las Líneas de Producto Software y la Variabilidad del Software.

3 Trabajos relacionados Esta sección analiza trabajos que abordan las diferentes dimensiones consideradas en esta tesis.

4 Overview En este capítulo se describe como es posible desarrollar software utilizando los paradigmas del Desarrollo Dirigido por Modelos junto a Líneas de Producto Software y se muestran las características básicas de los estudios empíricos desarrollados.

Parte II La segunda parte de la tesis se centra en la exposición de 4 de los trabajos de investigación que forman la parte nuclear de la tesis.

5 Procesado de requisitos Este capítulo describe un experimento que estudia como el procesado de requisitos para generar modelos software se ve afectado por el nivel de detalle utilizado en las descripciones textuales de esos requisitos.

6 Usabilidad Este capítulo presenta una evaluación de usabilidad de la propuesta *Common Variability Language* aplicada a una herramienta de modelado para la generación el código de firmware de placas de inducción.

7 Comprensión de fragmentos de modelo Este capítulo describe, mediante la realización de un experimento (en el que los participantes deben comprender la variabilidad de un sistema con el fin de lograr las configuraciones deseadas del producto), las dificultades para comprender la variabilidad con fragmentos de modelo software.

8 Gestión de errores Este capítulo describe, mediante un estudio empírico, como resulta el rendimiento de los ingenieros de software cuando corrigen errores y propagan estas correcciones a otros productos configurados en el contexto de una Línea de Producto Software industrial.

Parte III La tercera parte de la tesis presenta la conclusión.

9 Conclusión y trabajos futuros Este capítulo incluye la conclusión, la recapitulación de las preguntas de investigación presentadas y sus respuestas, los próximos pasos en la investigación y las observaciones finales.

2

Background

Índice

2.1 Introducción

En este capítulo se hace una descripción sobre los dos paradigmas de desarrollo software abordados en esta tesis: Desarrollo Dirigido por Modelos y Líneas de Producto Software. Se hace referencia a sus principales características, ventajas y dificultades en su correcta aplicación. Se introducen los conceptos de Ingeniería y Arquitectura de una Línea de Productos Software y, asociado a estos conceptos, se resalta la importancia del concepto Variabilidad del Software para el correcto desarrollo de software en una Línea de Productos Software.

2.2 Desarrollo Dirigido por Modelos

El Desarrollo Dirigido de Modelos *(Model Driven Development, MDD)* es un paradigma de desarrollo software donde los modelos son la parte fundamental del desarrollo. Relacionado con el MDD surgió la Arquitectura Dirigida por Modelos *(Model Drive Architecture, MDA)* se trata de una propuesta del *Object Management Group (OMG)*, MDA es una realización específica de MDD [33]. Posteriormente apareció un nuevo paradigma donde se generaliza el enfoque de MDA denominado *Model Drive Engineering (MDE)* [34].

MDD [35] es una propuesta para el desarrollo de software en la que se atribuye a los modelos el papel principal, frente a las propuestas tradicionales basadas en lenguajes de programación, plataformas de objetos y componentes software. El propósito de MDD es tratar de reducir los costes y tiempos de desarrollo de las aplicaciones software y mejorar la calidad de los sistemas que se construyen, con independencia de la plataforma en la que el software será ejecutado y garantizando las inversiones empresariales frente a la rápida evolución de la tecnología. El objetivo es convertir, de forma automática, una especificación de un sistema en un desarrollo software completamente funcional [36].

MDD es un paradigma de desarrollo software que utiliza modelos para diseñar los sistemas a distintos niveles de abstracción, y secuencias de transformaciones de modelos para generar unos modelos a partir de otros hasta generar el código final. MDD tiene un enfoque claramente descendente, aumentando en cada fase el nivel de detalle y concreción de los modelos generados por las transformaciones. En MDD el código se puede considerar como un modelo más. En la Figura 2.1, es una simplificación de [37], se muestra un ciclo de vida de desarrollo software basado en MDD. Se pueden apreciar las distintas fases: partiendo de los requisitos se generan los modelos, en la siguiente fase se compilan los modelos, se obtiene la primera versión del código fuente, finalmente

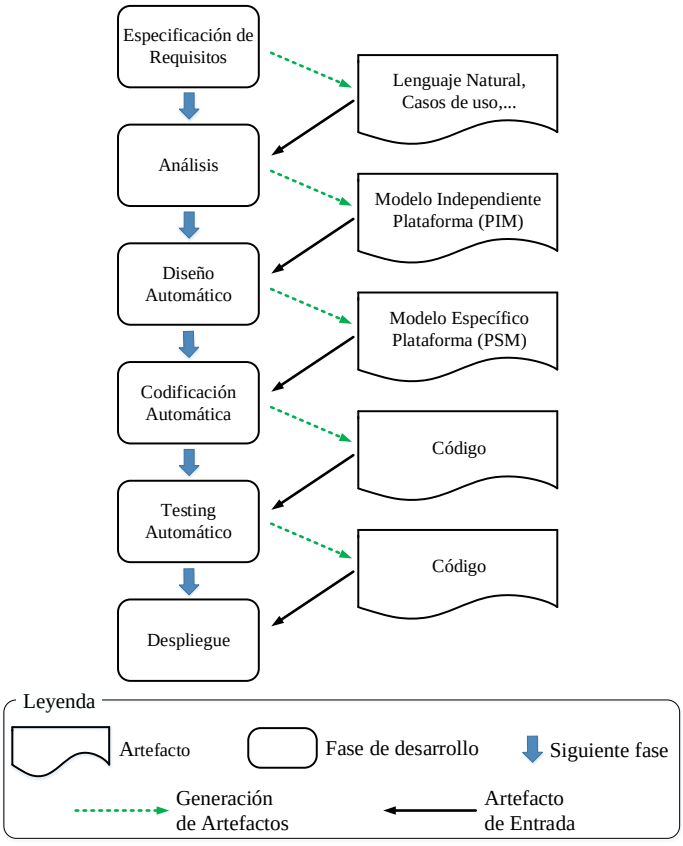


Figura 2.1: Fases y artefactos en MDD.

tras la fase de pruebas se obtiene la versión final del código fuente. Estas fases de desarrollo conforman dos "espacios" [38]:

- **Espacio del problema**: se centra en definir cual es el problema de la familia de productos software, o de un producto particular, a desarrollar.

- **Espacio de la solución**: se centra en generar los productos software para solucionar el problema.

MDD, entre otras ventajas, busca reducir las desventajas producidas por la modificación de los requisitos y la dependencia del lenguaje implementación

[39]. MDD propone que el analista debe concentrar todos sus esfuerzos en la construcción de un modelo conceptual que representa todas las características del sistema (un modelo conceptual holístico) [40]. Lo que quiere decir con "holístico" es que el modelo conceptual debe incluir todas las perspectivas relevantes de los sistemas: estructura de clase, funcionalidad e interacción [41].

Estrechamente relacionado con el MDD surgió la propuesta MDA. Dicha propuesta consiste en elaborar primero modelos de muy alto nivel partiendo de los requisitos del sistema, denominados Modelos Independientes de Plataformas *(Platform Independent Models, PIM)* y completamente independientes de las aplicaciones informáticas que los implementarán o las tecnologías usadas para desarrollarlos o implementarlos. MDA define algunos procesos para ir refinando esos modelos, particularizándolos progresivamente en Modelos Específicos de Plataforma *(Platform Specific Models, PSM)* cada vez más concretos conforme se van determinando los detalles finales.

Este planteamiento de usar modelos independientes de la tecnología presenta numerosas ventajas. Entre ellas, que las empresas pueden contar con modelos de alto nivel de sus procesos de negocio, de la información que manejan y de los servicios que ofrecen a sus clientes y los que requieren de sus proveedores, de forma independiente de la tecnología que los sustente y de las plataformas que usen en un momento dado. De esta forma, dichos modelos se convierten en los verdaderos activos de las compañías, con una vida muy superior a los sistemas de información que los implementan, ya que las tecnologías evolucionan mucho más rápido que los procesos de negocio de las empresas. El proceso de desarrollo de software se convierte por tanto en un proceso de modelado y transformación, mediante el cual el código final se genera mayormente de forma automática a partir de los modelos de alto nivel. Es una gran ventaja que la propagación de los cambios y la generación de código lo lleven a cabo las transformaciones de modelo de forma casi automática, ahorrando costes y esfuerzo, además de poder garantizar una mayor calidad.

Uno de los objetivos del MDD es la reutilización. Se debe entender reutilización no solo utilizarlo varias veces, sino utilizarlo en diferentes contextos. Una de las metas buscadas por la evolución de la programación es mejorar los mecanismos de reutilización. En algunos entornos se ha dado un paso diferente y se aplican técnicas de líneas de producto a la fabricación de software: el desarrollo de software mediante Líneas de Producto Software *(Software Product Lines, SPL)* surge con el objetivo de flexibilizar y abaratar el desarrollo de productos de software que comparten un conjunto amplio de características [42].

Existen dos enfoques bien distintos en cuanto a reutilización de software: (1) el enfoque oportunista aprovecha activos software construidos en proyectos anteriores, pero que no fueron especialmente desarrollados para ser reutilizados; (2) en un enfoque más planificado o proactivo, los activos se construyen pensando en que serán reutilizados, lo que puede significar, en el momento de su desarrollo, mayor inversión de recursos, puesto que se debe dotar al activo de suficiente generalidad. El enfoque de reutilización en SPL es más planificado que oportunista [43]: en desarrollo de software tradicional, se aprecia la posibilidad de reutilizar un componente después de haberlo desarrollado; en SPL, la reutilización es planificada, de manera que se reutilizan la mayor parte de los activos base en todos los productos de la línea.

2.3 Líneas de Producto Software

En 1976, Parnas introdujo el concepto de familias de productos software como resultado de una producción en cadena de software [44]. Posteriormente, en los primeros 90, Kang propuso la utilización de características (*features*) para dirigir la producción "en cadena" de software [45].

Los sistemas de producción de software actuales, forzados por la inmediatez de los productos y las reducciones en coste, han derivado en ocasiones en estas nuevas formas de desarrollo de software. Muchas compañías, para satisfacer las anteriores necesidades, han optado por utilizar la metodología propuesta en las SPLs. Las SPLs ofrecen una oportunidad significativa para que una empresa mejore su posición competitiva, pero, por supuesto, no es una panacea. Una incorrecta gestión en distintos aspectos, como puede ser la gestión de la Variabilidad del Software, puede ser la causa de una incorrecta utilización de las SPL [46].

2.3.1 Definición

Una SPL se define como "un conjunto de sistemas de software compartiendo características comunes y administradas que satisface las necesidades específicas de un segmento de mercado particular o misión y que son desarrolladas de forma prescrita a partir de un conjunto común de activos principales" [47]. En esta definición destaca [36]:

1. **Productos** (*"un conjunto de sistemas de software"*). Las SPLs cambian el objetivo de desarrollar una única aplicación a desarrollar un conjunto de ellas. Este cambio en el paradigma de desarrollo cambia la ingeniería en

el proceso de desarrollar software y aparece la distinción entre Ingeniería de Dominio *(Domain Engineering)* e Ingeniería de Aplicación *(Application Engineering)*. Esta distinción permite dividir el desarrollo de activos reutilizables y su variabilidad de la producción de aplicaciones en la línea de producto.

2. **Características** (*"compartiendo características comunes"*). Las características son unidades diferenciadoras mediante las cuales diferentes productos pueden ser definidos y distinguidos dentro de una SPL [48].

3. **Dominio** (*"que satisface las necesidades específicas de un segmento de mercado particular o misión"*). Una SPL es creada con un alcance determinado y dentro del ámbito de un determinado dominio. Un dominio es un cuerpo especializado de conocimiento, un área de especialización o una colección de funcionalidades relacionadas [49].

4. **Activos principales** (*"son desarrolladas de forma preescrita a partir de un conjunto común de activos principales"*). Un activo principal es "un artefacto o recurso que es utilizado en la producción de más de un producto en una SPL"[47].

5. **Plan de producción** (*"son desarrolladas de forma prescrita"*). Existe una estrategia para la creación de los productos en la SPL. El plan de producción es "una descripción de como los activos principales serán utilizados para desarrollar un producto dentro de la SPL y especifica como utilizar el plan de producción para construir los productos finales" [50].

2.3.2 Ingeniería en Líneas de Producto Software

Como se ha descrito anteriormente, una SPL es un enfoque para la reutilización estratégica y sistemática de un conjunto de activos dentro de una organización. Una SPL estará formada por una arquitectura de línea de producto, un conjunto de componentes software y un conjunto de productos derivados. El concepto ingeniería en una SPL afronta el desarrollo de familias de productos con similares características, alejándose de la producción de productos aislados.

La ingeniería de una SPL tiene tres procesos [36]: (1) creación de activos que componen los modelos (2) generación de los modelos de productos y (3) gestión de los anteriores procesos. A continuación, se describen estos tres procesos que pueden ser observados en la Figura 2.2 (la figura es una variación de [43]):

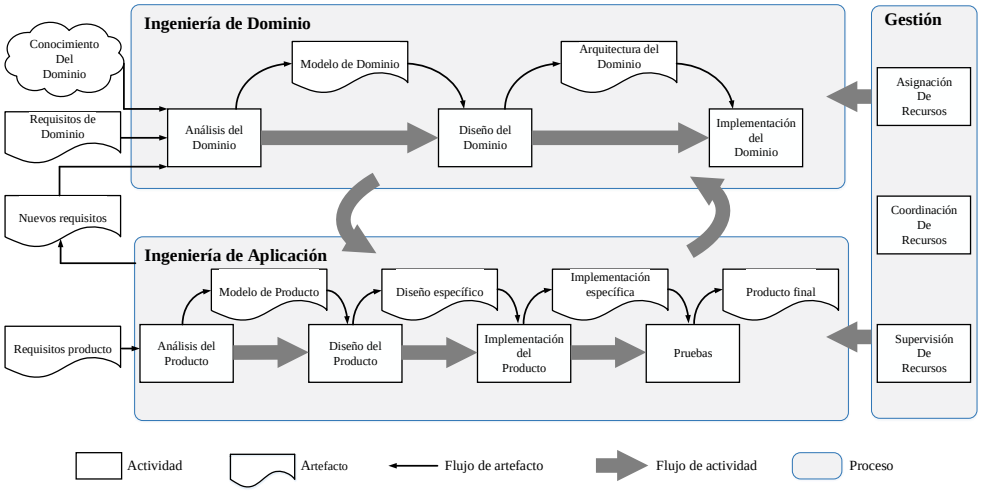


Figura 2.2: Ingeniería de una SPL.

- **Ingeniería de Dominio** *(Domain Engineering)*: se define como la actividad donde se analizan, diseñan, implementan y prueban las características comunes, que se incluirán en todos o casi todos los productos de la línea, y que se llevan a cabo mediante la construcción de componentes. Utilizando un enfoque de "diseño para la reutilización", la Ingeniería de Dominio (desarrollo de activos principales [47]) dilucida las partes comunes y variables de los miembros de la familia de productos software. La Ingeniería de Dominio se divide en análisis del dominio, diseño del dominio e implementación del dominio. Esta actividad es iterativa, ya que se re-alimentará en el futuro de la Ingeniería de Aplicación para ir evolucionando la capacidad productiva. Es decir, en todo momento, la Ingeniería de Dominio se retroalimentará de la información que recoja el equipo de aplicación durante la generación de productos [43].

  Un enfoque clásico de esta actividad implica no sólo analizar el dominio, sino diseñar los artefactos del dominio e implementarlos. Por lo tanto, a partir de los requisitos establecidos previamente se realizará el diseño genérico de la SPL que producirá eventualmente una Arquitectura de Dominio. A continuación, se implementarán las partes de código que soporten esta arquitectura. Estos activos serán reutilizados durante la construcción de productos. Sin embargo, un enfoque más reciente implica concentrarse en tres actividades: estudiar el alcance de la línea de pro-

ducto, desarrollar los activos principales y crear un plan de producción que articule toda la capacidad productiva, y que sirva de guía durante el proceso de Ingeniería de Dominio [47].

- **Ingeniería de Aplicación** *(Application Engineering)*: se analizan, diseñan e implementan y prueban los productos concretos, incorporando a cada uno de ellos las características que les correspondan y que procedan del nivel de Ingeniería de Dominio.

  Se partirá de unos requisitos concretos que deben de estar alineados de alguna forma con los requisitos de la línea de producto, aunque puede haber una pequeña parte nueva y que hay que desarrollar para un determinado producto. El estudio de estos requisitos producirá un análisis detallado con la viabilidad de su construcción, así como la cantidad de activos que podemos reutilizar. El siguiente paso consistirá en el diseño específico del sistema (con las partes comunes junto con las nuevas). A continuación, se utilizará el configurador de producto que (en base a los requisitos seleccionados) ensamblará los activos comunes y variables para producir el producto final que se entrega al cliente [43]. La Ingeniería de Aplicación se divide en análisis de la aplicación, diseño de la aplicación, implementación de la aplicación y pruebas.

- **Gestión** *(Management)*: en este proceso se abordan específicamente los temas referentes a la organización de las actividades anteriores. Sus responsabilidades son dotar de recursos, coordinar y supervisar las actividades las Ingeniería de Dominio y de Aplicación.

  Esta actividad juega un papel fundamental en el éxito de la línea de producto, ya que es no sólo la encargada de asignar recursos, sino de coordinarlos y supervisarlos. Esta gestión se realiza tanto en el nivel técnico (gestión de proyectos concretos) como en el nivel organizativo (estructura organizativa adecuada a los objetivos propuestos). Los artefactos de gestión creados como planificaciones también forman parte de los activos principales.

En el desarrollo de activos que compondrán los modelos se deberán considerar cuestiones como necesidades de mercado, dificultades con el modelado de los activos y la especificación de la variabilidad para alcanzar las calidades y objetivos de los productos software a conseguir. El proceso de la generación de los modelos de productos software consiste en una serie de actividades para analizar los objetivos y contextos de uso del producto, analizar los requisitos del producto y configurar el producto (es decir, seleccionar los activos apropia-

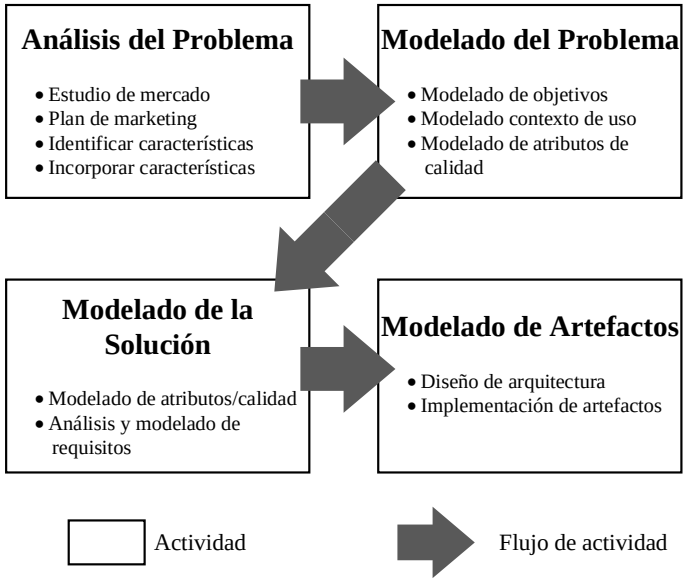


Figura 2.3: Actividades para la derivación de un producto en una SPL.

dos acorde a las especificación de variabilidad), seleccionar una arquitectura y componentes apropiados y hacer las adaptaciones necesarias a la arquitectura y componentes seleccionados para la generación y desarrollo del producto.

En determinadas ocasiones, la SPL se crea partiendo de un portfolio de productos ya existentes. En estas condiciones, se debe considerar en el ciclo de ingeniería de la SPL la creación de activos y una especificación de variabilidad que permitan crear los productos en el portfolio ya existentes además de sentar las bases para el desarrollo futuro. Por lo tanto, el inicio de una SPL puede venir marcado por un portfolio existente, por un plan de marketing o por una solución híbrida.

En [46] se propone un ciclo de ingeniería que dota de un mayor protagonismo a actividades como estudio de mercado o plan de marketing. El citado ciclo de ingeniería se puede observar en la Figura 2.3, se distingue:

- **Análisis del problema**: Se inicia haciendo un estudio de mercado y un plan de marketing. Se comenzará a desarrollar los activos que componen la SPL, la capacidad de re-utilización y el conocimiento del contexto de utilización. Para "vender" los productos deben ser vendidos por lo que se deberá tener en consideración las necesidades de los usuarios. La variabi-

lidad servirá como mecanismo para configurar productos que cumplan las necesidades de mercado y el plan de marketing de una manera óptima. El plan de marketing incluirá un análisis de mercado, y una estrategia de marketing y oportunidades en el mercado. El análisis de mercado incluye, para cada segmento de mercado, una evaluación de las necesidades, de los usuarios potenciales, las limitaciones culturales y legales, el tiempo de comercialización y el precio-distancia. La estrategia de marketing inicialmente incluye un esquema de los métodos de entrega, información sobre cómo se entregarán los productos a los clientes y otros consideraciones empresariales.

Una vez que se ha definido la parte del plan de marketing, es importante identificar las características de los productos en una SPL y desarrollar un plan para incorporar las características. Los productos tendrán características funcionales y no funcionales. Las características funcionales incluyen servicios, que a menudo se consideran unidades comercializables o unidades de incremento en una SPL, u operaciones, que son funciones internas de los productos que se necesitan para proporcionar servicios. Las características no funcionales incluyen al usuario final: presentación, calidad, precio, etc. Por ejemplo, la tolerancia a fallos sería un atributo de calidad.

Es importante como son "entregados" los productos a los clientes y usuarios, y cómo serán instalados y mantenidos. Esta fase debe encontrarse alineada con la estrategia de marketing. Una de las consideraciones a tener en cuenta son los perfiles de las personas que trabajan con los productos de la SPL.

- **Modelado del problema**: Las características del modelado de problema representan el contexto concreto de los productos de una SPL, es decir, las causas que dirigen la selección de arquitecturas específicas, los algoritmos de implementación o las técnicas de implementación; estas características ayudan a comprender los problemas del mundo real que la SPL debe abordar. El espacio del problema debe capturar información sobre porqué es la SPL requerida en el mercado, en qué condiciones se utiliza una determinada configuración o cuáles son las cualidades requeridas de la SPL. El espacio del problema se puede dividir en tres puntos de vista disjuntos: objetivos/metas, contexto de uso y calidad esperada.

  El proceso de modelado se inicia a partir del contexto de uso y los objetivos de los productos de la SPL. Alguna consideraciones para conocer los objetivos de una SPL son: (1) problemas del mundo real que los produc-

tos de la SPL abordan, (2) otras SPLs que consideran los problemas de forma similar pero con distintos enfoques, (3) beneficios potenciales que pueden ser considerados de otras SPL, y (4) propiedades no funcionales de la SPL (por ejemplo, comodidad, eficiencia, etc.) que deben lograrse. La siguiente etapa es estudiar el contexto de uso: para distintos contextos de uso la SPL deberá sufrir la correspondiente adaptación.

Es fundamental considerar el dominio en el que va a ser utilizada la SPL [51]. Así, en en el sector de la automoción cobra más importancia los aspectos de implementación y aseguramiento de la calidad *(quality assurance)*; por contra, en el ámbito de las finanzas tiene más relevancia la arquitectura y la implementación de la SPL. Diferentes dominios tienen diferentes estándares adaptados que les hacen tener distintas prioridades a la hora de desarrollar software mediante una SPL.

En la siguiente etapa se analizan los requisitos de calidad de una SPL y se modelan como características de atributos de calidad. Con este fin se pueden utilizar la especificación de requisitos de software, el documento de requisitos de calidad, etc., como entradas de esta actividad. Los objetivos citados anteriormente y el contexto de uso tienen implicaciones en los atributos de calidad.

- **Modelado de la solución**: se debe analizar el espacio de la solución para satisfacer los requisitos capturados por el espacio del problema. Después de finalizar el modelado para los espacios del problema y la solución, se establecerán relaciones entre el problema y los modelos del espacio de la solución. Para cada atributo de calidad del espacio del problema deberá existir su correspondiente atributo de calidad en el espacio de la solución.

  Durante el análisis de requisitos de la SPL se capturan las funcionalidades que los productos de la SPL deben proporcionar. Se capturan utilizando un conjunto de modelos como modelos de caso de uso, procesado de requisitos en lenguaje natural, etc. Una vez que los atributos de calidad y los requisitos de la SPL son desarrollados, esta información se utilizará para refinar el plan de marketing.

  La gestión de los requisitos, en particular los requisitos no funcionales, siguen siendo un tema no abordado en profundidad, impidiendo que las SPLs alcancen todo su potencial. Una óptima gestión de los requisitos mejorará la comunicación entre los diferentes interesados en el producto, permitirá impactar positivamente en la interacción analista-cliente respecto a la comprensión y definición del producto a construir [52].

- **Modelado de artefactos**: Tras establecer los modelos en los espacios del problema y de la solución y las relaciones entre ellos, se deben desarrollar los artefactos de la SPL basados en estos modelos. En esta etapa se implementarán los artefactos de la SPL, esto incluye la arquitectura de la SPL y la implementación de los elementos que la componen.

  En el diseño de la arquitectura se establecen los componentes conceptuales con un alto nivel de abstracción y sus dependencias, para ello el plan de marketing es un elemento clave. A partir de los componentes conceptuales, éstos deben ser refinados en componentes concretos. El diseño de los componentes está formado por especificaciones de componentes y sus relaciones.

2.3.3 Arquitectura en Líneas de Producto Software

Un aspecto fundamental de una SPL es la arquitectura subyacente, entendiendo como arquitectura en una Línea de Productos Software como la infraestructura y los componentes de la SPL que permiten desarrollar todos los productos requeridos. Como se ilustra en la Figura 2.4, podemos identificar una trayectoria de evolución típica para una SPL partiendo desde el alcance de los elementos compartidos en la línea de productos frente a la cantidad de funcionalidad cubierta en el código específico. El modelo utiliza cuatro etapas [46]:

Infraestructura estandarizada: partiendo de un conjunto de productos independientes, el primer paso para una organización es normalizar el software correspondiente a esos productos. Típicamente, estos componentes de software son de naturaleza infraestructural (puede tratarse de un conjunto de productos finales ya existentes). En esta etapa se busca estandarizar la infraestructura y hacer que cada producto se base en el mismo conjunto de componentes, puede lograr beneficios significativos apoyándose en la reutilización.

Plataforma: La segunda etapa es la formación de una plataforma en la parte superior de la infraestructura. La plataforma se refiere aquí a una capa de funcionalidad que es común a todos los productos dentro de la línea de productos.

Línea de Productos Software: Una vez establecido el valor de compartir software entre productos, habrá una tendencia a poner más funcionalidad en los componentes compartidos. En esta etapa, también la funcionalidad que es compartida por un subconjunto de los productos forma parte de la línea de productos.

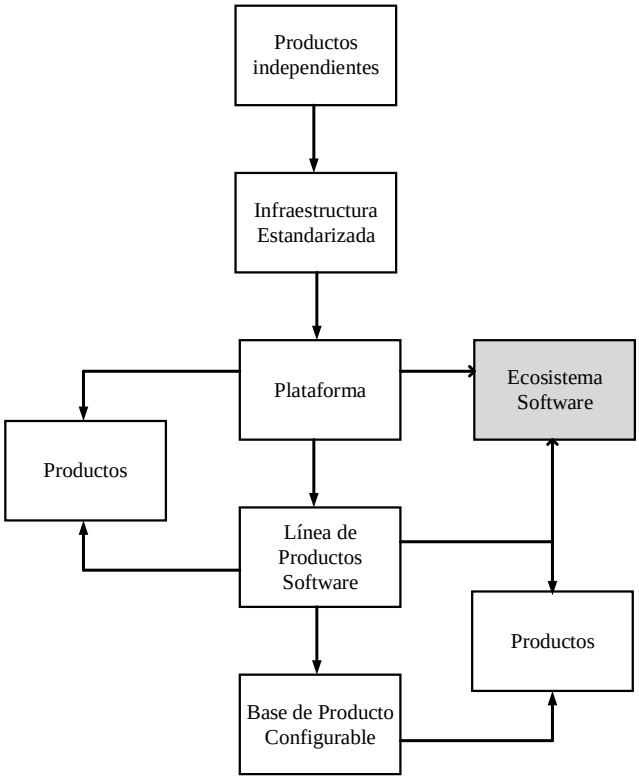


Figura 2.4: Arquitectura de una SPL.

Base de Producto Configurable: la etapa más avanzada es donde un conjunto completo de productos puede ser derivado automáticamente desde los activos que comparten características. Ya no hay necesidad de equipos de desarrollo específicos del producto, ya que los productos pueden derivarse automáticamente.

En las dos últimas etapas descritas anteriormente (Líneas de Producto Software y Base de Producto Configurable) aparece el concepto de Variabilidad del Software *(Software Variability)*. La Variabilidad del Software mediante los puntos de variación permitirá establecer comportamiento diferente, que se plasmará en la creación de diferentes productos software, en función de las posibilidades permitidas en el diseño de la SPL y de la elección en el momento de derivar el producto.

Es importante considerar el espacio temporal cuando una característica de un producto es eliminada, añadida o modificada del sistema software. Considerando las distintas etapas por las que puede pasar una caracterísitca en el ciclo de vida de una SPL se consideran 4 fases: desarrollo de activos, desarrollo de productos, preoperación (instalación) y operación (tiempo de ejecución).

El enfoque de las SPLs busca plasmar una economía de ámbito por contraposición a una economía de escala. La economía de escala busca producir de manera masiva un mismo diseño de un producto, la economía de ámbito persigue generar productos similares pero con diferencias en su diseño [53]. Para alcanzar este objetivo se comparten procesos, herramientas y materiales creando productos muy similares con pequeñas diferencias.

2.4 Variabilidad del Software

Como se ha comentado anteriormente, una parte fundamental de las SPLs es la gestión de la Variabilidad del Software. Se define Variablidad del Software como la capacidad de un sistema de software o artefacto de ser ampliado, modificado, personalizado o configurado para su uso en un contexto particular [46].

Las causas del incremento de la importancia de la variabilidad son: (1) el movimiento de la variabilidad del hardware hacia al software y (2) el intento de los ingenieros de software de retrasar sus decisiones de diseño a las últimas fases del desarrollo software. Referido a la segunda causa, la cantidad de software necesario para la generación de productos ha aumentado paulatinamente, actualmente cualquier producto cotidiano necesita de un desarrollo software. Por otra parte, existe una presión constante para aumentar el número de productos de software puestos en el mercado con el fin de servir mejor a los distintos segmentos de mercado.

2.4.1 Gestión de la Variabilidad del Software

Una forma de explotar la similitud entre los diferentes productos e implementar las diferencias entre los productos es mediante la Variabilidad del Software. La familia de productos de software de una SPL debe incorporar variabilidad para soportar posibles cambios en requisitos y creación de nuevos productos. Por estas razones, las decisiones de diseño se dejan abiertas y se determinan en etapas lo más lejanas posibles en el desarrollo del producto, por ejemplo, cuando se construye un producto particular o durante el tiempo de ejecución

de un producto particular: esta capacidad viene soportada por la Variabilidad del Software.

La gestión de variabilidad involucra todas las actividades relacionadas con la variabilidad a lo largo de todo el ciclo de vida de una SPL, desde la especificación de requisitos, modelado e implementación, hasta el testing de los productos derivados. La variabilidad se hace explícita a través de los puntos de variación *(variation points)*. Un punto de variación permite al ingeniero de software retrasar una decisión de diseño y puede, posteriormente, ser enlazado con una variación particular. Dado un punto de variación, pueden ser abierto (se pueden añadir más variantes) o cerrado (no se pueden añadir más variantes) [36]. Durante la fase de Ingeniería de Dominio se pueden introducir nuevos puntos de variación, durante la fase de Ingeniería de Aplicación los puntos de variación serán enlazados con las variantes seleccionadas.

En [54] se consideran las siguientes actividades relacionadas con la gestión de la Variabilidad del Software: definición de la variabilidad y los posibles puntos de variación, gestión de los componentes variables y resolución de la variabilidad en el momento de realizar el proceso de derivación de productos. En [55] se considera que la gestión de la variabilidad abarca las actividades de representar explícitamente la variabilidad de los artefactos de software a lo largo de su ciclo de vida, gestionar las dependencias entre diferentes variabilidades y apoyar la instanciación de las variaciones posibles. Una correcta gestión de la variabilidad es la clave del éxito o fracaso de una SPL [51].

2.4.2 Propuestas para el modelado de la variabilidad

Existen diferentes propuestas para el modelado de la variabilidad que son independientes del lenguaje en el que los productos software van a ser desarrollados. Estas propuestas incluyen *Common Variability Language (CVL)* [56], *Feature-Oriented Domain Analysis (FODA)* [45], *Orthogonal Variability Model (OVM)* [1] o *COVAMOF* [57]:

- **CVL** es una propuesta del Comité de Arquitectura del *Object Management Group (OMG)* como estándar para el modelado de la variabilidad. CVL expresa la variabilidad a través de modelos en términos de modelos de fragmentos denominados *Placements Fragments* (puntos de variación) y *Replacements Fragments* (variantes). La materialización de productos es realizada mediante la sustituciones entre un modelo base y fragmentos de una biblioteca.

- **FODA** es una propuesta donde un modelo está compuesto de dos elementos: las características *(features)* y las relaciones entre ellas. Las características se organizan conforme a una estructura jerárquica.
- **OVM** es una metodología para el modelado de la variabilidad en SPL, propone un modelo de variabilidad separado de los artefactos de la línea de producto. Es decir, un modelo OVM solamente documenta la variabilidad de la SPL porque la parte común entre las aplicaciones es documentada en modelos conceptuales tradicionales como modelo de requisitos, diagrama de componentes, etc.
- **COVAMOF** es un framework para el modelado de SPL que modela la variabilidad en términos de puntos de variación y dependencias en diferentes niveles de abstracción. En esta propuesta, la jerarquía de la SPL es modelada en tres niveles de abstracción: características, arquitectura e implementación de componentes. La variabilidad es modelada en todos los niveles, desde las características hasta la implementación de componentes. Un punto de variación en un nivel de abstracción puede realizar la variabilidad de un otro punto de variación del nivel superior.

Las anteriores aproximaciones se fundamentan en la idea de sobreponer variantes en un modelo, donde cada característica está relacionada con un fragmento de modelo que es expresado en los mismos términos que el producto software [58].

2.4.3 Alcance de la SPL

El alcance de la SPL *(SPL scope)* influye directamente en la variabilidad. Dentro del alcance de la SPL se deben considerar cuestiones como los contextos para los que se van a desarrollar productos o el catálogo de productos. Existen razones económicas, comerciales o técnicas para delimitar el alcance de los productos en una SPL y, como consecuencia, el alcance de la especificación de la variabilidad. Es muy importante un buen diseño de portfolio de productos y el conocimiento de la evolución futura de los mismos para que la variabilidad implementada sea lo suficientemente flexible como para soportar el portfolio inicial de productos y futuras variaciones de manera controlada. Si las necesidades de mercado, objetivos futuros, etc. no dirigen las decisiones relativas al alcance de la SPL, puede favorecer que el alcance de la SPL se base en cuestiones más técnicas, como la extensibilidad de modelos de variabilidad para soportar la evolución y las tareas de derivación [46]. Se pueden considerar los siguientes aspectos relativos al alcance de una SPL:

- Alcance del dominio: tiene como objetivo definir los límites del dominio donde se utilizarán artefactos y productos.
- Alcance del producto: define los productos que serán diseñados.
- Alcance de los activos: se centra en la identificación de los activos reutilizables que se emplearán en la construcción de los futuros productos.

Es fundamental delimitar correctamente los anteriores alcances para conocer que posibilidades para crear productos tiene la SPL. Alineado con esta afirmación en [51] se concluye que una de las mayores dificultades en entornos reales al trabajar con SPLs es la definición del portfolio de productos. Otras dificultades declaradas son: identificar los productos afectados por un cambio en los requisitos o la corrección de un error, implementación del mismo cambio en varias versiones de un mismo producto, aseguramiento de la calidad debido a las dependencias y el esfuerzo requerido para generar nuevas versiones de productos similares.

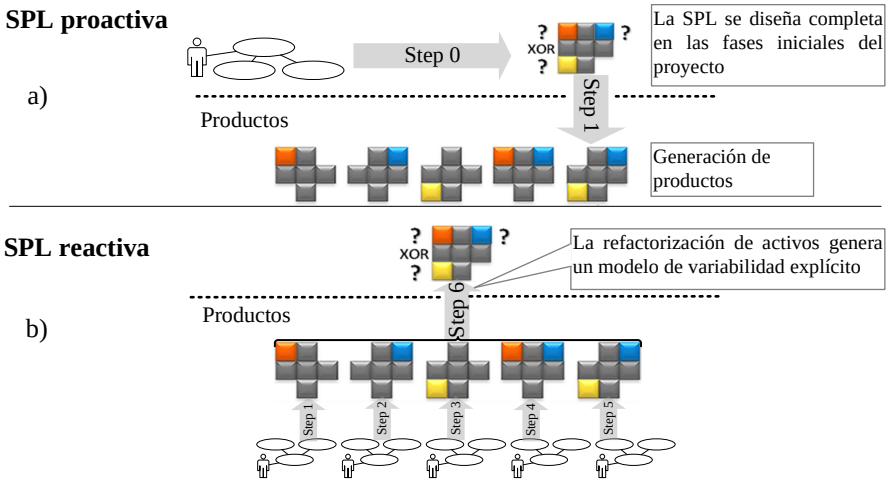


Figura 2.5: a) Enfoque proactivo en una SPL. b) Enfoque reactivo en una SPL.

Considerando la variable temporal en el desarrollo de una SPL aparecen dos enfoques en la utilización de la misma:

- Los enfoques proactivos intentan delimitar el alcance completo de los productos cuando se lanza una línea de productos desde cero. La Figura 2.5 a) muestra este enfoque.

- Los enfoques reactivos se ocupan más del alcance de nuevos productos a medida que la línea de productos evoluciona y cuando aparecen nuevos requisitos. En la Figura 2.5 b) aparece representado el enfoque reactivo.

Un enfoque intermedio es el denominado modelo extractivo. Este modelo parte de una o varias aplicaciones ya existentes para generalizarlas hacia una SPL. Tal vez sea éste el enfoque más realista en la medida en que la mayoría de las organizaciones raramente disponen de los recursos para acometer desde cero la SPL, y sin embargo, sí disponen de aplicaciones cuyo farragoso mantenimiento les impulsa a embarcarse en las SPLs [43].

Una vez que los activos reutilizables, productos configurados de la SPL y la especificación de la variabilidad estén bien delimitados en el modelo de variabilidad, la SPL estará lista para desarrollar actividades de configuración y realización de producto, es decir, para producir los activos reutilizables y nuevos productos. El alcance de la especificación de la variabilidad no sólo está limitado por las opciones configurables disponibles sino también por las reglas de restricción y dependencia que determinarán qué productos están permitidos o no dentro del alcance. Tales restricciones estarán descritas e implementadas como parte de la especificación de la variabilidad.

2.4.4 Fases de la Variabilidad del Software

En aquellos sistemas de desarrollo software que utilicen un enfoque basado en variabilidad deberán considerar las distintas fases que se suceden en un desarrollo que utilice este enfoque. En el desarrollo, desde el punto de vista de la Variabilidad del Software, se tienen las siguientes fases [46]:

- **Especificación de requisitos**. Durante esta etapa, el equipo pretende maximizar la claridad de lo que se va a construir. Puede haber requisitos explícitos para la variabilidad, pero frecuentemente las decisiones se toman como parte del proceso de especificación de requisitos que reduce la variabilidad requerida.
- **Diseño arquitectónico**. La división del sistema en sus componentes principales es la etapa en la que los primeros puntos de variación pueden ser introducidos.
- **Diseño detallado**. Una vez que el sistema se ha diseñado globalmente, pueden introducirse cambios al diseño de los componentes específicos. A este nivel, puede modificarse la especificación de la variabilidad.

- **Desarrollo de software**. La variabilidad definida en el sistema se implementa a nivel de código.
- **Compilación**. En esta etapa, se resuelven los primeros puntos de variación.
- **Enlazado**. Durante el enlazado, los puntos de variación son enlazados con variaciones específicas. La mayoría de los enlaces durante la compilación y durante esta fase son permanentes y no se podrán cambiar en etapas posteriores.
- **Instalación/configuración**. El sistema de software está instalado y configurado, se seleccionaran los productos a desarrollar y posteriormente se procederá a la instalación conforme a los ajustes seleccionados.
- **Puesta en marcha**. Durante el arranque del sistema, se pueden actualizar puntos de variación que pueden ser enlazados. En ocasiones, se utilizan archivos de configuración que se leen durante la puesta en marcha para resolver ciertos puntos de variación.
- **Tiempo de ejecución**. En la última etapa, deberán ser resueltos los puntos de variación que queden pendientes.

En alguna de las fases anteriores será introducido un punto de variación. Normalmente se hará en una de las primeras etapas, hay técnicas que permiten la introducción tardía de la especificación de la variabilidad. La siguiente etapa, tras introducir los puntos de variación, es añadir alguna de las variantes al punto de variación. Estas variantes capturan las diferencias de comportamiento que se requieren del sistema. La tercera etapa es el enlace del activo que varía con el punto de variación.

En función del momento elegido para especificar la variabilidad distintas técnicas de implementación de variabilidad pueden desarrollarse. Las técnicas de realización de la variabilidad están íntimamente ligadas a la forma y al momento en que se pueden desplegar los productos, y se pueden elegir varias alternativas para seleccionar la mejor estrategia de configuración y despliegue.

En el espacio de la solución, la variabilidad se describe mediante la especificación de la variabilidad y las posibles variaciones. Por lo tanto, la creación de los productos de software implica conocer en un momento determinado del proceso de desarrollo de software, las opciones configurables definidas en la arquitectura e implementadas en los activos y productos básicos. Es importante conocer las siguientes definiciones [46]:

- Técnica de realización de variabilidad: es la forma en que se realiza cualquiera de los miembros de la familia de productos software. La realización de productos de software concretos implica conocer la especificación de la variabilidad.
- Derivación del producto: es una etapa en el ciclo de vida de la Línea de Productos Software donde los productos de software se convierten en el resultado de un proceso de selección y configuración de las opciones de diseño definidas en el modelo de variabilidad.

La configuración del producto software puede establecerse al comienzo del proceso de derivación, o también puede ejecutarse en una etapa muy tardía si un producto tiene que ser reconfigurado una vez desplegado. A veces, se hace la configuración para seleccionar las opciones de variables que se incluirán en un producto antes de que la variabilidad se realice, mientras que en otros casos, un proceso de reconfiguración ocurrirá en la etapa final de la SPL o durante la ejecución del sistema. Esta situación aparece en las denominadas Líneas de Producto Software Dinámicas *(Dynamic Software Product Lines, DSPL)*.

En otras situaciones, la configuración del producto y la realización de la variabilidad también se superponen en el tiempo: en este caso, se selecciona la variabilidad al tiempo que se realiza. Estas posibilidades quedan reflejadas en la Figura 2.6.

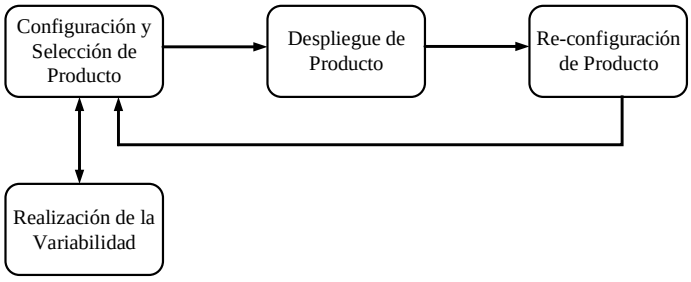


Figura 2.6: Actividades en la derivación de productos.

La Figura 2.6 describe las principales actividades de un proceso de derivación de un producto software en una SPL. Una vez que los requisitos son procesados y se define la variabilidad un nuevo producto, un proceso de selección y configuración del producto selecciona las variantes correctas. Con la técnica de implementación de variabilidad particular y cumpliendo las especificaciones de variabilidad se configura el producto. Una vez que se ha realizado la variabilidad y el producto ya está configurado, instalado y desplegado, cualquier

reconfiguración en etapas posteriores al despliegue o del tiempo de ejecución puede conducir a una nueva selección y configuración de la variabilidad. En ese caso, el producto reconfigurado o el nuevo producto puede o debe desplegarse nuevamente, mientras que en otras situaciones no se requiere un nuevo despliegue.

La realización de la variabilidad se puede desarrollar en distintas etapas:

- Etapa de diseño: la especificación de variabilidad y variantes se realiza a nivel de arquitectura. Por lo tanto, los pasos para realizar la variabilidad en el tiempo de diseño son: (1) selección de variantes y especificación de la variabilidad definidos en la arquitectura; (2) selección de los valores permitidos; (3) representación de la arquitectura del producto instanciando las variantes con valores apropiados para cada producto. En el momento del diseño, el ingeniero de software selecciona las variaciones permitidas y deseadas para realizar la construcción del sistema y posteriormente deriva la arquitectura del producto para ese sistema.

- Etapa pre-despliegue: los productos pueden ser configurados, instalados y desplegados por el "cliente". Cuando la variabilidad de los productos las realiza el cliente, las variantes y la especificación de la variabilidad pueden ser instanciados en diferentes tiempos, dependiendo de cómo se construye y configura el producto. La realización de la variabilidad depende de cómo se implemente. Por otra parte, la idea de la variabilidad escalonada fomenta la composición de las características que se pueden agregar o quitar para derivar diferentes configuraciones del producto. El ingeniero de software selecciona y anula la selección de las características de cada nueva versión del producto. Por lo tanto, la configuración se convierte en un proceso donde las personas toman las decisiones de configuración correctas en diferentes etapas. La configuración escalonada constituye un refuerzo escalonado del modelo de variabilidad. En algunos casos, este refinamiento conduce a una especialización donde se seleccionan soluciones durante el proceso de configuración del producto y se obtiene un modelo de características especializado.

- Etapa post-despliegue: la configuración de la variabilidad y la realización del producto en el cliente (actividades posteriores al despliegue) a menudo implica procesos de instalación y post-despliegue donde los productos son configurados y desplegados en nombre de un conjunto de opciones configurables para un entorno específico o considerando las preferencias del usuario. Un ejemplo es la configuración de los productos en el primer arranque del sistema. Otras situaciones pueden referirse a la carga diná-

mica de módulos de software o bibliotecas que afectan a la configuración del sistema, ya que en algunos casos, el sistema debe reiniciarse. Finalmente, durante la ejecución del sistema, la selección de variantes ocurre mientras el sistema está en ejecución. La capacidad de seleccionar una nueva variante se considera una realización de variabilidad en tiempo de ejecución pura.

Como ya ha sido comentado anteriormente, la adopción de un sistema de desarrollo software basado en el paradigma de la reutilización de activos software no es algo trivial. En [51] se resaltan, considerando distintas fases de ingeniería de la SPL, algunas consideraciones para la implantación de una SPL:

- El elevado consumo de recursos para generar especificaciones individuales para los productos en la fase correspondiente a la ingeniería de requisitos.
- Ausencia de trazabilidad entre artefactos de aseguramiento de la calidad y sus especificaciones.
- Insuficiente entrenamiento de las personas involucradas en la SPL en todas las fases.

2.5 Conclusiones

El objetivo de este capítulo es presentar los paradigmas de desarrollo software que van a ser objeto de estudio, mediante varios estudios empíricos, en el desarrollo de esta tesis. El Desarrollo Dirigido por Modelos (MDD) es un paradigma para desarrollo de software basado en el modelado, a diferencia del desarrollo de software tradicional que se fundamenta en la creación de código. Desde un nivel de abstracción más elevado que el utilizado en la generación de software se crean modelos que, tras varias transformaciones, permiten generar el código que implemente la funcionalidad requerida. Este paradigma permite incrementar la productividad, optimizar el *know-how* y facilita las tareas repetitivas.

Las Líneas de Producto Software (SPL) son un paradigma de desarrollo software para generar familias de productos software fundamentadas en el concepto de "producción en cadena". El objetivo cambia de generar un producto individual a generar activos reutilizables que se utilizan para generar familias de productos. Uno de los pilares de las SPL es la Variabiliad del Software, permite que un sistema software pueda ser personalizado para su uso en un contexto particular.

3

Trabajos Relacionados

Índice

3.1 Introducción

En este capítulo se muestra una recapitulación de los trabajos relacionados con las investigaciones realizadas para el estudio de cada una de las dimensiones tratadas en esta tesis. Cada uno de los apartados, ordenado por dimensión, consta con una breve descripción de cada uno de los trabajos, una comparación explícita con el trabajo realizado en esta tesis y una tabla donde se muestra el objetivo de cada uno de los trabajos, las métricas utilizadas y los sujetos participantes en el estudio cuando proceda. Finalmente, se realiza un análisis sobre como han sido abordadas las cuatro dimensiones tratadas en esta tesis considerando diferentes contextos de desarrollo software MDD y SPL o en entornos industriales.

3.2 Trabajos relacionados: procesado de requisitos

El proceso para obtener, informar y usar requisitos como entrada para los siguientes pasos en el proceso de desarrollo de software tiene una gran dependencia de factores humanos. Según Ahmed [59], existen habilidades no técnicas que pueden afectar el proceso de obtención de requisitos. Esta fuerte dependencia de las habilidades humanas y del tipo de analista puede llevar a que algunas especificaciones de los requisitos sean ambiguas en algunos casos. De acuerdo con la revisión bibliográfica realizada por Bano [60], el 81 % de los trabajos se centran en detectar la ambigüedad en las especificaciones textuales. Bano también afirma que hay una falta de evaluación empírica de las técnicas de procesado del lenguaje natural para abordar la ambigüedad en los requisitos.

La Tabla 3.1 muestra una relación de estudios previos realizados en el campo de la interpretación de especificaciones de requisitos textuales. La primera columna indica la referencia bibliográfica del trabajo, la segunda columna indica el año de publicación, en la tercera columna se puede leer el objetivo principal del trabajo referenciado, en la cuarta columna quedan reflejadas las métricas utilizadas en cada uno de los trabajos y, finalmente, en la quinta columna se muestra si en el trabajo participaron sujetos y la naturaleza de los mismos.

Varios trabajos se han ocupado de las especificaciones de requisitos y su procesado para la construcción de modelos. Estos trabajos tienen como objetivo extraer modelos conceptuales de textos con requisitos de lenguaje natural. Un ejemplo de estos trabajos es [71], los autores proponen derivar automáticamente modelos conceptuales a partir de historias de usuario que están escritas en lenguaje natural. El enfoque se basa en un algoritmo que combina varias heu-

Paper	Año	Objetivo	Métrica	Sujetos
[61]	2002	Evaluación calidad de requisitos	Expresividad, consistencia e integridad	No procede
[62]	2004	Análisis de grandes especificaciones de requisitos textuales	Relaciones entre conceptos, organización aspecto-temporal y causalidad	No procede
[63]	2009	Análisis de extracción de términos desde una descripción textual	Heurísticas	No procede
[64]	2010	Evaluar calidad de especificación de requisitos funcionales	Eficacia real, integridad, semántica, errores de granularidad y eficacia percibida	Estudiantes
[65]	2013	Evaluación calidad de requisitos	Indicadores: morfológicos, léxicos, analíticos y relacionales	No procede
[66]	2013	Comparar la comprensibilidad de los modelos de requisitos	Nivel de comprensión, esfuerzo y productividad	Estudiantes
[67]	2014	Rendimiento de la transformación de texto a modelos	Precisión, *recall* y sobre-especificación	Estudiantes
[68]	2014	Evalúa la herramienta CAR para medir integridad de requisitos	Relevancia de términos y Relevancia de relaciones	Autores del trabajo
[69]	2015	Evaluar el impacto de los defectos en la especificación de requisitos	Calidad semántica, calidad pragmática y conocimiento del dominio	Estudiantes
		Identificar la calidad de las especificaciones de los requisitos del software	Integridad y grado de utilización	Profesionales industria
[70]	2015	Evaluar y comparar las técnicas de especificación de requisitos	Sintáctica, semántica, capacidad de comunicación y usabilidad	Estudiantes
[71]	2016	Precisión de la transformación de texto a modelos	Precisión y *recall*	2 expertos

Tabla 3.1: Trabajos relacionados con la dimensión procesado de requisitos por orden cronológico.

rísticas y se evalúa a través de dos estudios de casos. Las métricas utilizadas en la evaluación son precisión y *recall* para comparar la exactitud de la técnica propuesta con respecto a un etiquetado manual, dos expertos intervienen para evaluar el resultado obtenido. En [67] también se propone una transformación automática de las especificaciones funcionales en lenguaje natural a modelos conceptuales. La propuesta se basa en el análisis de construcciones gramaticales. El resultado de la transformación es la construcción de un diagrama entidad-relación con anotaciones. La propuesta se evalúa a través de un estudio de caso que compara el modelo construido a través de las transformaciones con modelos construidos por estudiantes y modelos publicados en literatura cinetífica. El trabajo evalúa el rendimiento mediante tres métricas: precisión, *recall* y sobre-especificación (*over-specification*). En [68] se realizó una evaluación de una herramienta (denominada CAR) que admite una definición textual de requisitos. La evaluación se realizó utilizando la integridad métrica, en la que los experimentos comparan la integridad de los requisitos mediante el uso de CAR versus la no utilización de herramienta alguna: la herramienta mide la integridad de la especificación de requisitos expresado en lenguaje natural. Las métricas utilizadas son relevancia de términos y relevancia de relaciones. Los autores de ese documento también son los sujetos del estudio.

Hay trabajos que se centran en detectar la ambigüedad en las especificaciones textuales, como en [61]. Estos autores definieron un conjunto de métricas para realizar una evaluación de calidad de los especificaciones de requisitos basados en plantillas de casos de uso. Estas métricas se basan en tres variables que analizan las especificaciones de los requisitos a través de técnicas lingüísticas: expresividad, consistencia e integridad. Las técnicas lingüísticas no son suficientes para abordar por completo los aspectos relacionados con la corrección y la coherencia de los requisitos. En [65] se definieron indicadores para medir la calidad en los requisitos textuales, así como una herramienta que calcula medidas de calidad de forma totalmente automática. El enfoque tiene como objetivo mejorar la calidad de los requisitos y mejorar las habilidades de escritura de los analistas. Miden la calidad con indicadores: morfológicos, léxicos, analíticos y relacionales. En el trabajo se muestra la opinión de ingenieros que han utilizado la herramienta, no se desarrolla una evaluación formal. Los autores en [69] estudiaron en qué medida las especificaciones de requisitos de software se crean y usan en la práctica, así como el grado en que la calidad de dichas especificaciones afecta a las siguientes etapas del proceso de desarrollo software. La calidad se mide a través de un cuestionario de calidad completado por profesionales de la industria. El objetivo del experimento es identificar la calidad de las especificaciones de los requisitos del software. Las métricas utilizadas son integridad y grado de utilización de los requisitos para comunicar. Dentro del mismo trabajo de investigación se realiza un experimento controlado con estudiantes como sujetos sobre el impacto de los defectos en la especificación de requisitos; las métricas utilizadas para evaluar la influencia de los defectos son calidad semántica (influencia de enunciados incorrectos), calidad pragmática (impacto de enunciados negados) y relevancia del conocimiento del dominio.

Además de la calidad de las especificaciones de requisitos, otro aspecto importante a considerar es cómo se analizan estas especificaciones en las siguientes etapas del proceso de desarrollo de software. La información presentada en los documentos debe filtrarse y procesarse para continuar con el proceso de desarrollo de software. En [62], el autor definió técnicas de filtrado semántico para el análisis de grandes especificaciones de requisitos textuales. El enfoque utiliza un método de exploración contextual para identificar marcadores lingüísticos específicos que muestran la estructura del conocimiento semántico. El valor semántico se calcula acorde a relaciones entre conceptos, organización aspecto-temporal y causalidad. En el trabajo [63], el autor propuso un método para extraer ideas de una descripción textual de requisitos usando heurísticas. El enfoque consiste en una comparación sistemática de diferentes heurísticas de extracción de términos. Según Kof, las heurísticas basadas en el reconocimiento de entidades nombradas son las que proporcionan el mejor rendimiento.

Hay varios trabajos que han comparado dos técnicas de elicitación de requisitos. Uno de estos trabajos fue desarrollado en [70]. Estos autores realizaron un estudio para evaluar y comparar las técnicas de obtención de requisitos utilizando estudiantes como sujetos. Las técnicas que se comparan son: lenguaje natural, una plantilla de requisitos específicos llamada DIRT y la especificación de requisitos de software IEEE. Las dimensiones utilizadas para comparar son sintáctica, semántica, capacidad de comunicación y usabilidad. Los autores del trabajo presentado en [64] compararon casos de uso versus análisis de comunicación a través de un marco de trabajo denominado Modelo de Evaluación de Métodos (MEM). MEM distingue entre la eficacia real, integridad, semántica, errores de granularidad y eficacia percibida del método. La comparación se realizó utilizando estudiantes como sujetos y cuestionarios. En [66] los autores compararon la comprensibilidad de los modelos de requisitos en casos de uso versus Tropos a través de una familia de experimentos que utilizaron estudiantes como sujetos. Los experimentos se centran en tres categorías de preguntas de comprensión para la resolución de problemas, en el esfuerzo requerido para realizarlas considerando el tiempo empleado y la productividad.

3.2.1 Comparación con otros trabajos

En general, las métricas en los trabajos existentes se centran en la medición de la calidad, la sintaxis, la semántica y el tiempo. Además, todos los estudios empíricos que se han llevado a cabo para comparar las distintas técnicas de especificación se han hecho con estudiantes. Hasta donde sabemos, no hay trabajos previos donde se hayan llevado a cabo experimentos empíricos en la industria con el objetivo de comparar diferentes niveles de enriquecimiento en la especificación de requisitos. De todos los trabajos relacionados estudiados, solo parte del experimento de Mund et al. descrito en [69] (basado en cuestionarios de calidad) se realizó con profesionales de la industria. Como se declara en [72], existe la necesidad de realizar estudios empíricos en la industria ya que el contexto no es el mismo que en la academia. Por lo tanto, nuestro trabajo tiene como objetivo ayudar a cubrir la escasez de estudios que analizan cómo el nivel de descripción de los requisitos puede afectar al desarrollador software en la generación de modelos en un entorno industrial.

3.3 Trabajos relacionados: usabilidad

Se ha realizado un gran esfuerzo en estudiar la usabilidad desde distintos enfoques en el ámbito del desarrollo de software, si bien el esfuerzo en el ámbito de las Líneas de Producto Software ha sido menor y, particularizando para el tipo de operación descrito en el capítulo 6 y, acorde a nuestro conocimiento, no existe ningún estudio empírico que aborde los tres tipos de operación estudiados en el capítulo 6. La Tabla 3.2 muestra una relación de estudios previos realizados en el campo de la usabilidad y las Líneas de Producto Software.

Paper	Año	Objetivo	Métrica	Sujetos
[73]	2006	Evaluar la usabilidad de herramientas de Lenguaje Específico de Dominio	Cualidades gráficas, usabilidad, esfuerzo, evolución, integración y capacidad	No procede
[74]	2008	Propone una técnica para facilitar la visualización en tareas de Ingeniería de Líneas de Producto	Técnicas de visualización	Ingenieros SPL
[75]	2008	Propone una nueva forma de visualización en una herramienta SPL	Vistas en una SPL	No procede
[76]	2008	Muestra las lecciones aprendidas en el desarrollo de una herramienta para Ingeniería de Líneas de Producto Software	No procede	No procede
[6]	2009	Presentan una herramienta para Ingeniería de Líneas de Producto Software que sea flexible y con apoyo al usuario final	Flexibilidad y soporte usuario final	No procede
[77]	2012	Propuesta para focalizar la evaluación de la usabilidad en el desarrollo de DSLs	Usabilidad	No procede
[78]	2012	Evaluar herraminetas de configuración	Usabilidad y utilidad	9 *stakeholders*
[79]	2013	Evaluar una método mejorar la visualización en herramientas de SPL	Tareas completadas	2 graduados + 1 post doc
[80]	2014	Analizar el efecto de la familiaridad con el modelado de características sobre la comprensión de CVL	Comprensión y dificultad percibida	Estudiantes
[81]	2014	Analizar los problemas de comprensión en el modelado ortogonal de la variabilidad	Comprensión, tiempo y dificultad percibida	45 estudiantes

Tabla 3.2: Trabajos relacionados con la dimensión usabilidad por orden cronológico.

Se han realizado esfuerzos de investigación sobre la visualización en SPL relacionados con artefactos. Por ejemplo, en [79] los autores presentan un enfoque para visualizar variantes pareto-óptimas (variantes, con respecto a un conjunto de objetivos donde no se puede mejorar una sola calidad sin sacrificar otras cualidades). Realizan un experimento de evaluación que muestra que el enfoque puede ayudar a los usuarios finales a realizar un conjunto de tareas. Los

sujetos del experimento fueron investigadores en un contexto investigador. En [74] se presenta un enfoque que emplea técnicas de visualización e interacción para apoyar a los usuarios finales en el proceso de derivación del producto. En la misma línea, en [75] presentan una nueva técnica de visualización donde se aboga por la necesidad de soportar diferentes visualizaciones interactivas de mapeos entre características *(features)* y artefactos de realización en una SPL que pueden ser controlados por los desarrolladores. Los trabajos citados en este párrafo se centran en las cualidades propias de visualización de una herramienta de modelado; sin embargo, no consideran las tareas de ámbito y configuración. Por otro lado, sus enfoques se centran en modelos de características, mientras que nuestro enfoque afronta tareas propias del ámbito del modelado de variabilidad.

Existe una preocupación en la literatura científica sobre la comprensión de los modelos de características y las posibles dificultades para los diferentes grupos de usuarios. Por ejemplo, en [80] los autores presentan un enfoque experimental sobre la comprensión de las restricciones de árboles en los modelos de características, para ello participan en el estudio sujetos familiarizados con el modelado de características y sujetos que no lo están. En la misma línea, en [81] se lleva a cabo un experimento exploratorio para examinar posibles problemas de comprensión en dos lenguajes de modelado de variabilidad.

En [76] los autores presentan una herramienta de Línea de Producto Configurable *(Configurable Product Line)* que permite a los usuarios de la Línea de Producto *(Product Line, PL)* personalizarla. Los autores resumen los problemas técnicos de estas personalizaciones para ayudar a los usuarios de la PL a comprender las implicaciones de las decisiones tomadas durante la personalización. En la misma línea, en [6] los autores están preocupados por la flexibilidad de su PL. Y como consecuencia, presentan una herramienta orientada al usuario final que puede admitir a diferentes tipos de usuarios finales, como gerentes de proyecto, personal de ventas o ingenieros realizando sus tareas específicas. En [78] se realiza un análisis de herramientas de configuración existentes para identificar capacidades clave para guiar a los usuarios finales y discutir estas capacidades utilizando un marco de dimensiones cognitivas. Realizaron una investigación cualitativa, mediante la realización de 7 tareas, sobre la utilidad de las capacidades de su herramienta para guiar al usuario en la configuración del producto. Sin embargo, estos enfoques carecen de una evaluación formal de usabilidad que conduzca a medidas y problemas de usabilidad como nosotros presentamos en el capítulo 6.

También se ha detectado una preocupación relativa a la usabilidad en lenguajes específicos del dominio *(Domain Specific Languages, DSL)* y la herramientas

utilizadas para generarlos. Por ejemplo, en [73] los autores presentan una comparación entre cinco herramientas de desarrollo diferentes para crear DSL (y sus editores asociados). Toman en cuenta diferentes criterios como la integridad gráfica, facilidad de uso, esfuerzo de desarrollo, manejo de la evolución del lenguaje, integración con otros lenguajes o capacidades de análisis. En [77] los autores discuten cómo el Diseño Centrado en el Usuario *(User Centered Design)* se puede adaptar al contexto del desarrollo de DSL. Como resultado, argumentan que la usabilidad debe fomentarse desde el comienzo del ciclo de desarrollo del DSL, permitiendo a las personas utilizar el DSL. Estos esfuerzos de investigación podrían usarse en el contexto de una herramienta de modelado de variabilidad; sin embargo, su enfoque no se centra en aspectos específicos de este tipo de herramientas, como los tres tipos de tareas evaluadas en el capítulo 6.

3.3.1 Comparación con otros trabajos

Observando los trabajos referenciados en la Tabla 3.2 las métricas más utilizadas son comprensión, usabilidad y dificultad percibida. Los estudios empíricos que se han llevado a cabo para en el ámbito de las SPLs se han realizado con estudiantes como sujetos. Por otro lado, existen trabajos que tienen en consideración las tareas propias de visualización pero no así las de ámbito y configuración. Finalmente, es destacable la ausencia de estudios empíricos en la industria que hayan evaluado la usabilidad formalmente y hallan concluido con un estudio de los problemas encontrados y una serie de recomendaciones para mitigar o evitar los citados problemas de usabilidad en el ámbito de las SPLs.

3.4 Trabajos relacionados: comprensión de fragmentos

Existen trabajos de investigación sobre distintos aspectos del modelado de la variabilidad e incluso particularizando sobre el estudio de fragmentos, en concreto sobre fragmentos cruzados y varios centrados sobre la visualización. La Tabla 3.3 muestra una relación de estudios previos relacionados con la comprensión de fragmentos de modelo en el contexto de la variabilidad.

Relacionado con la investigación mostrada en el capítulo 7 existen trabajos en la literatura que analizan las dificultades en configuración de productos con fragmentos de modelo. En [88], se identifican los problemas generados después de la superposición de características con fragmentos de modelo cruzados. Estos fragmentos de modelo cruzados pueden causar resultados involuntarios

Paper	Año	Objetivo	Métrica	Sujetos
[74]	2008	Propone una técnica para facilitar la visualización en tareas de Ingeniería de Líneas de Producto	Técnicas de visualización	Ingenieros SPL
[76]	2008	Muestra las lecciones aprendidas en el desarrollo de una herramienta para Ingeniería de Líneas de Producto Software	No procede	No procede
[82]	2009	Estudiar como diferentes tipos de variabilidad influyen en la transfromación de la variabilidad	Confluencia y consistencia	No procede
[6]	2009	Presentan una herramienta para Ingeniería de Líneas de Producto Software que sea flexible y con apoyo al usuario final	Flexibilidad y soporte usuario final	No procede
[83]	2009	Evaluar la notación visual i*	Principios de notación visual	No procede
[84]	2011	Analizar la comprensión de la especificación de activos	Comprensión	Estudiantes
[85]	2011	Presentar un proceso iterativo para mejorar herramientas de metamodelado	Usabilidad	4 ingenieros y un diseñador
[86]	2012	Presentar una automatización de la evolución de modelos	Viabilidad	Autores trabajo
[87]	2014	Investigar como de comprensible son los conceptos de *goal-oriented*	Comprensión y errores	19 profesionales en un curso formativo
[80]	2014	Analizar el efecto de la familiaridad con el modelado de características sobre la comprensión de CVL	Comprensión y dificultad percibida	Estudiantes
[81]	2014	Analizar los problemas de comprensión en el modelado ortogonal de la variabilidad	Comprensión, tiempo y dificultad percibida	45 estudiantes
[88]	2014	Estudiar los problemas con fragmentos de modelo cruzados en procesos de sustitución	Interferencia de fragmentos	No procede
[89]	2014	Conocer la aplicación del modelado de la variabilidad en la industria	Prácticas, beneficios, retos	Representantes de la industria
[90]	2014	Mejorar la notación visual i*	Transparencia semántica	282 estudiantes + 3 evaluadores
[91]	2015	Analizar la utilización del preprocesador en lenguaje C	Utilización, alternativas, problemas, uso anotaciones	Desarrolladores software

Tabla 3.3: Trabajos relacionados con la dimensión comprensión de fragmentos por orden cronológico.

durante la configuración de un producto. En [82], se analizan los conflictos y propiedades que surgen cuando hay múltiples puntos de variación. En [86], se identifican las dificultades con fragmentos de modelo cruzados cuando se modifican los modelos DSL y se presenta una aproximación para automatizar la evolución de modelos cuando el modelo base es transformado. En la investigación descrita en el capítulo 7, no solo hemos encontrado que los fragmentos de modelo cruzados son difíciles de entender (lo que confirma el hallazgo de los trabajos anteriores) pero también analizamos fragmentos de modelo aislados y recursivos, el tiempo necesario para alcanzar una configuración de producto y la dificultad percibida por los sujetos. Esto nos permite obtener nuevos hallazgos sobre las formas geométricas de los fragmentos del modelo, combinaciones de fragmentos de modelo, la principal fuente de soluciones incorrectas, y los pasos redundantes en los procesos de configuración.

Otro tipo de trabajos de investigación en la literatura se han centrado en la comprensión del modelado de la variabilidad. En [80], se presenta un estudio para comprender el modelado de características mediante la realización de un experimento con sujetos que están familiarizados con el modelado y sujetos que no están familiarizados con él. En una línea similar [81] presenta un enfoque que investiga la comprensión de los lenguajes de modelado de variabilidad ortogonal (OVM). Específicamente, condujeron un experimento para examinar los potenciales problemas en la comprensión de dos lenguajes de modelado de variabilidad ortogonal: CVL y OVM. En [84], presentan un análisis comparativo para gestionar la variabilidad utilizando métodos de modelado orientados a características (FODA) vs basados en UML. Sin embargo, los anteriores trabajos no analizan las diferencias en la comprensión usando fragmentos de modelo recursivos, aislados, cruzados y las dificutades para alcanzar la configuración de un producto deseada como lo hace nuestro trabajo.

En el trabajo de investigación [91] se realizan entrevistas con desarrolladores sobre el uso del preprocesador de lenguaje C para manejar la variabilidad, los desarrolladores usan directivas de preprocesado condicionales para proporcionar características opcionales o para seleccionar entre implementaciones alternativas. En [89] presenta un estudio de caso exploratorio de tres empresas que aplican modelos de variabilidad para investigar sobre prácticas, características, beneficios y desafíos del modelado de variabilidad. A pesar de que estos trabajos proporcionan datos empíricos sobre la gestión de la variabilidad en entornos industriales, no investigan las dificultades en entender la configuración del producto utilizando superposición de características para la configuración de modelos.

Existe una línea de investigación para generar SPLs más fáciles de usar por los usuarios finales; en esta misma línea, también hay trabajos que facilitan la comprensión de modelos de variabilidad por los usuarios finales. Por ejemplo, en [76] presentan una herramienta de SPL configurable que permite la personalización por usuarios finales. Los autores resumen los problemas técnicos de esta personalización para ayudar a los usuarios finales y facilitar la comprensión de las implicaciones de las decisiones que toman durante la personalización. En la misma línea, en [6] los autores se preocupan por la flexibilidad de su línea de productos. Ellos presentan una herramienta orientada al usuario final que puede dar soporte a diferentes tipos de usuarios finales, como gerentes de proyectos, vendedores o ingenieros en sus respectivas tareas específicas. En [74] presentan un metamodelo y una herramienta que emplea técnicas de visualización para ayudar a los usuarios en el proceso de configuración del producto. Estos trabajos se centran en aumentar las capacidades de modelado de variabilidad de herramientas para mejorar el *feedback* que las herramientas proporcionan a sus usuarios. Por contraste, nuestro trabajo no aborda el soporte de una herramienta de modelado de la variabilidad y enfocamos en lenguajes de modelado de variabilidad genéricos.

Otros esfuerzos de investigación se han dirigido a la investigación de la comprensión en la interacción de los usuarios con el desarrollo de software. En [87] los autores investigaron cuán comprensibles eran los conceptos orientados a objetivos en dos experimentos con profesionales. En [83] se presenta un sistema sistematizado de análisis de la sintaxis visual de la notación i*. Se evalúa la notación usando un conjunto de principios basados en evidencias para el diseño de notación visual. El trabajo identifica errores graves en la notación visual i* y hace algunas recomendaciones para su mejora. Con un enfoque similar, en [90] se presenta un enfoque novedoso para aplicar una notación visual en la ingeniería de requisitos implicando usuarios inexpertos en el proceso. Concluyen que la notación diseñada por principiantes aumenta la transparencia semántica. Su investigación sugiere que el diseño visual de la notación debe ser realizado por un gran número de usuarios noveles. En [85] se presenta un estudio de caso en una herramienta de metamodelado. El estudio emplea una prueba de usabilidad con prototipos en papel, observaciones, entrevistas y consultas contextuales. Los autores presentan un proceso iterativo para ayudar a mejorar las herramientas para expertos de dominio.

3.4.1 Comparación con otros trabajos

Analizando la Tabla 3.3 se puede observar que las métricas más utilizadas son comprensión, dificultad percibida y errores; son métricas muy alineadas con las empleadas en nuestra investigación. Por contra, nuestra investigación afronta el estudio de fragmentos aislados, cruzados y recursivos. Adicionalmente nuestro estudio afronta el estudio de la comprensión de fragmentos de una forma genérica en el ámbito del modelado de variabilidad.

En [27], realizamos una evaluación de usabilidad de una herramienta de modelado de variabilidad en la que detectamos que los expertos de dominio de nuestro socio industrial (grupo BSH) tenían dificultades para comprender la variabilidad debido a la sintaxis concreta de fragmentos de modelo. El trabajo presentado en el capítulo 7 confirma que la sintaxis concreta de los fragmentos del modelo lleva a los participantes a cometer errores durante las configuraciones del producto; por otro lado, la investigación del capítulo 7 revela que es posible aprovechar la sintaxis para reducir significativamente el tiempo requerido para configurar los productos. En este capítulo también se describen nuevos hallazgos sobre combinaciones de fragmentos modelo, las fuentes principales de configuraciones erróneas y la realización de pasos de configuración redundantes.

3.5 Trabajos relacionados: gestión de errores

La gestión de errores ha sido un tema estudiado en el ámbito del desarrollo de software de forma general y en la generación de modelos software de forma particular. Siendo conscientes de esta realidad, observamos que la mayoría de los trabajos existentes desarrollados mediante estudios empíricos en el ámbito de las SPLs y relacionados con la gestión de errores se centran en analizar las técnicas de prueba para buscar errores, pero no se ocupan de cómo solucionar esos errores. La Tabla 3.4 se muestra una relación de estudios previos realizados en el campo de la gestión de errores ordenados de forma cronológica.

Considerando trabajos centrados en la detección de errores, en [97] se realizó una evaluación en una SPL para encontrar errores en un entorno industrial utilizando una técnica de test por pares llamada MoSo-PoLiTe, la técnica se utiliza sobre la herramienta pure::variants y las métricas utilizadas son tasa de errores detectados, esfuerzo y tiempo. En [99] se propone un método denominado *method-level project centralization* que unifica y comparte métodos para testear SPLs. La nueva técnica propuesta comparte código común siempre que sea posible, mientras conserva el comportamiento de cada variante de

Paper	Año	Objetivo	Métrica	Sujetos
[92]	2007	Busca los efectos negativos de cambios en modelos basados en UML	Corrección, integridad, usabilidad y escalabilidad	No procede
[93]	2010	Propuesta para solucionar modelos de características inconsistentes	Usabilidad y escalabilidad	3 estudiantes + 2 graduados
[94]	2010	Evalúa una técnica de test para SPLs	Rendimiento	No procede
[95]	2011	Propuesta para encontrar y solucionar errores en una SPL	Usabilidad	2 estudiantes+ 2 investigadores
[96]	2011	Analizar la evolución de una SPL	Fallos, cambios y relaciones fallo/cambio	No procede
[97]	2012	Evaluar una SPL para encontrar errores	Tasa de errores, esfuerzo y tiempo	No procede
[98]	2012	Evaluar el uso de patrones en la gestión de errores	Tiempo	No procede
[99]	2014	Evaluar una método para testear SPL	Eficacia y código procesado	No procede
[100]	2015	Detectar y corregir defectos en una SPL	Precisión y rendimiento	No procede
[101]	2016	Cuantificar el impacto de la variabilidad en la búsqueda de errores	Tiempo y precisión	63 estudiantes (M y Ph D) + 6 Ph. D
[102]	2016	Mejorar el rendimiento del control de calidad para las SPL	Métricas de proceso	No procede
[103]	2017	Detectar y corregir defectos en una SPL	Precisión y escalabilidad	No procede

Tabla 3.4: Trabajos relacionados con la dimensión gestión de errores por orden cronológico.

producto dada para pruebas unitarias. Implementa la generación de casos de prueba aleatorios considerando la SPL centralizada como entrada y probando variantes de productos en cada ejecución. La evaluación se realiza sobre 3 SPLs midiendo código procesado y eficacia. En [94] se evalúa una técnica de test que genera incrementalmente pruebas para SPLs. Para realizar la evaluación se realiza una comparativa sobre el rendimiento entre la técnica presentada y una técnica convencional.

Existen otros enfoques que han estudiado la fiabilidad de las mejoras en la evolución del SPL a través de estudios empíricos, un ejemplo es mostrado en [96]. En este trabajo de investigación se analiza hasta qué punto las partes comunes y los puntos de variación cambian con el tiempo. Se estudia la ocurrencia de fallos, cambios en el código fuente, y la adición de nuevos archivos en la línea de productos de Eclipse entre 2007 y 2010. Para ello se tiene en cuenta las tendencias de errores, las tendencias de cambio y relaciones fallo/cambio. Otros enfoques como el caso mostrado en [98] han evaluado cómo el uso de patrones puede mejorar la gestión de errores. Estudiando la gestión de los errores en los proyectos Eclipse y JTD se analiza la influencia de los distintos roles implicados en la citada gestión. En [102] se investiga cómo los modelos de predicción de defectos se pueden utilizar para identificar características defectuosas en

SPLs utilizando técnicas de aprendizaje automático, el objetivo de mejorar el rendimiento de las actividades de control de calidad para las características de las SPL.

Algunos trabajos de tipo empírico han abordado la corrección de errores en desarrollos software fundamentado en modelos. Por ejemplo, en [100] hace una propuesta sobre un ejemplo teórico para detectar posibles correcciones en un modelo con fallos de una SPL. Propone un método que identifica errores semánticos en un modelo de características y estima posibles correcciones para cada defecto. En [92] explora como descubrir los efectos negativos de cambios en modelos basados en UML, como encontrar opciones de corregirlos y las consecuencias de estas correcciones. En [103] se muestra un estudio sobre errores en modelos de caracterísiticas que representan SPLs y se propone un método basado en reglas para detectar y corregir estos errores.

Considerando aquellos estudios empíricos en los que participan de forma activa sujetos en el estudio, en [93] se propone, basado en prioridad dinámica, una aproximación para solucionar modelos de características inconsistentes. Para ello utilizan dos estudios de caso: en el primero comprueban si el enfoque es útil para analistas de dominio (participan 3 estudiantes y 2 graduados) y en el segundo caso comprueban si el enfoque es usable mediante un modelo teórico. En [95] se propone una herramienta automatizada para encontrar y solucionar errores en una SPL representada mediante modelos de características. Realizan una prueba experimental, en la que participan 2 estudiantes y 2 investigadores midiendo la usabilidad de la herramienta. Finalmente, en [101] se lleva a cabo un experimento controlado para cuantificar el impacto de la variabilidad en la la búsqueda de errores. Se mide la velocidad y la precisión para detectar errores considerando tres grados diferentes de variabilidad. Los resultados muestran que identificar el conjunto de configuraciones en las que se manifiesta un error es difícil incluso con un bajo grado de variabilidad. Identificar el conjunto exacto de las configuraciones afectadas parece ser más difícil que encontrar el error primigenio.

3.5.1 Comparación con otros trabajos

Observando los trabajos descritos en el apartado anterior aparece un abanico amplio de métricas, si bien destacan: usabilidad, escalabilidad, rendimiento, tiempo y precisión. Por otro lado, y referido al enfoque, encontramos que la mayoría de trabajos abordan la gestión de errores desde el punto de vista de la detección de los mismos: son minoría los enfoques que abordan la corrección de los mismos. Finalmente, y hasta donde sabemos, no se han realizado estudios

empíricos en la industria con profesionales donde se avalúe y analice la gestión de errores por los citados profesionales. En la línea iniciada en [101], y aunque en el citado trabajo el estudio no se realiza con profesionales, es necesario desarrollar trabajos de investigación para cubrir la escasez de estudios que analizan cómo los profesionales ingenieros de software realizan las gestión de errores en su desempeño profesional.

3.6 Conclusión

Realizando un análisis numérico de los trabajos referenciados se puede concluir que son muy pocos los trabajos que abordan las dimensiones tratadas en esta tesis en el estudio de desarrollo de software MDD+SPL. Y haciendo un análisis con un mayor nivel de concreción son muy escasos los trabajos realizados en la industria o con sujetos que no sean estudiantes.

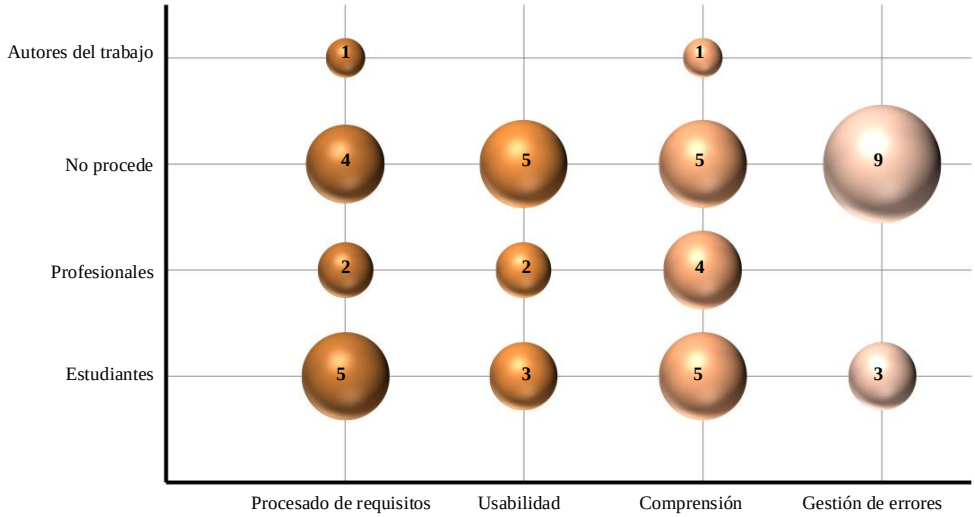


Figura 3.1: Resumen trabajos referenciados considerando sujetos participantes.

Un análisis de los datos mostrados en la Figura 3.1 arroja los siguientes resultados:

- En el 46.9 % de los trabajos referenciados "No procede" considerar a los sujetos participantes en el estudio: se trata de aproximaciones sin validar,

evaluaciones heurísticas o estudios teóricos donde no ha existido participación de usuarios.

- En los trabajos de carácter empírico en los que han participado sujetos, en un 61.5 % de los casos los sujetos son estudiantes, en un 30.7 % de las ocasiones los sujetos son profesionales de la industria y en un 7.6 % de las ocasiones los sujetos son los propios autores del trabajo científico.

Observando todos los trabajos referenciados en este capítulo, y siguiendo la referencia particularizada para el estudio de los requisitos [72], podemos concluir que para mejorar e incrementar el conocimiento y adopción por parte de la industria de paradigmas de desarrollo software MDD+SPL es necesario involucrar en la investigación a la propia industria. Se antoja necesaria una mayor colaboración en dos sentidos: conocer con mayor detalle sus procesos de desarrollo y hacer partícipes a los propios integrantes de la industria en los estudios empíricos que permitan avanzar en los campos de modelado y variabilidad del software.

4

Overview

Índice

4.1 Resumen del capítulo

En este capítulo se describe como cualquier desarrollo software se inicia a partir de unos requisitos que se deben satisfacer. Partiendo de esta información y en determinadas situaciones, este desarrollo software puede llevarse a cabo utilizando los paradigmas del MDD y las SPLs. Este trabajo de tesis tiene como objetivo conocer y proponer mejoras para aquellas situaciones en las que se desarrolla software con los dos paradigmas descritos en el capítulo 2 de forma simultánea.

4.2 Desarrollo de Software

Actualmente cualquier ser humano en un país avanzado se encuentra rodeado de objetos que, de una u otra forma, necesitan software para funcionar: televisiones inteligentes, ordenadores o *smartphones* son ejemplos cotidianos de esta realidad. El proceso que ha permitido crear este software se denomina desarrollo de software *(Software Development)*, proceso por el cuál necesidades de los usuarios son materializadas en un producto software. El proceso involucra traducir las necesidades del usuario a requisitos software, transformar los requisitos software en un diseño, implementar el diseño en código fuente, probar el código fuente y, en ocasiones, instalar y chequear el software en funcionamiento [104]. Por lo tanto, el desarrollo de software puede incluir investigación, desarrollo, creación de prototipos, modificación, reutilización, reingeniería, mantenimiento o cualquier otra actividad que resulte en productos de software.

El software se puede desarrollar con varios propósitos (uso particular, o empresarial) y en innumerables dominios: telefonía móvil, ordenadores, placas de inducción o fabricación de trenes son algunos ejemplos. La variedad de sus posibles aplicaciones es innumerable, pero hay algo común a todos los desarrollos software: su objetivo es satisfacer unas necesidades específicas en un determinado dominio. Esas necesidades quedarán reflejadas en los requisitos software que el producto software a desarrollar debe cumplir. Se define requisito software como [104]:

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.

2. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.

3. Una representación documentada de una condición o capacidad como en (1) o (2).

El incremento de los sectores en los que el software se hace imprescindible, la cada vez mayor calidad exigida al software, y la variación de necesidades de los usuarios a cubrir por productos software ha desembocado en la necesidad de un mejor control de calidad del proceso de desarrollo de software. Como consecuencia surge la disciplina de Ingeniería de Software, que pretende aplicar el enfoque sistemático ejemplificado en el paradigma de ingeniería al proceso de desarrollo de software. Han aparecido muchos enfoques para la gestión de proyectos de software, conocidos como modelos de ciclo de vida de desarrollo de software, metodologías, procesos o modelos: dos de ellos son MDD y SPL (ver 2.2 y 2.3 respectivamente).

Una SPL se fundamenta en un conjunto de opciones posibles, denominados puntos de variación, y un conjunto de activos predifinidos, que pueden ser conectados a esos puntos de variación. Una SPL está pensada expresamente para gestionar lo común, y su complementario, lo variable. El término que mejor define una SPL es la reutilización. Esta reutilización no es oportunista, es planificada, y la incorporación de nuevas variantes se realiza de forma sistemática y controlada. Esto agiliza el desarrollo del producto, su puesta en el mercado y su mantenimiento [43]. El paradigma de SPL utiliza dos procesos [38]:

- Ingeniería de Dominio: se establece la plataforma de productos software, identificando requisitos del espacio del problema, se desarrolla la arquitectura software que toma en consideración estos requisitos y los componentes software que se ajustan a esa arquitectura.

- Ingeniería de Aplicación: es responsable de derivar aplicaciones individuales desde los requisitos de un cliente, cumpliendo con las variaciones permitidas desde la arquitectura y los componentes permitidos.

4.3 Desarrollo de Software: MDD+SPL

Independientemente del paradigma de desarrollo software utilizado, el proceso de desarrollo software debe garantizar que el producto software final corresponda de forma precisa con los requisitos del sistema y esté alineado con las reglas de la organización del sistema analizado [105]. El producto software final desarrollado con MDD, SPL o mediante un proceso de desarrollo tradicional debe estar alineado con las metas y procesos del negocio. En otras palabras, debe satisfacer las necesidades de los usuarios expresadas (como se puede apreciar en la Figura 4.1a)) en una serie de requisitos.

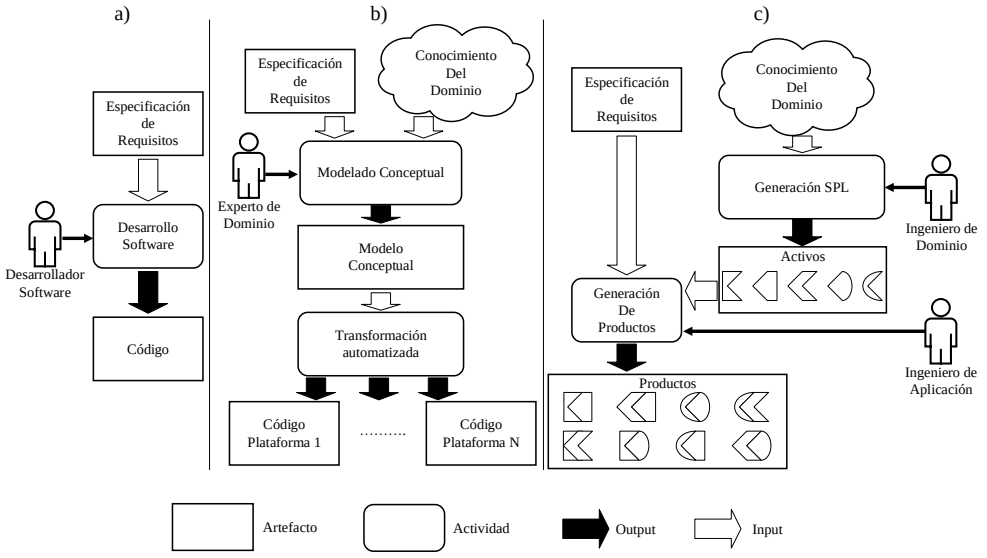


Figura 4.1: a) Desarrollo de software tradicional. b) Desarrollo de software con MDD. c) Desarrollo de software con MDD+SPL.

Los paradigmas de desarrollo software han aparecido y evolucionado a la par que lo han hecho las necesidades y exigencias requeridos a ese desarrollo: de una forma similar que desde los lenguajes de programación estructurados se ha evolucionado hacia los lenguajes orientados a objetos. De igual forma que en este trayecto se ha incrementado el nivel de abstracción, acercándose el desarrollo más al concepto que al ámbito funcional, el paradigma del MDD se acerca el desarrollo de software al pensamiento conceptual del ser humano, alejándose de la implementación subyacente.

El desarrollo MDD (ver Figura 4.1 b)) permite a los expertos de dominio generar modelos conceptuales de alto nivel de abstracción, que permiten mediante las transformaciones adecuadas derivar modelos de más bajo nivel de abstracción y finalmente generar el producto software final. Los modelos pueden ser generados utilizando lenguajes de modelado de propósito general, como puede ser el Lenguaje Unificado de Modelado *(Unified Model Language, UML)*, o Lenguajes Específicos de Dominio *(Domain Specific Language, DSL)*, como pueden ser un lenguaje específico de dominio para la generación de placas de inducción [106]. La generación de los productos software desde modelos mediante transformaciones permiten, y es una de las grandes ventajas, que el código en el que es generado el producto software final pueda ser modificado con menos coste que en el desarrollo de software tradicional.

En el desarrollo de software basado en MDD+SPL (ver Figura 4.1 c)) se generan fragmentos de modelo preparados para ser utilizados en el desarrollo de futuros productos, este hecho va a permitir generar productos en serie de forma personalizada *(mass customization)* y, como en los otros ejemplos, acordes a unos requisitos establecidos. El paradigma de desarrollo MDD se ajusta a la perfección al concepto de SPL [38]: se pueden desarrollar modelos que pueden ser transformados en distintas aplicaciones o productos software, familiarizados entre ellos, en función de los requisitos del cliente.

Como se ha descrito anteriormente, en teoría es claramente plausible un desarrollo de software utilizando al unísono los paradigmas Desarrollo Dirigido por Modelos y Líneas de Producto Software (MDD+SPL). Desde le grupo de investigación *SVIT (Software Variability for Internet of Things)*[1] hemos desarrollado herramientas para la generación de software utilizando los anteriores paradigmas en diferentes contextos industriales. Dos de estos contextos son (1) la fabricación de placas de inducción y (2) la fabricación de trenes. Se han generado dos herramientas de desarrollo software siguiendo MDD+SPL: una herramienta para el desarrollo de software en el dominio de las placas de inducción[2] y otra herramienta en el dominio de la fabricación de trenes [3].

Los dos procesos de desarrollo MDD+SPL que utilizan las citadas herramientas tienen un nexo común: en ambos casos existía anteriormente un desarrollo MDD al que posteriormente se le aplicó una SPL. Si consideramos los dos procesos que conforman la SPL (Ingeniería de Dominio e Ingeniería de Aplicación, ver 4.2) el MDD es utilizado en ambos procesos, puesto que en ambos se utilizan las reglas y modelos acordes al MDD inicial. En relación al

[1] `https://svit.usj.es/`
[2] `https://svit.usj.es/see-our-model-patterns-variability-tool/`
[3] `https://svit.usj.es/know-our-train-control-management-system-tcms-variability-tool/`

MDD, se puede observar en la parte inferior de la Figura 4.2, en el desarrollo MDD+SPL estudiado se produce una única transformación de modelo a código: dicha transformación convierte el artefacto "Modelos de Productos" a "Código de Productos".

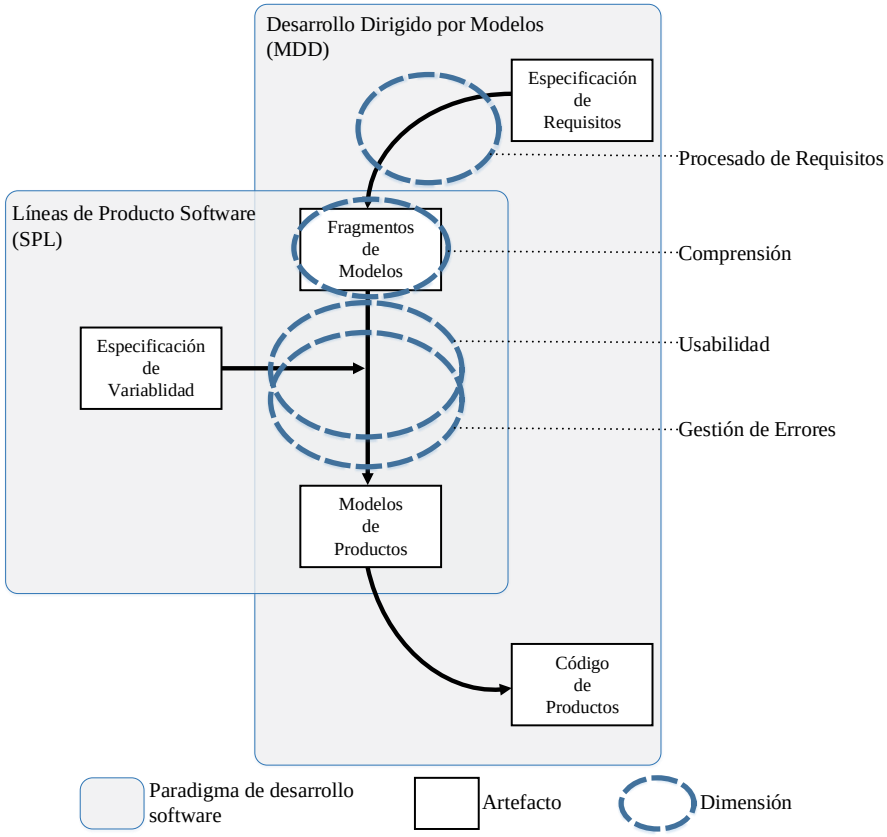


Figura 4.2: Dimensiones investigadas en desarrollo de software MDD+SPL.

Durante el proceso de generación de las herramientas citadas anteriormente, y con el objetivo último de mejorar el proceso de desarrollo software MDD+SPL, se ha intentado conocer los valores de las siguientes dimensiones:

- **Procesado de requisitos.** Como ya se ha comentado, cualquier desarrollo software tiene como claro objetivo desarrollar un producto software acorde a unos requisitos. En relación a esta afirmación, surge el interrogante de como afectan las características de los requisitos en la generación de modelos.

- **Usabilidad.** Se considera importante conocer que características, desde el punto de vista de la usabilidad, se deben considerar y mejorar en una herramienta generada para ser utilizada con el paradigma de desarrollo MDD+SPL en un entorno industrial.
- **Comprensión.** El objetivo de la tesis es mejorar el proceso de desarrollo software MDD+SPL: se tienen fragmentos de modelo preparados para ser reutilizados y combinados en la generación de nuevos productos software mediante la utilización de modelos. El desafío es intentar conocer como se produce la gestión de esos fragmentos de modelo.
- **Gestión de errores.** En [43] menciona como son los esfuerzos de mantenimiento de una SPL o el control de errores debido a la naturaleza de la SPL: *"un error detectado en un producto puede ser relativamente fácil de corregir en todos los productos de la línea gracias precisamente a la existencia de este marco común que ofrece la SPL"*. El desafío es descubrir la facilidad para solucionar errores dependiendo de las características de los mismos.

Para investigar las dimensiones anteriores se ha seguido una perspectiva empírica, implementada principalmente en entornos industriales reales. En cualquier desarrollo de software, incluidos los llevados a cabo en entornos industriales reales, se parte de un conjunto de requisitos como entrada que, tras pasar las etapas de desarrollo establecidas, dan como resultado un código que deberá satisfacer las necesidades descritas en los citados requisitos. La Figura 4.2 refleja como han sido estudiadas las anteriores dimensiones dentro de un proceso de desarrollo software MDD+SPL:

- **Procesado de requisitos.** Para investigar el procesado de requisitos se ha estudiado como fragmentos de modelo y modelos que los usan son generados a partir de una especificación de requisitos.
- **Usabilidad.** Se ha estudiado la usabilidad de una herramienta generada para ser utilizada con el paradigma de desarrollo MDD+SPL en un entorno industrial. Como queda se puede observar en la Figura 4.2 la usabilidad ha sido estudiada considerando la relación entre fragmentos de modelo, especificación de variabilidad y modelos de productos.
- **Comprensión.** Para conocer como se comprende la generación de productos con fragmentos de modelo se ha evaluado como son entendidos y combinados para generar productos software.

- **Gestión de errores.** De forma análoga a la dimensión usabilidad, la gestión de errores se ha estudiado considerando la relación entre fragmentos de modelo, especificación de variabilidad y modelos de productos. El estudio ha sido desarrollado con una herramientra generada para ser utilizada con el paradigma de desarrollo MDD+SPL en un entorno industrial.

4.3.1 Antecedentes MDD+SPL

Los enfoques MDD y SPL tienen preocupaciones comunes: mejorar la productividad, hacer que el desarrollo de software sea rentable, capacitar a expertos de dominio en desarrollo de software, capturar dominio y conocimiento tecnológico en artefactos separados, aumentar la calidad del software y automatizar la construcción de productos de software tanto como sea posible. También son complementarios. El desarrollo basado en modelos parece una tendencia prometedora en la automatización de la cadena de producción necesaria para la creación de SPLs y es adecuado para modelar diversos artefactos de una SPL [107].

Por otro lado, una SPL puede ser construida utilizando un enfoque MDD, lo cual puede ser una ventaja porque los productos de la línea podrán ser construidos mediante un proceso generativo que utiliza modelos y transformaciones de los mismos [108]. Las Líneas de Productos de Software impulsadas por modelos combinan la capacidad de abstracción del Desarrollo Dirigido por Modelos y la capacidad de administración de variabilidad de la Ingeniería de Líneas de Producto Software [109]. En las SPLs basadas en modelos, los activos principales son fragmentos de modelo de software reutilizables en lugar de fragmentos de código reutilizables. Por lo tanto, los productos de las SPLs impulsadas por modelos adoptan la forma de modelos, a partir de los cuales el código de los productos de software se puede generar mediante transformaciones de modelo a texto.

Considerando el enfoque MDD, el aumento del nivel de abstracción permite representar el modelo del problema y utilizarlo como elemento de primera clase para obtener el modelo de la solución independientemente de la plataforma a través de transformaciones. La combinación de MDD y SPL permite aprovechar el carácter generativo para facilitar la derivación de productos a través de la representación del problema y no de la solución. Esta combinación permite separar la solución de la plataforma específica. La ventaja que tiene la combinación de MDD y SPL es que es posible abstraer los conceptos principales de los diferentes dominios y plasmarlos en metamodelos.

En la raíz de un desarrollo que combina MDD y SPL radica el reconocimiento no solo de su complementariedad, sino que su integración conlleva el potencial de importantes sinergias. Si bien MDD puede ayudarnos a representar aspectos diferentes de una línea de productos de manera más abstracta, la Ingeniería de Líneas de Producto Software proporciona un ámbito de aplicación bien definido, que establece el desarrollo y la selección de lenguajes de modelado apropiados sobre una base sólida. Además, el análisis automatizado y la generación de código ofrecidos por modelos precisos pueden ayudar a automatizar la creación de productos de una SPL [109].

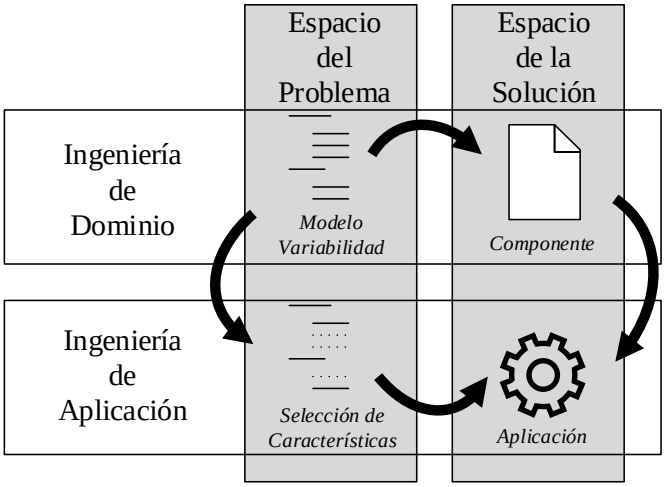


Figura 4.3: Relación en desarrollo software MDD+SPL.

En [38] se muestra como se establece la relación entre MDD y SPL. Como se puede apreciar en la Figura 4.3 el paradigma SPL usa dos procesos (Ingeniería de Dominio e Ingeniería de Aplicación) y dos espacios (Espacio del Problema y Espacio de la Solución): (1) el Espacio del Problema se enfoca en definir que problema abordará la familia de aplicaciones, o una aplicación individual en la familia; (2) el Espacio de la Solución se enfoca en producir los componentes de software para resolver ese problema; (3) la Ingeniería de Dominio es responsable de establecer la plataforma de familia de software, primero identificando los requisitos típicos del espacio problemático y dónde estarán autorizados a variar, segundo desarrollando la arquitectura de software que aborda estos requisitos y los componentes de software que encajan en el arquitectura, y; (4) la Ingeniería de Aplicación es responsable de derivar aplicaciones individuales de los requisitos de un cliente (dentro del conjunto de variaciones autoriza-

das) y componer esta aplicación a partir de la arquitectura y los componentes disponibles.

El MDD es un candidato natural para encajar en el marco general de una SPL: se puede desarrollar un metamodelo que se pueda transformar en diferentes aplicaciones de acuerdo con los deseos del cliente. La aproximación general se puede observar en la Figura 4.3. Primero, en la Ingeniería de Dominio, se define un metamodelo del problema (parte superior-izquierda), especificando los conceptos que se pueden usar en la creación de una solución, y un modelo de todas las características disponibles. En segundo lugar, siguiendo en Ingeniería de Dominio, pero en el Espacio de la Solución (parte superior-derecha), se definen las reglas de transformación para generar código a partir del modelo de aplicación (cuando esté disponible). Obviamente, el futuro modelo de aplicación debe ajustarse al metamodelo de la línea de productos. En tercer lugar, en la Ingeniería de Aplicación (parte inferior-izquierda), se define un modelo de una aplicación particular (que se ajusta al metamodelo definido en la Ingeniería de Dominio). También se seleccionan las características que deberían implementarse en esta aplicación en particular. Finalmente, en el Espacio de la Solución (parte inferior-derecha), la aplicación se deriva automáticamente del modelo aplicando las reglas de transformación.

Se han dado esfuerzos para combinar el desarrollo basado en modelos y las Líneas de Producto Software. Por ejemplo, en [110] aparece una propuesta para combinar MDA y SPL. Considerando que el propósito principal de MDA no es la creación de familias de productos, la separación de dominios y la naturaleza generativa, hacen de MDA un enfoque de gran utilidad para la creación de SPLs. Esto se logra partiendo de un modelo inicial, y utilizando varios mecanismos de transformación automáticos para obtener las diferentes aplicaciones miembros de una familia de productos.

En [109] se describe una técnica particular para el desarrollo impulsado por modelos de líneas de productos: una plantilla de modelo puede representar un conjunto de variantes de modelo en una forma superpuesta dentro de un único artefacto. Las variantes representadas podrían ser modelos conductuales, estructurales o no funcionales. Además, una plantilla de modelo basada en características también contiene un modelo de características que representa concisamente las opciones disponibles dentro del conjunto de variantes.

4.4 Descripción de las caracterísiticas de los trabajos de investigación publicados

Antes de pasar a describir las características compartidas por los estudios empíricos realizados como parte nuclear de esta tesis, es importante conocer los principales métodos de investigación empírica en el ámbito de la Ingeniería del Software. Existen cinco clases de métodos de investigación más relevantes [111]:

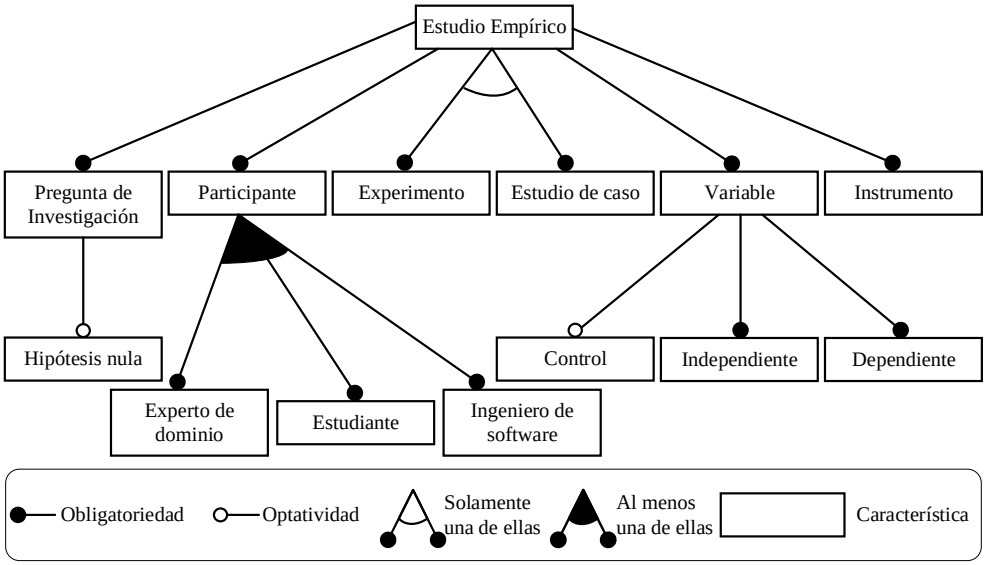
- Experimentos controlados: un experimento controlado es una investigación de una hipótesis comprobable en la que una o más variables independientes se manipulan para medir su efecto en una o más variables dependientes. Los experimentos controlados nos permiten determinar en términos precisos cómo están relacionadas las variables y, específicamente, si existe una relación de causa-efecto entre ellas. Cada combinación de valores de las variables independientes es un tratamiento.
- Estudios de casos: una investigación empírica que investiga un fenómeno dentro de su contexto real, especialmente cuando los límites entre el fenómeno y el contexto no son claramente evidentes [112]. Los estudios de casos ofrecen una comprensión profunda de cómo y por qué ocurren ciertos fenómenos y pueden revelar los mecanismos de relaciones de causa-efecto.
- Investigación-acción: los investigadores intentan resolver un problema del mundo real mientras que simultáneamente estudian la experiencia de resolver el problema [113]. Mientras que la mayoría de los métodos de investigación empírica intentan observar el mundo tal como existe actualmente, este método de investigación intenta intervenir en las situaciones estudiadas con el propósito explícito de mejorar la situación.
- Etnografías: es una forma de investigación centrada en la sociología del significado a través de la observación de campo. Para la Ingeniería del Software, la etnografía puede ayudar a entender cómo las comunidades técnicas construyen una cultura de prácticas y estrategias de comunicación que les permite llevar a cabo el trabajo técnico de manera colaborativa. Una etnografía podría centrarse en una amplia comunidad técnica (por ejemplo, programadores java en general), o una pequeña comunidad estrechamente unida (por ejemplo, un solo equipo de desarrollo).
- Encuestas de investigación: se utilizan para identificar las características de una amplia población de individuos. Está más estrechamente asociado con el uso de cuestionarios para la recopilación de datos. Sin embargo, las

encuestas de investigación también puede realizarse mediante entrevistas estructuradas o técnicas de registro de datos. La característica definitoria de la investigación encuestada es la selección de una muestra representativa de una población bien definida y las técnicas de análisis de datos utilizadas para generalizar de esa muestra a la población.

Se puede aplicar una variedad de métodos a cualquier problema de investigación; con frecuencia es necesario usar una combinación de métodos para desarrollar un correcto análisis. La elección de los métodos depende varios factores: postura teórica de los investigadores, acceso a los recursos (por ejemplo, estudiantes o profesionales como sujetos) y la adecuación del método a las preguntas planteadas. En el proceso de diseño de la investigación se debe seleccionar un método de investigación para un problema en particular, de tal forma que se aprovechen sus fortalezas, y mitiguen sus debilidades.

Una vez conocidos los métodos de investigación empírica en el ámbito de la Ingeniería del Software se van a describir los diversas características de los estudios empíricos realizados. Para representar estos estudios se va a utilizar un modelo de características. Este tipo de representación es utilizada tradicionalmente para mostrar las partes comunes y variables en SPLs (se ha utilizado una de las notaciones para modelos de características mostrada en [114]). En este trabajo se utilizan para definir todas las posibles variables y las relaciones existentes entre ellas en cada uno de los estudios empíricos desarrollados. La Figura 4.4 explica el significado de algunos de los conceptos representados en el modelo de características y muestra los componentes básicos de los estudios empíricos desarrollados durante la realización de esta tesis. Algunas características de nuestros trabajos son:

- Tipo de estudio: define el tipo de estudio. Los estudios puestos en práctica han sido experimentos controlados y estudios de caso en el ámbito de la Ingeniería del Software.
- Preguntas de investigación: se enuncian en la primera fase de diseño del estudio y una de las finalidades del estudio es responder a estas preguntas. Especifican la dirección de la investigación. Cada pregunta de investigación debe ser respondida por el trabajo, por lo que debe ser una pregunta específica a la que se pueda responder de forma concreta.
- Hipótesis nula: es una idea o sugerencia que está basada en hechos conocidos y es utilizada como base para el razonamiento o mayor investigación. Algunos de los estudios definen hipótesis nulas para las preguntas de investigación que serán aceptadas o rechazadas.



Características: son los conceptos que van a definir el estudio empírico, algunos ejemplos son *Pregunta de Investigación* o *Participante*.

Círculos rellenos: la presencia de un círculo relleno al final de cada línea indica que la característica al final de la línea es de obligada presencia en el estudio. Algunos ejemplos son *Variable* o *Instrumento*, todos los estudios tendrán *Variables* e *Instrumentos*.

Círculos vacíos: la presencia de un círculo relleno al final de cada línea indica que la característica al final de la línea es de optativa en el estudio. Un ejemplo es *Variable de Control*.

Arcos vacíos: los arcos vacíos indican que solo una de las características agrupadas pueden formar parte del estudio empírico (función lógica XOR). Por ejemplo, el *Estudio empírico* puede ser *Experimento* o *Estudio del caso*.

Arcos rellenos: los arcos rellenos indican que de las características agrupadas en un arco relleno, al menos una de ellas debe formar parte del estudio. Por ejemplo, de las características *Experto de dominio*, *Estudiante* e *Ingeniero de software* al menos una de ellas debe formar parte del estudio.

Figura 4.4: Modelo de características estudio empírico genérico.

- Participante: engloba a todas las personas que, de una u otra forma, han colaborado en el estudio.
- Variable, en los estudios desarrollados aparecen los siguientes tipos de variables [115]:
  - Independiente: una variable independiente es aquella cuyo valor no depende de otra variable. Es aquella característica o propiedad que se supone es la causa del fenómeno estudiado. En investigación experimental se llama así a la variable que el investigador manipula para ver los efectos que produce en otra variable. A cada una de las posibles combinaciones entre todas las variables independientes se le denomina tratamiento. Las variables independientes utilizadas han sido: tipo de requisito, regla de extracción, propagación, tipo de fragmento, tipo de operación y capa de operación.
  - Dependiente: es el factor que el investigador observa o mide para determinar el efecto de la variable independiente. La variable dependiente es la variable salida o respuesta, se le considera dependiente porque sus valores van a depender de los valores de la variable independiente. Las variables dependientes utilizadas para valorar las cuatro dimensiones en estudio han sido: eficacia, eficiencia, satisfacción, dificultad percibida, tiempo y comprensión.
  - Control: son variables que pueden afectar el resultado del experimento y que deberán ser bloqueadas o controladas por el investigador para evitar su posible efecto y no contaminen los valores obtenidos de la variable dependiente. Con la finalidad de contrarrestar su influencia se pueden asumir valores constantes o introducir aleatoriedad.
- Instrumento: cualquier elemento, de tipo físico o software, que es utilizado para recoger medidas durante el desarrollo del estudio empírico o para la correcta ejecución del mismo. Los instrumentos utilizados para realizar los estudios han sido variados: hojas con descripción de tareas a realizar, distintos tipos de cuestionarios a completar por los sujetos, modelos software a generar, focus group y entrevistas.

Además de las características descritas anteriormente, en todos los estudios empíricos se ha considerado:

- Amenazas de validez: para describir las amenazas de validez de cada uno de los estudios empíricos se ha utilizado la clasificación proporcionada por

[116], se distinguen 4 aspectos para la validez. Estos aspectos son validez de construcción, validez interna, validez externa y confiabilidad.

- Todos los trabajos han sido realizados contando con la colaboración de sujetos (aquellas personas que han formado parte de los estudios realizando las distintas tareas, completando cuestionarios o respondiendo en entrevistas).

En los siguientes capítulos se aborda una dimensión en cada capítulo, en cada capítulo se especifica y completa una variación detallada de la Figura 4.4 donde es representado cada trabajo de investigación utilizado en el estudio de las citadas dimensiones.

4.5 Conclusiones

El paradigma de las SPLs se ajusta a la perfección a otro paradigma de desarrollo de software: MDD. Este trabajo de tesis estudia las características y posibles mejoras de procesos de desarrollo software, que se sustentan en la producción de software con MDD+SPL, especialmente en entornos de desarrollo industriales reales. Para cumplir este objetivo se han generado 7 trabajos de investigación, centrados en 4 dimensiones *(Procesado de requisitos, Comprensión, Usabilidad y Gestión de errores)* del desarrollo software cuando se utiliza MDD+SPL y de marcado carácter empírico en el ámbito de la Ingeniería del Software. En este capítulo se han descrito los elementos fundamentales de los estudios empíricos que se han realizado. Para facilitar la comprensión de los diseños de estos trabajos empíricos se ha utilizado un diagrama de características que contiene los aspectos básicos de los citados diseños.

Parte II

Desarrollo

Los tipos de fragmentos de modelo utilizados son: aislados, cruzados y recursivos.

6. **Evaluating Bug-Fixing in Software Product Lines: an Industrial Case Study (ESEM´16)** [30]: Aborda la dimensión *Gestión de errores*. Este trabajo de investigación recoge un estudio realizado en un entorno industrial. Se evalúa el desempeño de ingenieros de software solucionando problemas en modelos software dependiendo de la capa en la que está localizado el fallo y si, para solucionar el fallo, es necesario propagar los cambios o no propagarlos. Este estudio de caso aborda la modificación de fragmentos de modelo, la generación de modelos de productos y la especificación de la variabilidad.

7. **Research on Augmenting the MDD Process with Variability Modeling (IDoESE´16)** [31]: Aborda las dimensiones *Comprensión, Usabilidad y Gestión de errores*. Este trabajo de investigación es un compendio resumido de los trabajos realizados hasta la mitad del desarrollo de esta tesis.

Hay dos proyectos en los que ha formado parte el trabajo presentado en esta tesis: (1) (VARIAMOS) un proyecto del plan de investigación nacional español cuyo objetivo es la extracción de variabilidad en forma de fragmentos de modelo para lograr la adopción de desarrollos fundamentados en la utilización de Líneas de Producto Software; (2) (REVaMP2) un proyecto internacional ITEA 3 call 2 cuyo principal objetivo es la creación de una plataforma holística y un proceso para la extracción de la variabilidad.

Los trabajos enumerados anteriormente se han realizado en colaboración de dos socios industriales: (1) CAF, siglas que corresponden a la empresa de fabricación de trenes Construcciones y Auxiliar de Ferrocarriles y (2) BSH, en su división de grupos de electrodomésticos. La estrecha colaboración con estas empresas ha permitido desarrollar los estudios empíricos en entornos industriales reales.

1.4 Metodología de investigación

Para el desarrollo de este trabajo de tesis se ha seguido la estructura metodológica para el desarrollo de tesis doctoral propuesta por R. Wieringa [32]. Se trata de una metodología cíclica e iterativa (ver Figura 1.2) y que consta de las siguientes fases:

5

Procesado de requisitos

Índice

5.1 Resumen del capítulo

La descripción textual de los requisitos es una técnica de especificación que se usa habitualmente en la industria, donde el tiempo es clave para el éxito. La forma en que los requisitos se especifican depende en gran medida de factores humanos: habilidades comunicativas, conocimiento del medio, etc. Con el fin de estudiar cómo el procesado de requisitos se ve afectado por el nivel de detalle utilizado en las descripciones textuales, este capítulo compara especificaciones de requisitos textuales enriquecidas con especificaciones de requisitos textuales no enriquecidas para generar modelos software. Para ello, hemos llevado a cabo un experimento en la industria con 19 ingenieros de la empresa CAF (Construcciones y Auxiliares de Ferrocarril), que es un proveedor de soluciones ferroviarias. El experimento es un diseño *crossover* que analiza eficiencia, eficacia y dificultad percibida a partir de una especificación escrita de requisitos que los sujetos deben procesar para construir modelos de software. Los resultados muestran que eficacia y eficiencia para los requisitos enriquecidos son mejores, mientras que los requisitos no enriquecidos entrañan una pequeña mayor dificultad para ser procesados. Por lo tanto, a pesar de que los requisitos enriquecidos requieren más tiempo para ser especificados, los resultados son mejores al utilizarlos.

5.2 Introducción

La primera medida del éxito de un sistema software es el grado de cumplimiento que alcanza el sistema respecto a las cualidades y objetivos para los que fue creado. "Las cualidades y objetivos para los que fue creado" están especificados en los requisitos software. IEEE define requisito software como [104]:

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
3. Una representación documentada de una condición o capacidad como en (1) o (2).

En términos generales, la Ingeniería de Requisitos *(Requeriments Engineering, RE)* es el proceso de descubrir el propósito de un sistema software, identifi-

cando a los *stakeholders* y sus necesidades, y documentar esta información en una forma que sea susceptible de análisis, comunicación y posterior implementación [117]. En el desarrollo software, tradicionalmente, se le dedica un mayor esfuerzo a los requisitos en las etapas tempranas del desarrollo motivado por la evidencia que los errores debidos a requisitos, como requisitos omitidos o mal interpretados, son más caros de arreglar en etapas posteriores del proyecto software [118].

Habitualmente la elicitación de requisitos es el primer paso en el ciclo de desarrollo software. La información capturada durante la elicitación de requisitos posteriormente será interpretada, analizada, modelada y validada antes de que se considere que el conjunto de requisitos, denominados especificación de requisitos, sea considerada definitiva. El objetivo de la elicitación de requisitos es encontrar que problema necesita ser solucionado e identificar el alcance del sistema a desarrollar [117].

El desarrollo de software basado en la generación de familias de productos es una forma cada vez más importante de actividad de desarrollo. Como ya se ha comentado en el capítulo 2, existe la necesidad de desarrollar una gama de productos de software que comparten similares requisitos y características arquitectónicas, pero difieren en ciertos requisitos clave. El proceso de identificar requisitos para desarrollar arquitecturas que son (a) estables ante la necesidad de cambio, y (b) lo suficientemente flexibles como para ser personalizados y adaptados a los requisitos cambiantes, se ha mostrado como uno de los principales problemas de investigación en Ingeniería de Software [119].

Los requisitos, entre otros objetivos, deben facilitar una comunicación efectiva entre los diferentes *stakeholders* de un proyecto de desarrollo software. Por lo tanto, la forma en la que los requisitos son documentados cobra enorme importancia. Se reconoce como crucial la gestión de requisitos: la capacidad, no solo de escribir requisitos, sino también para hacerlo en una forma que sea legible, comprensible y, en último lugar, genere desarrollos correctos. Para mejorar la legibilidad e interpretación de los requisitos se han desarrollo una gran variedad de estándares para proporcionar directrices para la creación de documentos de requisitos. Una de las formas, por su legibilidad y facilidad de comprensión, más extendida para representar requisitos es la utilización de descripciones textuales. En este capítulo se aborda la influencia de los requisitos en la generación de modelos software en un ciclo de desarrollo software MDD + SPL.

La elicitación de requisitos es un proceso que tiene una gran dependencia de factores humanos. Cappel [120] manifiesta que habilidades no técnicas como

la comunicación oral y escrita, la resolución de problemas y la capacidad de aprendizaje tienen una importancia vital en el proceso de elicitación de requisitos. La forma en que los factores humanos subjetivos pueden afectar el proceso de elicitación se suele tratar en el campo de la psicología y está fuera del alcance de este trabajo.

Existen numerosas técnicas para la elicitación de requisitos, como prototipado, entrevistas estructuradas, observación de realización de tareas, encuestas, tormentas de ideas, técnicas de grupos nominales o *focus group*, entre muchos otros. Existen trabajos teóricos que han estudiado que técnicas son mejores dependiendo del contexto [121]. Establecer que técnica de elicitación de requisitos es mejor desde un punto de vista práctico sigue siendo un estudio desafiante para la comunidad de Ingeniería de Software.

De todas las técnicas existentes, este capítulo se centra en el estudio de la técnica de descripción textual, donde el analista redacta una especificación textual que describe todas las características que el sistema debe cumplir. La descripción proporcionada actúa como entrada para los próximos pasos del proceso de desarrollo. Una descripción textual es subjetiva ya que el grado de detalle de la descripción puede variar según el tema.

El objetivo de este capítulo es analizar qué tipo de descripción de requisitos es mejor: un requisito no enriquecido o un requisito enriquecido . Una descripción de los requisitos no enriquecida contiene exclusivamente lenguaje natural *(Natural Language, NL)*. Por el contrario, la descripción de un requisito enriquecido contiene lenguaje natural que se potencia con pares de elementos propiedad-valor que se incluyen en la descripción.

Este capítulo propone un experimento basado en un diseño cruzado. Los sujetos parten de una descripción textual ya existente de los requisitos y deben procesarlos para crear modelos de software. Los sujetos son ingenieros de software de nuestro socio industrial, CAF (`http://www.caf.net/en`) en dos de sus sedes. Una de sus sedes se encuentra en Zaragoza (España), donde seis sujetos formaron parte del experimento y, en la sede de Beasaín (España), trece sujetos formaron parte del experimento. CAF es un proveedor mundial de trenes. Sus trenes se pueden encontrar en todo el mundo en diferentes formas (trenes regulares, metro, tren ligero, monorraíl, etc.).

El experimento se realizó en términos de tres variables: eficacia (proporción de errores), eficiencia (tiempo empleado) y dificultad percibida (percepción subjetiva del ingeniero de software). Los resultados muestran que existen diferencias significativas entre los requisitos enriquecidos y no enriquecidos para

las citadas variables, lo que demuestra que la eficacia y la eficiencia para los requisitos enriquecidos tienen mejores valores; por el contrario, los requisitos no enriquecidos son más difíciles de tratar que los enriquecidos.

5.3 Background

Como ya ha sido comentado, la ingeniería de requisitos es una de las partes más importantes del proceso de desarrollo de software. El procesado de requisitos ha sido identificado como un factor crucial para el éxito de un proyecto. Esto es particularmente válido en grandes proyectos que requieren un esfuerzo significativo en las etapas iniciales de especificación del producto, donde se compromete una gran proporción del costo del producto. De hecho, cuando se desarrollan productos de alta complejidad en industrias como la aeroespacial, se reconoce ampliamente que casi el 60 % del coste del producto se computa durante el primer 5 % del ciclo de desarrollo del producto [122].

Cuando comenzamos el análisis del proceso de desarrollo software en nuestro socio CAF, identificamos que la compañía trabaja principalmente con dos tipos de descripciones textuales de requisitos (no enriquecidos y enriquecidos):

- **La descripción del requisito no enriquecido** contiene solo lenguaje natural. La siguiente declaración es un ejemplo de un requisito no enriquecido: *El PLC inhibirá el apagado del interruptor de circuito siempre que exista inhibición por parte del control de la ACR, cuando haya alimentación externa o cuando no se otorgue permiso desde ninguna UCU.*
- **El requisito enriquecido** contiene un lenguaje natural que se mejora con los pares propiedad-valor de los elementos del modelo que se incluyen en la descripción (ver Figura 5.1). Un ejemplo de este tipo de requisito es: *El PLC inhibirá el apagado del interruptor automático (AT-INHIB-HSCB = 1) siempre que haya inhibición del control de la ACR (SC-INHIB-HSCB = 1), cuando haya alimentación externa (CV-POWEROUT = 1, -AT-POWEROUT = 1) o cuando no se concede permiso desde ninguna UCU (UCU-Cx-ControlW.b-HSCBPerm = 0).*

Tanto los requisitos enriquecidos como los no enriquecidos son los artefactos de entrada para construir modelos de software que describen la funcionalidad de los trenes de nuestro socio industrial. El lenguaje de modelado utilizado es el Lenguaje de Control y Gestión del Tren *(Train Control and Management Language, TCML)*, que es un lenguaje específico del dominio que tiene tanto

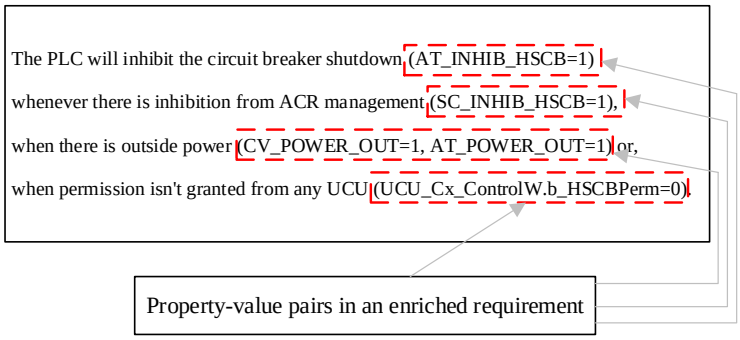


Figura 5.1: Requisito enriquecido con pares propiedad-valor.

la expresividad requerida para describir la interacción entre los elementos principales de un tren como la expresividad requerida para especificar aspectos no funcionales que están relacionados a la regulación.

Nuestro objetivo es analizar cuál de los dos tipos de descripciones textuales de requisitos es mejor para construir modelos de software en un entorno industrial. Por esta razón, llevamos a cabo un experimento en la empresa CAF, donde los sujetos deben construir un modelo de software de acuerdo con la descripción de un requisito. Nuestro objetivo es comparar los resultados utilizando requisitos enriquecidos versus requisitos no enriquecidos.

5.4 Diseño del experimento

Como ya ha sido comentado en la sección 4.4 del capítulo 4 cada uno de los estudios empíricos desarrollados para al realización de este trabajo de tesis es representado con un modelo de características que refleja las propiedades del citado estudio. La Figura 5.2 muestra, destacando con un fondo gris, todas aquellas características que forman parte del estudio, algunas características incluidas son *Experimento* o *Pregunta de investigación*. Por contra las características no presentes en este estudio han sido "suavizadas" como *Estudiante* o *Estudio de caso*. En este apartado se va a describir con detalle el diseño del experimento.

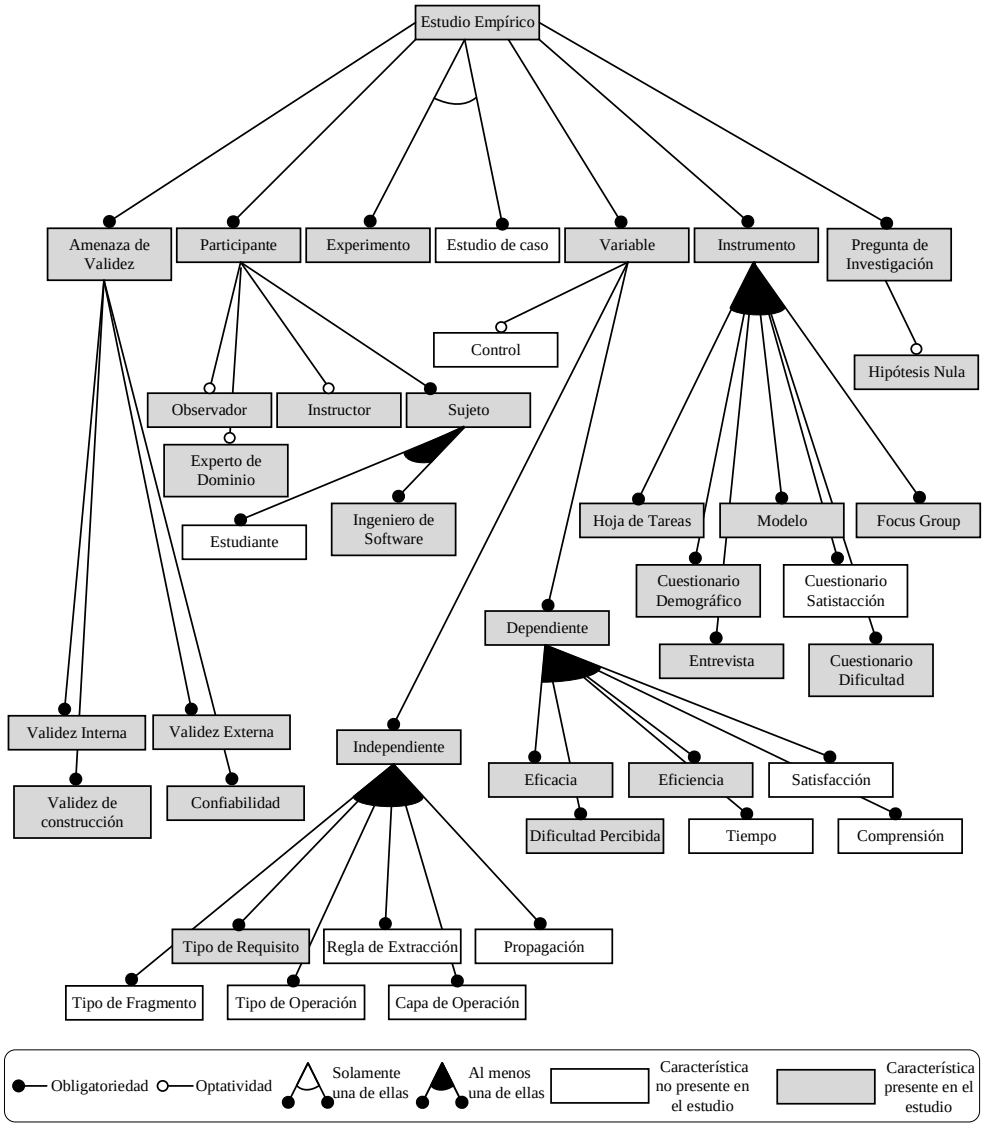


Figura 5.2: Modelo de características del experimento para evaluar la gestión de requisitos.

5.4.1 Objetivo

Nuestro socio industrial CAF, dedicado a la fabricación de diversos tipos de trenes, utiliza como *inputs* iniciales en el proceso de generación de modelos software dos tipos de requisitos (enriquecidos y no enriquecidos). Para cualquier proceso de generación de modelos software, incluyendo el llevado a cabo en CAF, cobra importancia conocer si la utilización de uno u otro tipo de requisitos tiene influencia en cuestiones como la productividad, calidad de los modelos generados o esfuerzo realizado por los ingenieros en el proceso. Para conocer que influencia tiene uno u otro tipo de requisitos se ha diseñado un experimento donde se ha medido eficacia, eficiencia y dificultad percibida.

El objetivo de la investigación descrita en este capítulo es analizar cuál de los dos tipos de descripción textual, requisitos enriquecidos o no enriquecidos, es mejor como punto de partida para construir modelos de software en un entorno industrial. Siguiendo las recomendaciones de Wohlin [123], el objetivo de nuestra investigación fue:

Analizar el rendimiento de los ingenieros de software cuando procesan los requisitos;

Con el propósito de llenar la ausencia la evaluación empírica sobre este tema;

Con respecto a los diferentes niveles de enriquecimiento de requisitos;

Desde el punto de vista de ingenieros de software;

En el contexto de una compañía de fabricación de trenes.

Las medidas utilizadas en nuestra investigación para lograr el objetivo anterior son la eficacia, la eficiencia y la dificultad percibida. La eficacia y la eficiencia son medidas ampliamente aceptadas para medir el rendimiento de los ingenieros de software [30, 124]. Además, la dificultad percibida es uno de los modelos teóricos más aplicados al analizar la aceptación del usuario en la comunidad de investigación de Ingeniería de Software [29, 80]. Utilizando estas medidas buscamos respuesta a las siguientes tres preguntas de investigación:

RQ1 ¿Existen diferencias cuando se usan diferentes niveles de enriquecimiento de requisitos con respecto a la eficacia en la creación de modelos de software?

RQ2 ¿Existen diferencias cuando se usan diferentes niveles de enriquecimiento de requisitos con respecto a la eficiencia en la creación de modelos de software?

RQ3 ¿La dificultad percibida es diferente cuando los ingenieros de software usan diferentes niveles de enriquecimiento de requisitos para construir modelos de software?

Para responder a las anteriores preguntas de investigación, hemos formulado las siguientes hipótesis nulas:

- $\mathbf{H_{01}}$: No hay diferencia en la eficacia del rendimiento de los ingenieros de software cuando se trabaja con requisitos con diferentes niveles de enriquecimiento.
- $\mathbf{H_{02}}$: No hay diferencia en la eficiencia del rendimiento de los ingenieros de software cuando se trabaja con requisitos con diferentes niveles de enriquecimiento.
- $\mathbf{H_{03}}$: No existe diferencia en la dificultad percibida de los ingenieros de software cuando se trabaja con requisitos con diferentes niveles de enriquecimiento.

5.4.2 Participantes

Los sujetos fueron 19 ingenieros de software de la compañía CAF. Estos ingenieros son expertos en desarrollar software y requisitos. En su trabajo diario, estos ingenieros desarrollan software a partir de requisitos enriquecidos y no enriquecidos. Seis ingenieros de software trabajan en la sede de la compañía en Zaragoza y trece ingenieros de software trabajan en la sede de la compañía en Beasaín. En su vida profesional han pasado de 1 a 15 años trabajando como ingenieros de software (una media de 6.65 años). Afirmaron que en su jornada laboral pasan un promedio de 3.68 horas diarias desarrollando software. Por otro lado, el tiempo promedio desarrollando requisitos era de 3.36 horas diarias.

Además de los sujetos, también participaron un instructor, dos observadores y un experto en el dominio. El instructor proporcionó información sobre la realización de los ejercicios, aclaró dudas durante el experimento y gestionó los *focus group*. Los observadores tomaron notas para análisis posteriores. Finalmente, el experto de dominio supervisó y creo los enunciados de los requisitos, diseñó la corrección, corrigió los ejercicios y resolvió las dudas sobre los enunciados de requisitos para los sujetos durante el experimento. El experto de dominio no formó parte del artículo de investigación al que dio lugar el experimento.

5.4.3 Definición de variables

En esta subsección se enumeran y definen todas las variables relevantes en el experimento presentado en este capítulo.

Variables independientes

Realizamos un experimento de factor único donde la variable independiente es el nivel de enriquecimiento del requisito, que es una variable nominal con dos valores: requisitos enriquecidos y requisitos no enriquecidos. Se puede encontrar una explicación detallada del significado de estos valores en la sección 5.3.

Variables dependientes

En nuestro experimento, los ingenieros de software tuvieron que realizar cuatro ejercicios, donde cada ejercicio contenía una descripción de los requisitos. El resultado de cada ejercicio fue un modelo de software que utilizaba conceptos conocidos por los sujetos. Durante la transformación de la especificación de requisitos al modelo de software, medimos la eficacia, la eficiencia y la dificultad percibida. Las variables dependientes se definen como:

- La eficacia se define como el porcentaje de un ejercicio realizado correctamente por el ingeniero de software. Los ejercicios son descompuestos por un experto en dominio en una serie de pasos, y cada paso tiene un porcentaje ponderado con respecto a todo el ejercicio.
- La eficiencia es la relación entre la eficacia y el tiempo empleado (en minutos) para realizar el ejercicio.
- La dificultad percibida es la percepción que tiene un ingeniero de software de la complejidad de un ejercicio. Se mide usando una escala *Likert*. Los ingenieros de software deben completar un valor para la dificultad percibida para cada ejercicio.

5.4.4 Instrumentos

En esta subsección se muestra una descripción de todos los instrumentos utilizados para el desarrollo del experimento.

Cuestionario demográfico

El cuestionario demográfico está compuesto por un conjunto de preguntas que permiten identificar el perfil de cada sujeto participante en el experimento. La información solicitada en el cuestionario a cada sujeto es: nivel de educación, tiempo de trabajo en su departamento actual (en años), edad, género, tiempo dedicado al desarrollo de software (diariamente) y tiempo dedicado al desarrollo de requisitos en su jornada diaria.

Hoja de tareas

A cada sujeto participante en el experimento se le proporcionó una hoja de tareas por cada requisito. Cada hoja de tareas contiene una descripción del requisito, una escala *Likert*, un área de texto y dos campos de texto. La descripción del requisito contiene el requisito que debe ser procesado. Los sujetos también tuvieron que completar el valor dentro de la escala de *Likert*, la escala comprende valores entre 1 y 7: 1 es muy fácil hasta 7 considerado muy difícil. Este valor de la escala se usa para calcular la dificultad percibida. En el área de texto, los sujetos podían escribir cualquier opinión que tuvieran sobre el proceso de generación de modelos software a partir de requisitos. Finalmente, los sujetos tuvieron que anotar tanto el tiempo de inicio de realización del ejercicio para generar los modelos software a partir de los requisitos, como el tiempo de finalización del ejercicio. Estos tiempos permiten el cálculo de la eficiencia.

Modelo software

Un modelo software es el resultado de cada ejercicio cuando el sujeto procesa los requisitos. Los modelos software resultantes son corregidos por un experto de dominio, que determina el porcentaje de pasos que se realizan correctamente dentro de cada tarea. El porcentaje de pasos correctos proporciona el cálculo de eficacia y eficiencia. La Figura 5.3 muestra una representación en árbol del modelo software y en la página web `https://www.youtube.com/watch?v=Ypcl2evEQB8` puede observarse la sintaxis concreta del modelo software.

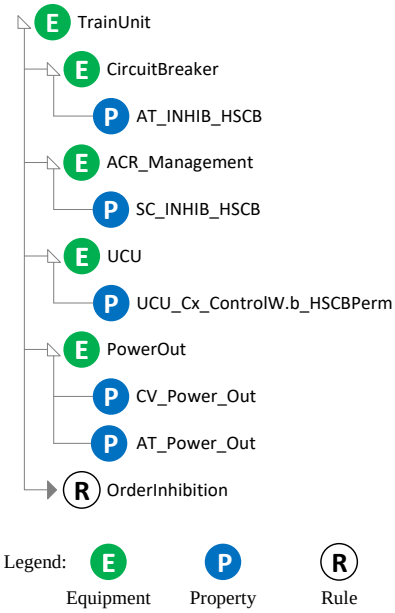


Figura 5.3: Modelo software.

Focus Group

El objetivo de realizar un *focus group* fue obtener datos cualitativos de los comentarios de los sujetos [125]. Este *focus group* está compuesto por preguntas abiertas. El objetivo de estas preguntas es detectar los conceptos o procesos en la realización del ejercicio que son más problemáticos para los sujetos, así como determinar las causas reales de los problemas. Los materiales utilizados en este experimento están disponibles en la página web `http://svit.usj.es/requerimentinfluenceexperiment`.

5.4.5 Procedimiento experimental

Para desarrollar el experimento seleccionamos un diseño cruzado donde los sujetos aplicaban aleatoriamente dos tratamientos (requisitos enriquecidos y no enriquecidos). Cada sujeto debe construir cuatro modelos software a partir de cuatro requisitos: dos requisitos enriquecidos y otros dos requisitos no enriquecidos. Al aplicar aleatoriamente los tratamientos, nueve sujetos comenzaron la sesión con el tratamiento de los requisitos enriquecidos y diez sujetos comenza-

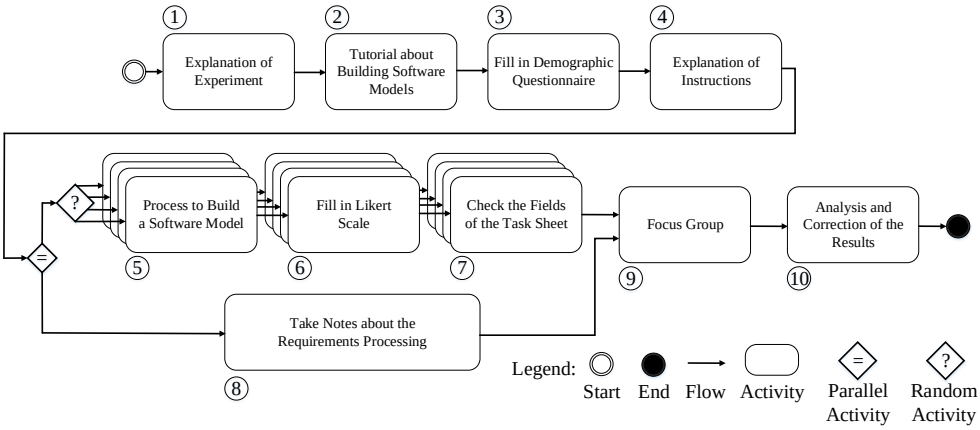


Figura 5.4: Procedimiento experimental.

ron la sesión con el tratamiento de los requisitos no enriquecidos. Las ventajas del diseño seleccionado son:

1. Utiliza el tamaño de muestra más grande posible debido a medidas repetidas.

2. El efecto de aprendizaje se minimiza ya que los tratamientos se alternan.

3. Es fácil de justificar en un entorno industrial donde ambos tratamientos se usan con frecuencia [126].

Con la finalidad de comprobar el correcto diseño del experimento, llevamos a cabo un estudio piloto con un único participante [127]. Este participante, posteriormente, no formó parte en el experimento. Como resultado del estudio piloto, decidimos reducir el número de ejercicios debido a la cantidad de tiempo necesaria para procesar todos los requisitos del estudio piloto. Inicialmente, el experto de dominio diseñó trece requisitos para el experimento. Tras realizar el estudio piloto, el experto de dominio redujo el conjunto de ejercicios a cuatro requisitos representativos. El criterio para seleccionar los cuatro requisitos fue elegir los más importantes para la especificación del sistema de software.

El experimento se realizó en dos días diferentes. El primer día se realizó en la sede de la CAF en Zaragoza con un grupo de seis ingenieros de software. El segundo día, el experimento fue replicado en la sede de la CAF en Beasaín con trece ingenieros de software. En este último caso, el experimento fue realizado por tres grupos de ingenieros de software según su disponibilidad horaria. En el

primer y segundo grupo, había cinco ingenieros de software; en el tercer grupo había tres ingenieros de software. El procedimiento (ver Figura 5.4) para todos los sujetos fue el siguiente:

1. Los sujetos recibieron información sobre el desarrollo del experimento. Se les comunicó que no se trataba de una prueba de sus habilidades.

2. Los sujetos recibieron un tutorial sobre cómo construir los modelos software acorde a los requisitos. Este tutorial fue enseñado por el instructor. El tiempo promedio dedicado a este tutorial fue de 24 minutos. Los sujetos podían desarrollar un ejemplo durante la explicación de este tutorial. Las copias impresas de las diapositivas que se usaron en el tutorial se les dieron a los sujetos y estuvieron disponibles para ellos durante el experimento.

3. Se solicitó a los sujetos que completaran un cuestionario demográfico antes de realizar las tareas del experimento.

4. A los sujetos se les dio una serie de instrucciones claras para procesar los requisitos y para completar la hoja de tareas.

5. Se pidió a los sujetos que interpretaran cuatro requisitos (dos requisitos enriquecidos y dos requisitos no enriquecidos). Como resultado de esta interpretación, los sujetos tuvieron que construir un modelo de software que expresara todas las ideas articuladas en los requisitos. Este modelo se utilizó para calcular la eficacia y la eficiencia del proceso de interpretación de los requisitos. Para evitar un posible efecto límite *(celling effect)*, no hubo un límite de tiempo para interpretar los requisitos. Por otro lado, los dos tipos de requisitos se asignaron a los sujetos aleatoriamente para evitar el efecto de aprendizaje.

6. Se solicitó a los sujetos que completaran una escala *Likert* sobre la dificultad percibida para cada requisito. Estas respuestas se usaron para calcular la dificultad percibida sobre la comprensión de los requisitos.

7. Cuando un sujeto terminaba el desarrollo de un requisito, antes de pasar a desarrollar el siguiente, un observador verificaba que el sujeto hubiera completado todos los campos en la hoja de tareas de una forma correcta.

8. Los observadores tomaron notas sobre los comentarios y dudas de los sujetos al procesar los requisitos.

9. El instructor realizó un *focus group* sobre los ejercicios con cada grupo de sujetos.

10. Finalmente, el experto de dominio corrigió los modelos y los observadores analizaron los resultados. El porcentaje de ejercicios completados con éxito proporcionó un valor para la eficacia.

5.5 Resultados

Para nuestro diseño de experimento, la prueba estadística más adecuada es el modelo lineal mixto (*Linear Mixed Model)* [128]. Las variables dependientes para esta prueba son eficacia, la eficiencia y dificultad percibida. El factor fijo (*Fixed Factor*) es el método, y el factor aleatorio (*Random Factor*) es el sujeto, ya que necesitamos representar las medidas para cada sujeto. Las conclusiones extraídas del modelo lineal mixto son compatibles con los datos descriptivos a través de diagramas de caja y bigotes (*box-and-whiskers plots*) e histogramas.

El uso del análisis estadístico mediante modelo mixto lineal implica la suposición de que los residuos deben distribuirse normalmente. Con los datos recopilados, todos los residuos obtienen un valor de p que es mayor que 0.05 con la prueba K-S, lo que significa que los residuos se distribuyen normalmente. El tamaño del efecto muestra la magnitud de las diferencias para cada factor. Por lo general, se aplica cuando las hipótesis nulas son rechazadas al estudiar el nivel de la significancia de las diferencias de las medias de los tratamientos. Calculamos el tamaño del efecto a través de Cohen d, que describe la proporción de la variabilidad en la medida dependiente atribuible a un factor. La interpretación más común es la siguiente: entre 0.2 y 0.3 es un efecto pequeño; alrededor de 0.5 es un efecto medio; y más de 0.8 es un efecto grande [129].

5.5.1 Eficacia

Se realizó una prueba del modelo lineal mixto para comparar la eficacia con los dos tipos diferentes de requisitos (enriquecidos y no enriquecidos). Los resultados muestran que existe un efecto significativo de los diferentes requisitos sobre la eficacia [$F(2,18) = 559.78$ $p = 0.000$]. Dado que el valor p es menor a 0.05, podemos concluir que hay diferencias significativas entre los dos tratamientos. El tamaño del efecto de 0.601 muestra que la magnitud de esta diferencia es media.

La Figura 5.5 muestra el diagrama de cajas y bigotes para la variable de respuesta *eficacia*. Se puede observar que el valor para los requisitos enriquecidos es mejor que para los requisitos no enriquecidos. La mediana, el primer cuartil y el tercer cuartil obtienen mejores valores para los requisitos enriquecidos.

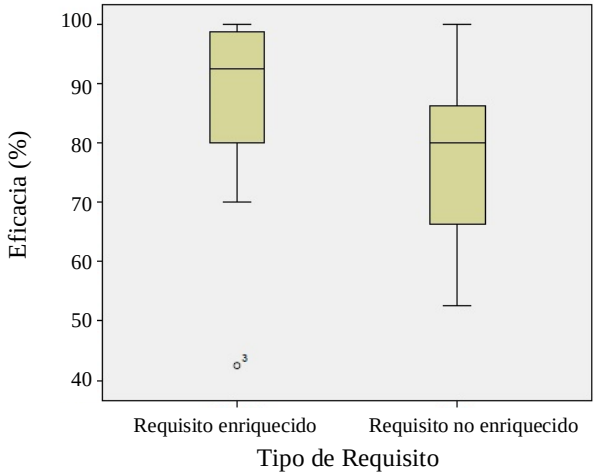


Figura 5.5: Diagrama de caja para la eficacia con requisitos enriquecidos y no enriquecidos.

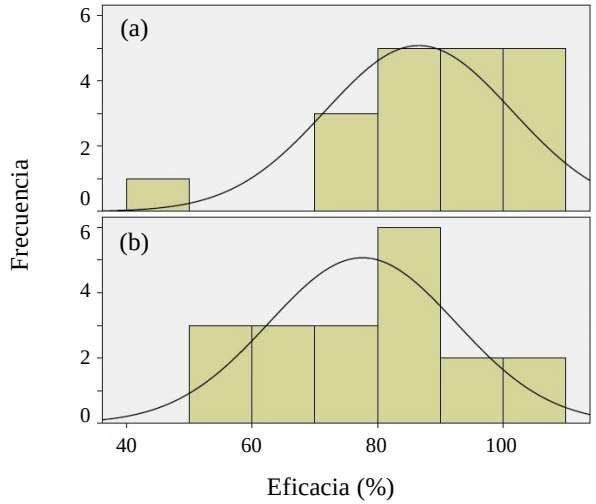


Figura 5.6: (a) Histograma para la eficacia con requisitos enriquecidos. (b) Histograma para la eficacia con requisitos no enriquecidos.

La media de eficacia para los requisitos enriquecidos (86.578 %) también es mejor que para los requisitos no enriquecidos (77.605 %). La Figura 5.6 (a) muestra el histograma para la eficacia con requisitos enriquecidos, y la Figura 5.6 (b) muestra el histograma para la eficacia con requisitos no enriquecidos. Es destacable que una eficacia del 100 % es más frecuente en los requisitos enriquecidos que en los requisitos no enriquecidos. Hay cinco muestras del 100 % de eficacia cuando se usan requisitos enriquecidos, y dos muestras cuando se usan requisitos no enriquecidos. La curva normal muestra que los valores de eficacia tienden a ser más altos (alrededor del 80 %) con requisitos enriquecidos.

De acuerdo con nuestro análisis, podemos afirmar que existen diferencias significativas entre los requisitos enriquecidos y los no enriquecidos en términos de eficacia. Los valores para requisitos enriquecidos son mejores que los valores para requisitos no enriquecidos. Por lo tanto, rechazamos H_{01}, que afirma que la eficacia cuando se trabaja con requisitos enriquecidos es la misma que cuando se trabaja con requisitos no enriquecidos.

5.5.2 Eficiencia

Los resultados del análisis con el modelo lineal mixto muestran que hay un efecto significativo de los dos tipos diferentes de requisitos sobre la eficiencia [F(2,18) = 56.574 p = 0.000] ya que el valor p es menor que 0.05. El tamaño del efecto de 0.627 es medio, lo que significa que esta diferencia entre los tratamientos es considerable.

La Figura 5.7 muestra el diagrama de cajas y bigotes para la variable de respuesta *eficiencia*. La mediana, el primer cuartil y el tercer cuartil obtienen mejores valores para los requisitos enriquecidos. Además, la media para los requisitos enriquecidos (7.969 %/min) es más alta que para los requisitos no enriquecidos (5.434 %/min). La Figura 5.8 (a) muestra el histograma para la eficiencia con requisitos enriquecidos, y la Figura 5.8 (b) muestra el histograma para la eficiencia con requisitos no enriquecidos. Se puede observar que la mayoría de las muestras para requisitos no enriquecidos (12 muestras) están entre 2.5 %/min y 5 %/min; por otro lado, el intervalo con el mayor número de muestras para requisitos enriquecidos (8 muestras) es entre 7.5 %/min y 10 %/min. La curva normal muestra que los valores de eficiencia tienden a estar de alrededor del 8 %/min.

De acuerdo con nuestro análisis, podemos afirmar que existen diferencias significativas entre los requisitos enriquecidos y los no enriquecidos en términos de eficiencia. Los valores para requisitos enriquecidos son mejores que los valo-

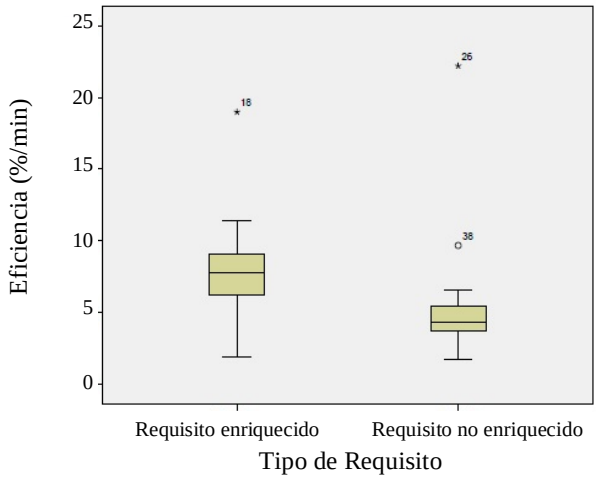


Figura 5.7: Diagrama de caja para la eficiencia con requisitos enriquecidos y no enriquecidos.

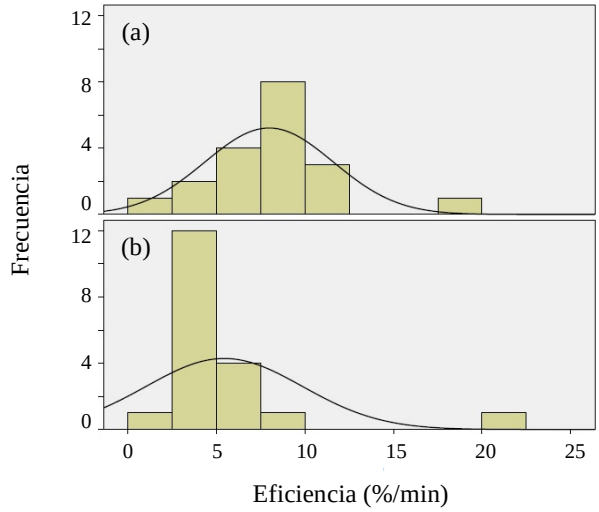


Figura 5.8: (a) Histograma para la eficiencia con requisitos enriquecidos. (b) Histograma para la eficiencia con requisitos no enriquecidos.

res para requisitos no enriquecidos. Por lo tanto, rechazamos H_{02}, que afirma que la eficiencia cuando se trabaja con requisitos enriquecidos es la misma que cuando se trabaja con requisitos no enriquecidos.

5.5.3 Dificultad percibida

Se realizó un análisis con el modelo mixto lineal para comparar la dificultad percibida de los sujetos con los dos tipos diferentes de requisitos. Hay un efecto significativo de los diferentes requisitos sobre la dificultad percibida ya que el valor de p es menor a 0.05 [F(2,18) = 185.77 p = 0.000]. El tamaño del efecto de 0.102 muestra que la magnitud de esta diferencia es pequeña.

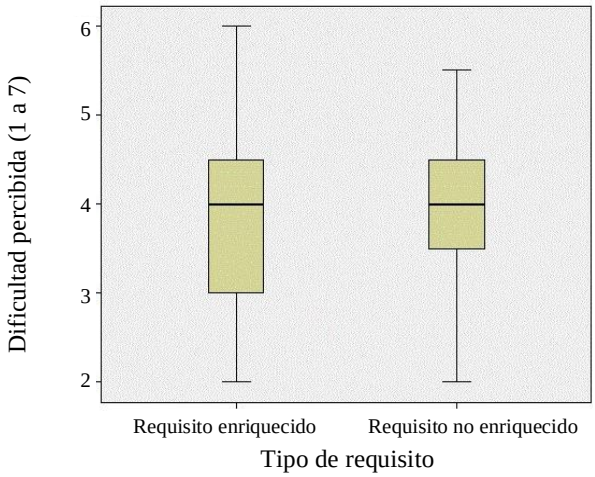


Figura 5.9: Diagrama de caja para la dificultad percibida con requisitos enriquecidos y no enriquecidos.

La Figura 5.9 muestra el diagrama de cajas y bigotes para la variable respuesta *dificultad percibida*. La diferencia entre las medias de los dos tratamientos es pequeña, la media de la dificultad percibida para los requisitos enriquecidos es 3.868, y el promedio de los requisitos no enriquecidos es 3.974. La Figura 5.10 (a) muestra el histograma para la dificultad percibida con requisitos enriquecidos, y la Figura 5.10 (b) muestra el histograma para la dificultad percibida con requisitos no enriquecidos. Destaca que el valor de las medias de los dos tratamientos están muy próximos. Por otro lado, el intervalo con el mayor número de muestras (intervalo alrededor de 4) es el mismo para ambos tratamientos. Hay cuatro muestras en este intervalo para requisitos enriquecidos y seis muestras para requisitos no enriquecidos.

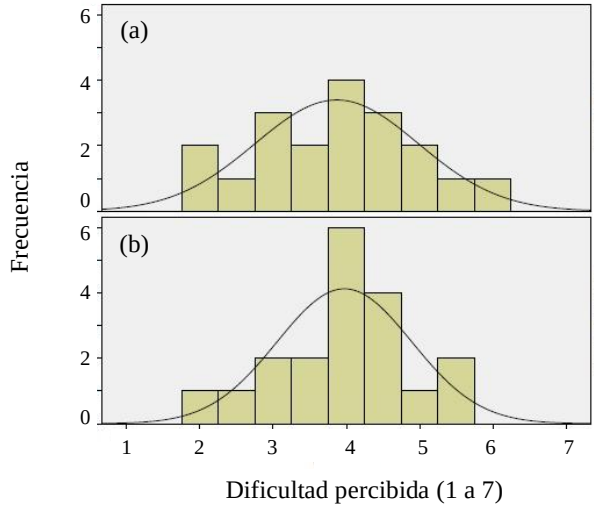


Figura 5.10: (a) Histograma para la dificultad percibida con requisitos enriquecidos. (b) Histograma para la dificultad percibida con requisitos no enriquecidos.

De acuerdo con nuestro análisis, podemos afirmar que existen diferencias significativas entre los requisitos enriquecidos y los requisitos no enriquecidos en términos de dificultad percibida. Los valores para requisitos enriquecidos son mejores que los valores para requisitos no enriquecidos. Por lo tanto, rechazamos H_{03}, que afirma que la dificultad percibida al trabajar con requisitos enriquecidos es la misma que cuando se trabaja con requisitos no enriquecidos.

5.6 Discusión

El *focus group*, realizado en la parte final del experimento, y los resultados presentados en la sección anterior nos permiten valorar el desarrollo de modelos software a partir de requisitos no enriquecidos y enriquecidos en términos de eficacia, eficiencia y dificultad percibida de la siguiente forma:

Eficacia: A diferencia de los requisitos no enriquecidos, los requisitos enriquecidos incluyen pares propiedad-valor específicos que los ingenieros de software pueden usar durante la construcción de un modelo a partir de una especificación de requisitos. Sin embargo, los resultados de nuestro experimento muestran que las propiedades y valores incluidos en un requisito enriquecido no siempre coinciden con las propiedades y valores que los ingenieros introducen en los

modelos. En algunos casos, los ingenieros de software cambiaron los nombres de las propiedades, el número de propiedades utilizadas y los valores porque necesitaban realizar operaciones intermedias para crear el modelo software asociado al requisito.

Aunque, en ocasiones, los ingenieros de software no incluyeron las propiedades y los valores en los modelos tal como figuraban en el requisito, detectamos que los ingenieros de software usaron estas propiedades y valores como puntos de control, lo que les permitió verificar si el requisito se había desarrollado por completo. En otras palabras, los requisitos enriquecidos tenían n propiedades que los ingenieros de software consideraban como n puntos de control, mientras que los requisitos no enriquecidos se consideraban como un todo. Estos puntos de control ayudaron a los ingenieros de software a generar los modelos software de forma completa desde el requisito.

Eficiencia: Cuando se les preguntó sobre el uso de pares propiedad-valor para interpretar los requisitos, los ingenieros de software respondieron que los pares propiedad-valor no aumentaban la dificultad y servían a los ingenieros de CAF como guía para transformar los requisitos a modelos. En otras palabras, (1) el lenguaje natural brinda flexibilidad a los ingenieros, (2) el lenguaje natural permite a CAF compartir los requisitos con sus clientes, y (3) los pares propiedad-valor mejoran la eficiencia de los ingenieros de software.

Dificultad percibida: Nuestros resultados mostraron que la diferencia en la dificultad percibida es pequeña entre los requisitos no enriquecidos y enriquecidos. La principal dificultad en los requisitos no enriquecidos es conocer su detalle, mientras que la dificultad en los requisitos enriquecidos es comprender los pares propiedad-valor. Los resultados están alineados con las respuestas que obtuvimos durante *focus group*, donde el conocimiento adquirido indicó que los ingenieros de software percibieron una dificultad similar entre los requisitos no enriquecidos y enriquecidos. De hecho, la empresa no requirió a sus ingenieros que utilizaran un tipo específico de requisito ni que tuvieran un consenso sobre qué tipo de requisito era más beneficioso que el otro. Sin embargo, la eficacia, la eficiencia y la dificultad percibida mostraron una diferencia significativa a favor de los requisitos enriquecidos, lo que motiva su uso en el futuro.

5.7 Amenazas a la validez

Esta sección describe las amenazas que no pudimos evitar pero que mitigamos y las amenazas que no pudimos abordar. Usamos la clasificación de amenazas a la validez de [116]; esta clasificación distingue cuatro aspectos de validez que serán descritos en las siguientes subsecciones.

5.7.1 Validez de construcción

Este tipo de validez refleja como las medidas que se estudian representan lo que los investigadores tienen en mente y lo que se investiga en base a las preguntas de investigación.

- Sesgo del autor: esta amenaza significa que las personas que definen los artefactos pueden influir subjetivamente en la obtención de los resultados que están siendo buscados. Para mitigar esta amenaza, los ejercicios y respuestas fueron diseñados por un experto en el dominio que fue externo al diseño del experimento y que no participó en este documento. El citado experto ha desarrollado herramientas de modelado en entornos industriales (en el dominio de la placas de inducción y el dominio del software de control de trenes).
- Diseño de tareas: esta amenaza aparece cuando las tareas se pueden realizar correctamente por casualidad. Para mitigar esta amenaza, los ejercicios propuestos no tenían una respuesta verdad/falso; los sujetos tuvieron que construir un modelo. Es muy difícil para los sujetos responder correctamente si no entienden el ejercicio. Por otro lado, los enunciados de los requisitos eran requisitos reales que se extrajeron del catálogo de nuestro socio industrial CAF.
- Sesgo monomodo: esta amenaza se debe a la utilización de un solo tipo de medida [126]. Las medidas de dificultad percibida, eficacia y eficiencia se vieron afectadas por esta amenaza. Para mitigar esta amenaza para las mediciones de eficacia y eficiencia, mecanizamos estas mediciones tanto como fue posible por medio de la descomposición del ejercicio.
- Adivinación de hipótesis: esta amenaza significa que el sujeto puede adivinar las hipótesis y trabajar para cumplirlas. Para mitigar esto, no hablamos con los sujetos sobre las preguntas de investigación o el objetivo del experimento.

- Aprehensión de evaluación: esta amenaza aparece cuando los sujetos temen ser evaluados. Para mitigar esta amenaza, el instructor les dijo a todos los sujetos que no se trataba de una prueba sobre sus habilidades.
- Interacción de diferentes tratamientos: esta amenaza aparece cuando se aplican varios tratamientos al mismo tiempo. Para resolver esta amenaza, los tratamientos se aplicaron de forma aleatoria.
- Sesgo de operación simple: esta amenaza aparece cuando los tratamientos dependen de una sola herramienta. El experimento se vio afectado por esta amenaza ya que trabajamos con una herramienta MDD + SPL única para cada ejercicio. Por esta razón, la generalización de los resultados debe hacerse con precaución.

5.7.2 Validez interna

Este tipo de validez aparece cuando se examinan las relaciones causales. Existe el riesgo de que las variables de respuesta estudiadas sean afectadas por otros factores que no se consideran en el experimento.

- Efecto de aprendizaje: esta amenaza aparece cuando los sujetos aprenden algo durante el experimento que puede influir en tareas posteriores. Mitigamos esta amenaza aleatorizando el orden de realización de los ejercicios.
- Intercambio de información: dado que el experimento se diseñó para tener lugar en dos sesiones, los sujetos pudieron haber podido intercambiar información durante el tiempo entre las sesiones. Esto se minimizó porque el experimento se realizó en dos ubicaciones diferentes en dos días sucesivos.
- Comprensibilidad: esta amenaza aparece cuando los sujetos no entienden cómo proceder para realizar el experimento. Esta amenaza fue mitigada al escribir los materiales experimentales en la lengua materna de los sujetos. Además, el instructor explicó antes del experimento un tutorial sobre cómo construir los modelos de software de acuerdo con los requisitos.
- Efectos de fatiga: esta amenaza aparece cuando los sujetos se cansan durante el experimento. Esto se resolvió calculando un tiempo total de 90 minutos para todo el experimento (incluida la preparación). Este tiempo resultó lo suficientemente corto en comparación con el tiempo de trabajo usual en una jornada laboral de los sujetos.

- Motivación del sujeto: esta amenaza aparece cuando los sujetos no están motivados para participar en el experimento. El experimento se vio afectado por esta amenaza ya que los sujetos fueron reclutados como parte de su trabajo diario (no fueron voluntarios).
- Selección de los sujetos: esta amenaza aparece cuando los resultados del experimento pueden depender del tipo de sujetos. El experimento se vio afectado por esta amenaza ya que todos los sujetos fueron reclutados de la compañía CAF.

5.7.3 Validez externa

Este tipo de validez se refiere a la medida en que es posible generalizar los hallazgos y en qué medida los hallazgos son relevantes para otros casos.

- Potencia estadística: esta amenaza aparece cuando el número de sujetos no es suficiente para generalizar los resultados. Nuestro experimento se vio afectado por esta amenaza, porque el número de sujetos (19) no fue lo suficientemente alto como para generalizar los resultados. Sin embargo, es importante tener en cuenta que el papel de los sujetos (ingenieros de software en un entorno industrial) hace una contribución interesante en un área donde la mayoría de los experimentos se llevan a cabo con estudiantes. Por otro lado, hemos utilizado un intervalo de confianza en el que las conclusiones son 95 % representativas. Esto significa que si siguieron una distribución normal, los resultados serían verdaderos el 95 % del tiempo.
- Influencia del dominio: esta amenaza aparece cuando los resultados dependen de un dominio específico. Este experimento se vio afectado por esta amenaza ya que solo analizamos el dominio del sector ferroviario.

5.7.4 Confiabilidad

Este tipo de validez se refiere a la medida en que los datos y el análisis dependen de los investigadores específicos que han desarrollado el experimento.

- Recolección de datos: esta amenaza aparece cuando la recopilación de datos no se realiza de la misma manera en las diferentes sesiones. Esto se mitigó aplicando el mismo procedimiento a cada sesión y usando la misma fórmula para calcular los valores de las variables dependientes.

- Datos de finalización: esta amenaza aparece cuando faltan algunos datos después del proceso de recopilación de datos. Para mitigar esta amenaza, dos observadores probaron la coherencia de los datos cuando los sujetos terminaron cada ejercicio porque los propios sujetos escribieron los datos utilizados en las métricas del experimento.

5.8 Conclusión

Las técnicas para especificar requisitos siguen siendo un tema clave en la comunidad de la Ingeniería de Software. Los estudios empíricos deben llevarse a cabo en la industria ya que el contexto no es el mismo que en la academia. En este trabajo, llevamos a cabo un experimento cruzado para analizar que tipo de descripción de requisitos textuales es mejor para la especificación de requisitos: un requisito no enriquecido o un requisito enriquecido.

Los sujetos del experimento fueron 19 ingenieros de software de CAF, un proveedor mundial de soluciones ferroviarias. Los sujetos partieron de una descripción textual ya existente de requisitos (no enriquecidos o enriquecidos) que debían procesar para crear modelos software. Medimos tres variables: eficacia (proporción de errores), eficiencia (tiempo empleado) y dificultad percibida (subjetividad de los ingenieros de software al realizar las tareas). Nuestros resultados mostraron que existen diferencias significativas entre los requisitos enriquecidos y no enriquecidos para estas tres variables, lo que demuestra que la eficacia y la eficiencia para los requisitos enriquecidos tienen mejores valores, mientras que los requisitos no enriquecidos son ligeramente más difíciles de tratar que los enriquecidos. Además, detectamos que los sujetos utilizaban los pares de propiedad-valor de los requisitos enriquecidos como puntos de control que debían verificar para determinar si el requisito se había procesado por completo. Se requirió más tiempo para procesar los requisitos enriquecidos que los requisitos no enriquecidos, pero influyó positivamente en la eficacia y la eficiencia.

Como trabajo futuro, planeamos replicar este experimento en más compañías para generalizar los resultados, independientemente del perfil de los sujetos. Además, también planeamos analizar los requisitos enriquecidos para construir modelos software así como para generar código.

6

Usabilidad

Índice

6.1 Resumen del capítulo

Common Variability Language (CVL) es una propuesta reciente del *Object Management Group (OMG)* para el modelado de la variabilidad. CVL modela la variabilidad en términos de fragmentos de modelo *(Model Fragments)*. La usabilidad es un criterio de calidad ampliamente reconocido, esencial para garantizar el uso exitoso de herramientas que ponen las ideas anteriores en práctica. Considerando la necesidad de evaluar la usabilidad de las herramientas de modelado con CVL, este capítulo presenta una evaluación de usabilidad de CVL aplicada a una herramienta de modelado para la generación el código de *firmware* de placas de inducción. Esta evaluación aborda las tareas de configuración, ámbito y visualización. La evaluación se desarrolló con usuarios finales de la herramienta, los citados usuarios son ingenieros de nuestro socio industrial en el ámbito de las placas de inducción. Los resultados de eficacia y eficiencia indican que la configuración de modelos en términos de sustituciones de fragmentos del modelo es lo suficientemente intuitiva, pero tanto el ámbito como la visualización requieren un mejor soporte de la herramienta. Los resultados también nos permitieron identificar una lista de problemas de usabilidad que pueden contribuir a minimizar los problemas de ámbito y visualización en CVL.

6.2 Introducción

Common Variability Language (CVL) fue propuesto por el consejo de arquitectura del OMG como estándar de modelado de la variabilidad (*Variability Modeling*) [130]. CVL expresa la variabilidad entre modelos en términos de fragmentos de modelo *(Model Fragments)* conocidos como *Placement Fragments* (puntos de variación) y *Replacement Fragments* (variantes). La materialización de modelos de producto se realiza por medio de sustituciones de fragmentos *(Fragment Substitutions)* entre un modelo base *(Placements)* y una biblioteca de modelos *(Replacements)*.

CVL ha cobrado impulso como lenguaje independiente del dominio para especificar y resolver la variabilidad [131, 132, 80]. Aunque parece que las ideas que subyacen en CVL permiten afrontar las principales características exigibles en una herramienta de modelado de variabilidad como son configuración, ámbito y visualización. No tenemos constancia de la existencia de estudios empíricos que respondan a esta cuestión.

La usabilidad es un criterio de calidad ampliamente reconocido, esencial para garantizar el uso con éxito de herramientas que ponen en práctica las ideas citadas en párrafos anteriores. Este apartado presenta una evaluación de usabilidad de una herramienta de modelado *(Modeling Tool, MT)* aumentada con CVL (MT+CVL). La pregunta de investigación abordada por esta evaluación es: ¿Las herramientas de modelado aumentadas con CVL son lo suficientemente intuitivas como para realizar las principales tareas desde la perspectiva del modelado de variabilidad (configuración, ámbito y visualización)?

Para poner en valor y materializar las ideas de CVL, se establece una colaboración con nuestro socio industrial, una compañía de placas de inducción que genera el firmware de sus placas de inducción siguiendo un enfoque Desarrollo Dirigido por Modelos (MDD). Este enfoque de desarrollo, sin definición explícita de variabilidad, fue aumentado en su herramienta de modelado con CVL para modelar la variabilidad existente entre sus productos.

Nuestra evaluación de usabilidad comprende tanto métodos que involucran usuarios finales *(Test Methods)* como son medición del desempeño *(Performance Measurement)*, cuestionario de satisfacción *(Satisfaction Questionnaire)* y entrevista *(Interview)* y métodos de inspección que no involucran usuarios finales *(Inspection Methods)* como *Keystroke-Level Model* [133]. La comunidad de investigación de interacción hompre-máquina aconseja combinar estos 2 tipos de métodos para lograr una correcta evaluación. Los métodos de evaluación de usabilidad seleccionados nos permiten (1) evaluar la eficacia, eficiencia y satisfacción y (2) identificar problemas de usabilidad.

Los resultados de eficacia y eficiencia (para tareas de configuración 85 % y 132.2 %/min, para tareas de ámbito 65 % y 49.93 %/min, y para tareas de visualización 88 % y 64.62 %/min) indican que la configuración de modelos en términos de sustituciones de fragmentos de modelo es lo suficientemente intuitiva pero, tanto el ámbito como la visualización permiten una mejora. Los resultados también nos permitieron identificar una lista de problemas de usabilidad que son relevantes para adoptar la variabilidad como forma de desarrollo software, para el proceso de estandarización de la variabilidad de OMG y para los proveedores de herramientas que utilizan modelado de la variabilidad.

6.3 *Common Variability Language*

Esta sección presenta los conceptos principales del *Common Variability Language* y cómo se aplica al modelado de la variabildiad. CVL es un Lenguaje Específico de Dominio *(Domain Specific Language, DSL)* para modelar la variabilidad de cualquier DSL basado en *Meta-Object Facility* (MOF) [134], una especificación de la OMG para definir un metamodelo universal para describir los lenguajes de modelado. *Common Variability Language* [3] define las variantes de un modelo base (que se ajusta a MOF) mediante la sustitución de partes variables del modelo base con reemplazos de modelos alternativos provenientes de un modelo de biblioteca. CVL tiene diferentes modelos: modelo base *(Base Model)*, modelo de biblioteca *(Library Model)*, modelo de especificación de variabilidad *(Variability Specification Model)* y modelo de resolución *(Resolution Model)*.

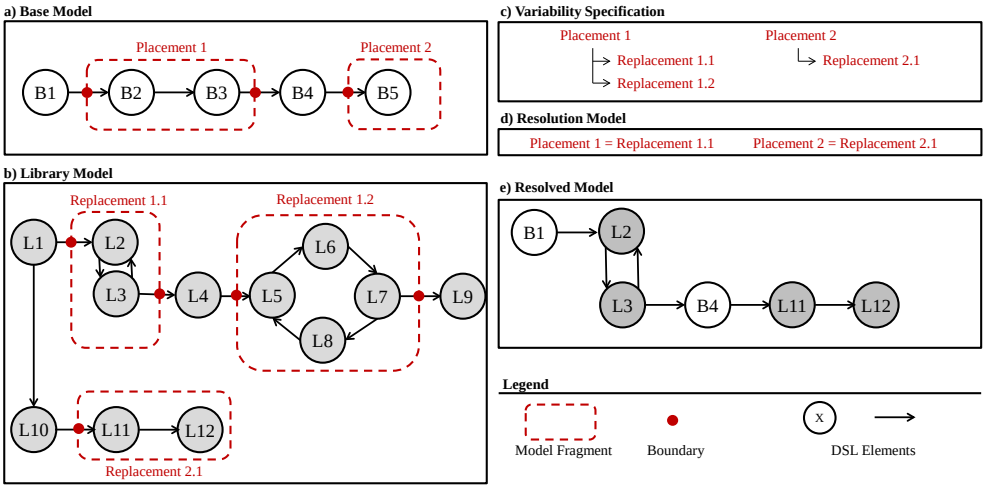


Figura 6.1: Variabilidad de fragmentos de modelo en CVL.

El modelo base es un modelo descrito por un DSL especificado que sirve como base para las diferentes variantes definidas sobre él. En CVL, los elementos del modelo base que están sujetos a variaciones son los denominados *placement fragments* (en lo sucesivo, *placements*). Un *placement* puede ser cualquier elemento o conjunto de elementos que está sujeto a variación. La Figura 6.1 a) muestra un ejemplo de un modelo base. En este modelo base, dos *placements* se definen sobre un modelo DSL simple: Placement 1 y Placement 2 (representados por líneas discontinuas ovales). Los elementos del DSL son círculos y flechas.

Para definir alternativas para un *placement*, CVL usa un modelo de biblioteca. El modelo de biblioteca se describe con el mismo DSL que el modelo base. Cada una de las alternativas para un *placement* es un *replacement fragment* (en lo sucesivo, *replacement*). De forma similar a los *placements*, un *replacement* puede ser cualquier elemento o conjunto de elementos que se pueden usar como variación para un *placement*. La Figura 6.1 b) muestra un ejemplo de un modelo de biblioteca. En este modelo de biblioteca, se definen tres *placements*: Replacement 1.1, Replacement 1.2 y Replacement 2.1 (representados por líneas ovales discontinuas).

Cada *placement* y *replacement* se define junto con sus fronteras. Las fronteras indican que hay dentro o fuera de cada fragmento (*placement* o *replacement*) en términos de referencias entre otros elementos del modelo. Por ejemplo, el Placement 1 en la Figura 6.1 tiene dos fronteras representadas por puntos.

El modelo de especificación de variabilidad en CVL contiene las reglas que rigen las posibles sustituciones en el DSL en función de los *placements* y *replacements* involucrados. Por ejemplo, en la Figura 6.1, el Placement 2 solo puede ser sustituido por el Replacement 2.1, pero el Placement 1 puede ser reemplazado por Replacement 1.1 o por el Replacement 1.2. El modelo de resolución especifica un conjunto de sustituciones de fragmentos de modelo que se deben realizar para crear una configuración particular del modelo base. Cada sustitución hace referencia a un *placement* (modelo base) y un *replacement* (modelo de biblioteca). Por ejemplo, el modelo de resolución en la Figura 6.1 especifica las siguientes sustituciones de fragmentos del modelo: Placement 1 = Replacement 1.1, y Placement 2 = Reeplacement 2.1.

La materialización de un modelo de resolución produce un modelo resuelto. Para cada sustitución de fragmentos del modelo de resolución, el proceso de materialización elimina elementos de un *placement* e inyecta elementos de un *replacement*. Cuando se materializa una sustitución, el modelo resuelto (con *placements* sustituidos por *replacements*) continúa siendo conforme al metamodelo del modelo base y el modelo de biblioteca. La parte inferior de la Figura 6.1 muestra el modelo resuelto que se genera por la materialización del modelo de resolución. Los elementos del Placement 1 fueron reemplazados por los elementos de Replacement 1.1, y los elementos de la Placement 2 fueron reemplazados por los elementos del Replacement 2.1.

En este capítulo, se estudian tareas de configuración que abordan (1) la manipulación de las sustituciones de fragmentos en un modelo de resolución y (2) la materialización del modelo de resolución para producir nuevos productos de

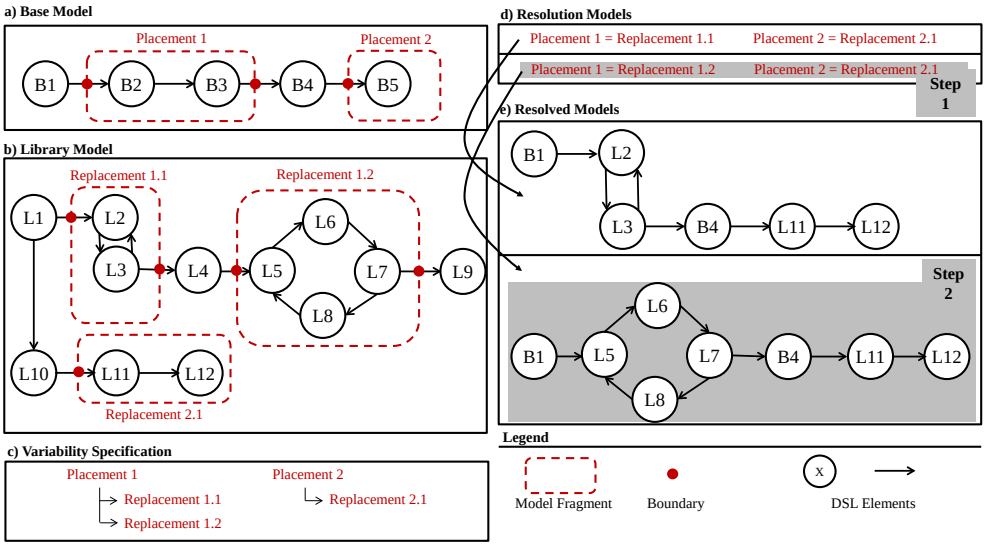


Figura 6.2: Tarea de configuración.

acuerdo con las sustituciones realizadas con fragmentos. La Figura 6.2 muestra una tarea de configuración en CVL que implica los siguientes pasos:

- Paso 1: En el modelo de resolución Figura 6.2 d), se crea un nuevo modelo de resolución (representado con fondo gris en d) de la Figura 6.2) con las sustituciones del Placement 1 por Replacement 1.2 y Placement 2 por Replacement 2.1.
- Paso 2: El nuevo modelo de resolución anterior se materializa generando un nuevo modelo resuelto (representado con fondo gris en la Figura 6.2 e)).

Las tareas de ámbito abordan el modelado explícito y la gestión de la variabilidad y las características comunes en activos reutilizables. En CVL, esto se traduciría en la creación, modificación o eliminación de *placements* (elementos del modelo sujetos a variaciones), *replacements* (posibles variaciones) y sustituciones (conjunto de un *placement* y un *replacement* adecuado). La Figura 6.3 muestra una tarea de ámbito para crear un nuevo *replacement* de la siguiente manera:

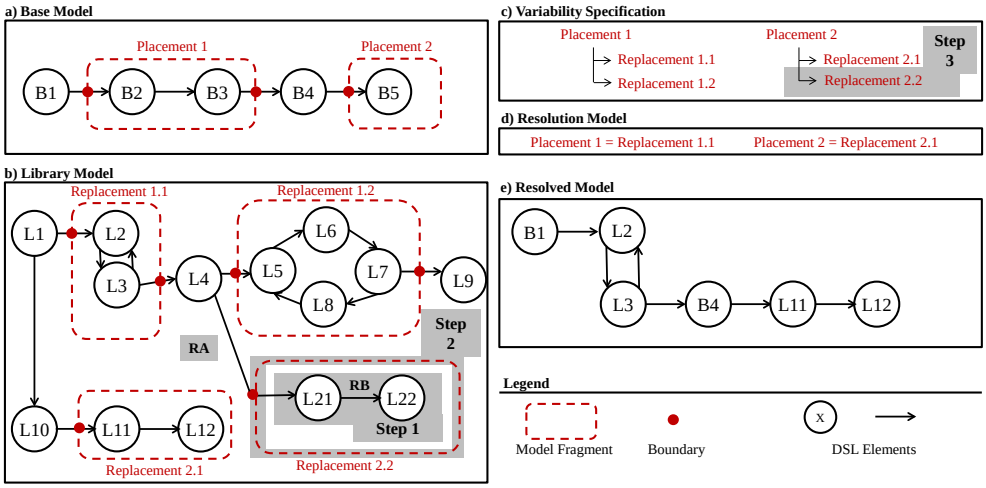


Figura 6.3: Tarea de ámbito.

- Paso 1: Se crea un nuevo *replacement* en la biblioteca de modelos (ver b) de la Figura 6.3). La biblioteca de modelos se amplía con elementos del DSL (L21, L22 y las relaciones RA y RB).
- Paso 2: los elementos se formalizan como un *replacement* del modelo, en particular, la relación RA juega el papel de frontera de *replacement* y L21, RB y L22 conforman el *replacement* en sí. El nuevo *replacement* se llama Replacement 2.2 (representado por un fondo gris en b) de la Figura 6.3).
- Paso 3: en el árbol de variabilidad del Placement 2 se ha ampliado con el Replacement 2.2 que es una hoja nueva en el árbol de variabilidad (representada por un fondo gris en c) de la Figura 6.3). Ahora, el Placement 2 puede sustituirse por el Replacement 2.1 o el Replacement 2.2 (el nuevo *replacement* creado en el Paso 2). Anteriormente, el Placement 2 solo podía ser sustituido por el Replacement 2.1.

Las tareas de visualización estudian como comunicar efectivamente al usuario final las relaciones existentes entre diferentes *placements*, *replacements*, sustituciones y resoluciones. Es decir, informan al usuario del modelo de variabilidad que subyace a los productos. La Figura 6.4 muestra una tarea de visualización que implica los siguientes pasos:

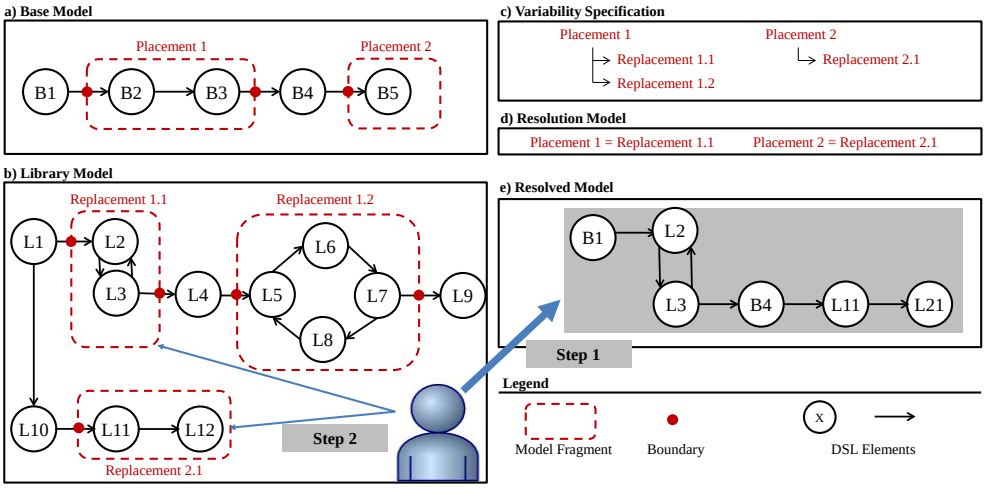


Figura 6.4: Tarea de visualización.

- Paso 1: Un modelo resuelto es inspeccionado por el modelador CVL (representado por un fondo gris en e) de la Figura 6.4).
- Paso 2: El modelador de CVL observa en la biblioteca de modelos los *replacements* que conforman el modelo resuelto (ver b) en la Figura 6.4).

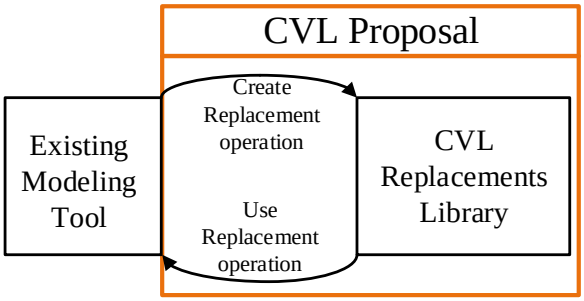


Figura 6.5: Herramienta de modelado aumentada con CVL.

La propuesta de CVL [135, 136] está diseñada para funcionar junto con un editor DSL existente previamente. La Figura 6.5 muestra una visión general de la aplicación de CVL a un editor de DSL ya existente. La parte izquierda muestra el editor DSL, mientras que la parte derecha representa la biblioteca de *replacements* que se usará para definir las variantes del modelo base.

Mediante la operación de reemplazo, los usuarios pueden realizar sustituciones, incluyendo fragmentos de la biblioteca de modelos que está siendo editada. Mediante la operación de creación de *replacements*, los usuarios pueden crear nuevos fragmentos de sustitución (*replacements*) e incorporarlos a la biblioteca [1].

Los anteriores son los principales elementos y operaciones de CVL, y deben cumplirse para aplicar CVL para un DSL determinado. Es necesario aumentar el editor DSL para habilitar las operaciones definidas por CVL, pero su aplicación es la misma para cualquier DSL dado. Para más detalles sobre el funcionamiento interno de CVL ver [135, 136].

6.4 Estudio empírico

Se ha desarrollado un estudio empírico, concretamente un estudio de caso. En nuestro caso, como queremos evaluar los mecanismos proporcionados por CVL al modelar variabilidad, vamos a utilizar la herramienta de modelado de nuestro socio industrial (una compañía de placas de inducción). Hemos aumentado la capacidad de desarrollo de la herramienta de modelado con CVL *(Modeling Tool + Common Variability Language, MT+CVL)*, lo que permite modelar la variabilidad de las placas de inducción creadas por nuestro socio industrial.

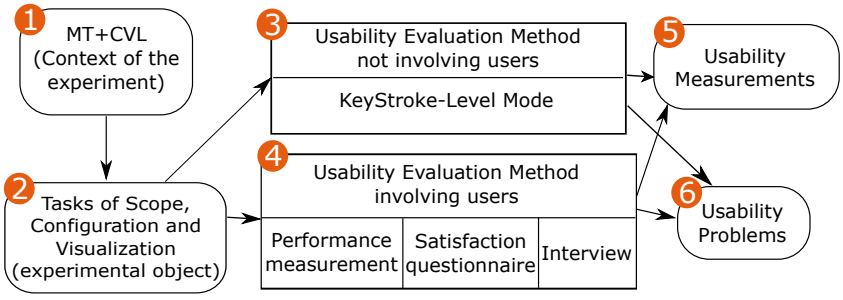


Figura 6.6: Perspectiva general de estudio empírico.

Para diseñar el estudio de caso, se pueden seleccionar diferentes Métodos de Evaluación de la Usabilidad *(Usability Evaluation Methods, UEM)*. En relación a los diferentes UEMs, la comunidad de investigación en el ámbito de la interacción de hombre-máquina sugiere combinar UEMs que involucran usuarios finales *(Test Methods)* con UEMs que no involucran usuarios finales *(Inspection Methods)*, para lograr unos mejores resultados en la evaluación [133].

[1] Ejemplo de operaciones con fragmentos de modelo en: http://folk.uio.no/oysteinh/demo1.htm

La Figura 6.6 muestra una visión general del estudio empírico que será presentado a lo largo del resto de este capítulo. Primero, (1) presentamos el contexto del estudio empírico: MT+CVL. Luego, (2) se obtiene un conjunto de tareas que representan las tres facetas del modelado de variabilidad. Luego, (3) el UEM de inspección seleccionado (sin usuarios finales) se aplica directamente a las tareas. Posteriormente, (4) llevamos a cabo los UEMs de tipo *test* con usuarios finales de nuestro socio industrial para medir la eficacia, la eficiencia y la satisfacción. Finalmente, presentamos los resultados obtenidos, (5) un conjunto de medidas de usabilidad (en términos de eficacia, eficiencia y satisfacción) y (6) un conjunto de problemas de usabilidad.

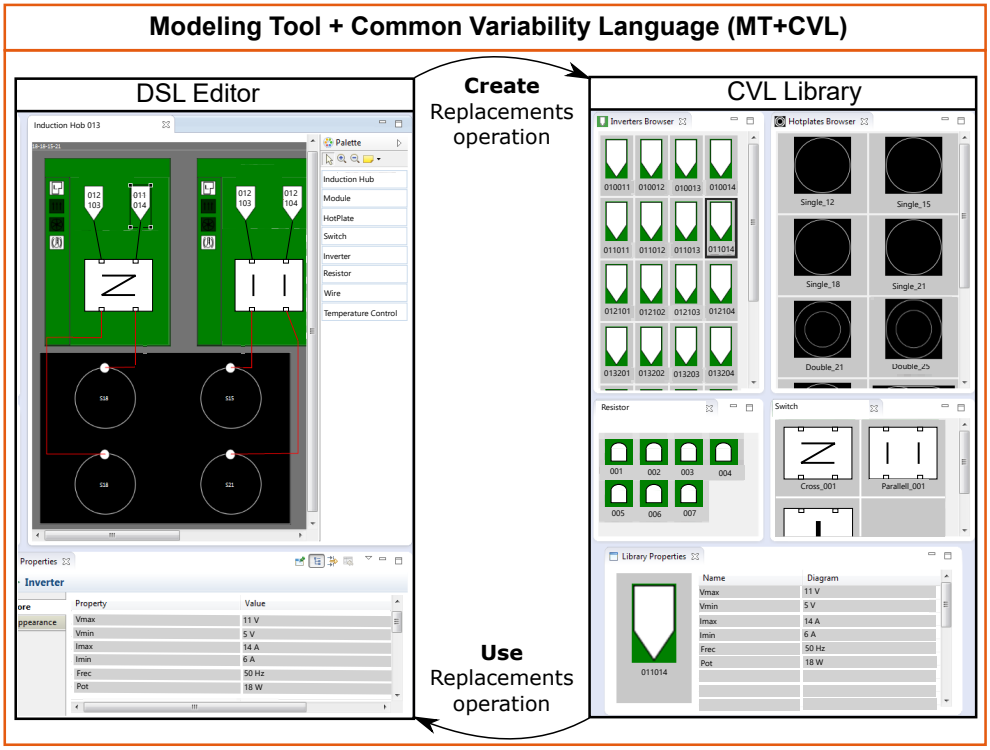


Figura 6.7: MT+CVL para el desarrollo de placas de inducción.

6.4.1 Contexto del estudio

Hemos aplicado CVL a la herramienta de modelado de nuestro socio industrial. Es decir, hemos aumentado la herramienta de modelado que incluye e integra las operaciones de CVL y la biblioteca (tal como se presenta en la sección 6.3, lo que da como resultado la herramienta MT+CVL que se utilizará durante el resto del estudio). Esta es la operación habitual para aumentar una herramienta de modelado con CVL, el mismo proceso se desarrollaría al aplicar esta ampliación en cualquier otra herramienta de modelado.

El DSL para placas de inducción utilizado por nuestro socio industrial está formado por 46 metaclases, 74 referencias entre ellas y más de 180 propiedades de metaclases. Las placas de inducción usan electromagnetismo para generar calor que se transfiere a los utensilios de cocina. Cada placa de inducción se compone de dos módulos de potencia y cada uno de ellos tiene dos inversores, que se encargan de proporcionar el suministro eléctrico requerido para generar el campo magnético. Los inversores están conectados a los inductores, el elemento donde se crea el campo magnético modificable. Los inversores y los inductores están conectados por un canal, que transfiere energía desde el inversor al inductor. La interfaz de usuario de una placa de inducción tiene controladores para configurar el nivel de potencia de cada inductor. Tiene puertos para conectar cada inductor con su controlador.

La parte izquierda de la Figura 6.7 muestra el editor gráfico de los modelos. Cada vez que se abre un modelo de placa de inducción, se muestra aquí. Este editor ha sido generado por medio de *Graphical Model Framework (GMF)* y permite al usuario crear y modificar modelos para un DSL (en este caso, placas de inducción). A partir de este editor, el usuario puede crear nuevos modelos desde cero o modificar modelos existentes (mediante la paleta y modificando las propiedades de los elementos del modelo).

Además, la operación de creación de *replacements* permite a los ingenieros crear nuevos *replacements* que se incluyen en la biblioteca. El ingeniero selecciona el elemento modelo (o elementos) que se usarán como *replacements* del editor y crea un *replacement*. Luego, ese *replacement* se incluye en la biblioteca, quedando disponible para incluirlo en otros modelos de placas de inducción (por medio de la operación de replacement).

La parte derecha de la Figura 6.7 muestra la biblioteca de *replacements*, donde se muestran todos los *replacements* que son parte de la MT+CVL. Un *replacement* puede ser cualquier elemento individual del modelo o un conjunto de elementos del modelo. En particular, la biblioteca se divide en *replacements*

de inversores (esquina superior izquierda), *replacements* de inductores (esquina superior derecha), *replacements* de sensores (esquina inferior izquierda) y *replacements* de redes *inverter* (esquina inferior derecha).

Los ingenieros utilizan la herramienta mostrada en la Figura 6.7 en diversas circunstancias:

- Modificación de un producto ya existente.
- Generación de un nuevo producto.
- Modificación de un *replacement*.
- Creación de un nuevo *replacement*.
- Visualización de los *replacements* que componen un producto.
- Visualización de los productos que utilizan un *replacement*.

Algunas de las causas que llevan a los ingenieros a realizar las anteriores operaciones pueden ser: lanzamientos de nuevos productos, modificaciones en las especificaciones técnicas de componentes o valoración de la influencia de un cambio de un *replacement* en el catálogo de productos. Cuando el ingeniero realiza cualquiera de las operaciones que le permite la herramienta, la selección de las distintas opciones viene marcada por:

- Especificación de la variabilidad: la herramienta tiene unas especificaciones que deben ser cumplidas por todos los productos.
- Conocimiento del ingeniero: El ingeniero puede escoger, de entre las opciones calificadas como validas por la herramienta acorde al modelo de variabilidad, la que considere oportuna.

La Figura 6.8 muestra los pasos para realizar una tarea de configuración usando MT+CVL. En esta tarea, se crea un nuevo modelo resuelto (una nueva placa de inducción) mediante la sustitución de un *replacement* (un inversor) en un modelo base (una placa de inducción existente). Esta tarea implica dos pasos:

- Paso 1: Utilizando la paleta del Editor DSL (parte izquierda de la Figura 6.8) se muestra la placa de inducción existente y se seleccionará el inversor para sustituir (inversor original). A continuación, se selecciona un nuevo inversor desde el navegador de inversores en la Biblioteca CVL (parte derecha de la Figura 6.8).

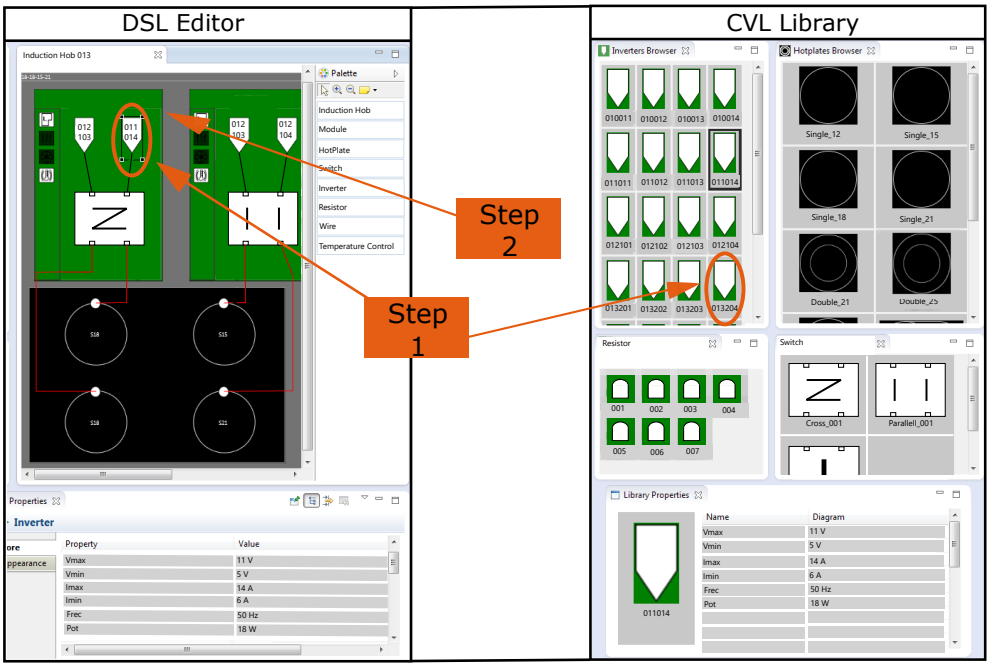


Figura 6.8: Tarea de configuración en MT+CVL.

- Paso 2: Mediante un menú contextual, utilizando el botón derecho del ratón, se realiza la sustitución del inversor original por el nuevo inversor. Finalmente, este cambio se materializa generando un nuevo modelo de placa de inducción.

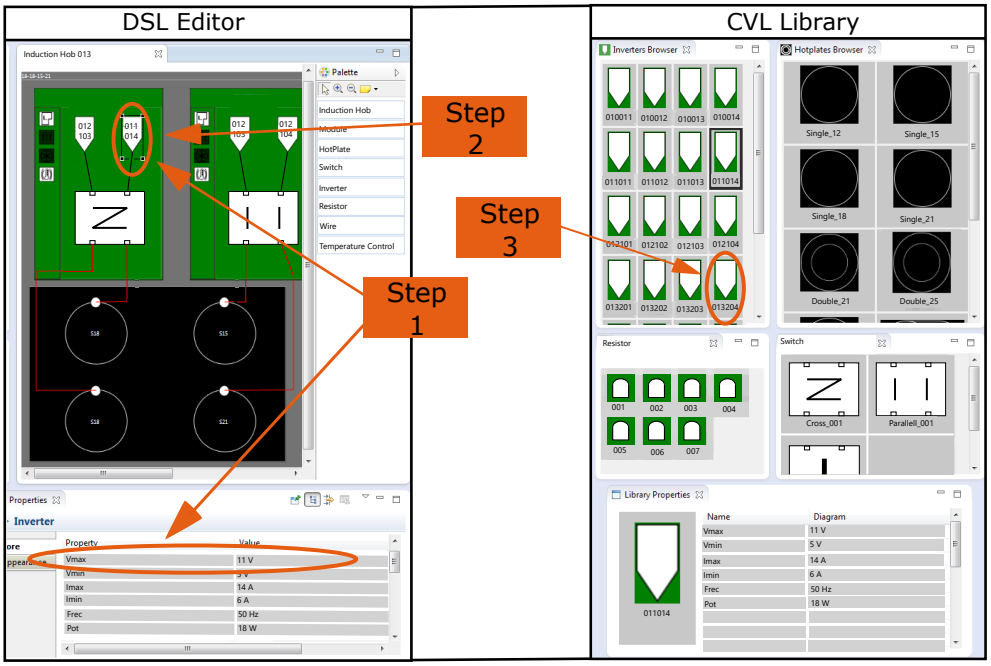


Figura 6.9: Tarea de ámbito en MT+CVL.

La Figura 6.9 muestra los pasos para realizar una tarea de ámbito utilizando MT+CVL. Al realizar esta tarea, el usuario crea un nuevo inversor y puede utilizar este nuevo inversor para generar nuevas placas de inducción a partir de un modelo base (una placa de inducción existente). Esta tarea implica tres pasos:

- Paso 1: Usando la paleta y la ventana de propiedades del Editor DSL (parte izquierda de la Figura 6.9) se crea un nuevo inversor, este inversor está conectado a un *switch*.
- Paso 2: mediante un menú contextual y de selección, los elementos se formalizan como un *replacement*. La relación *switch*-inversor es el límite y el inversor es el nuevo *replacement*.

- Paso 3: mediante un menú contextual, el nuevo *replacement* (un nuevo inversor) se agrega al navegador de inversores en la Biblioteca CVL (parte derecha de la Figura 6.9). El navegador de los inversores desempeña el papel de árbol de variabilidad en la especificación de variabilidad.

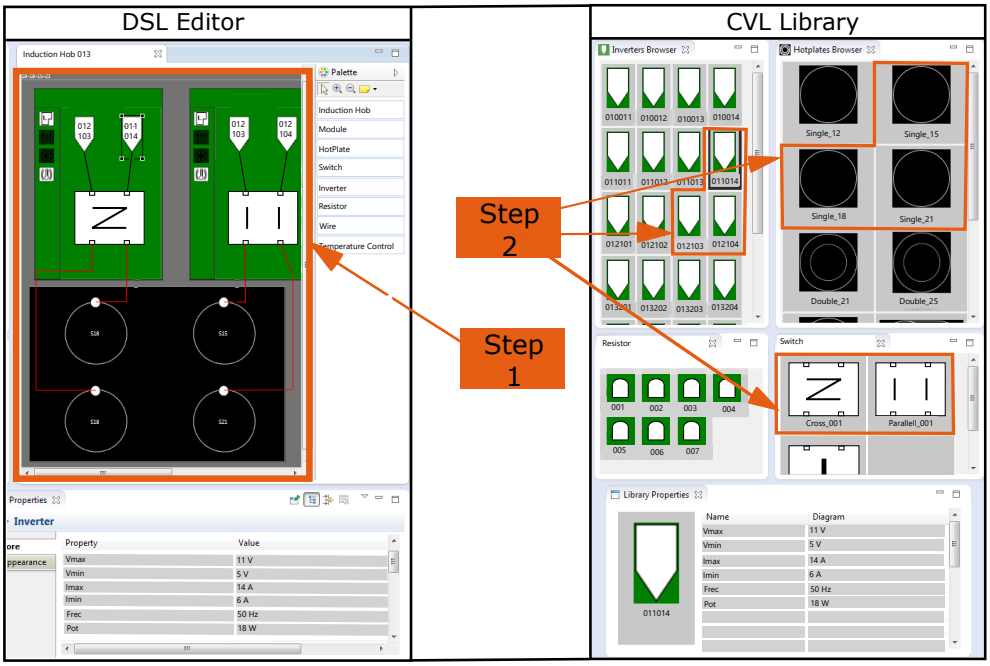


Figura 6.10: Tarea de visualización en MT+CVL.

La Figura 6.10 muestra los pasos para realizar una tarea de visualización usando MT+CVL. Al realizar esta tarea, el usuario visualiza los *replacements* de un modelo resuelto seleccionado (una placa de inducción). Esta tarea implica dos pasos:

- Paso 1: Usando la ventana del Editor DSL (parte izquierda de la Figura 6.10) se muestra la placa de inducción seleccionada.
- Paso 2: Mediante una opción de filtrado, todos los *replacements* de la placa de inducción seleccionada se resaltan en la Biblioteca CVL (parte derecha de la Figura 6.10).

6.4.2 Selección de tareas

Para identificar las tareas realizadas por los usuarios finales de la MT+CVL se utilizaron un conjunto de preguntas abiertas en una entrevista con los citados usuarios finales. Las preguntas abiertas se extrajeron de la información almacenada por los ingenieros de software en la fase de captura de requisitos. Las preguntas abiertas tienen el objeto de encontrar información sobre las personas que usarán la herramienta y cómo usarán la MT+CVL. Algunas preguntas fueron: "¿Puede explicar su trabajo sobre el software de placas de inducción?", "¿Cuáles son las tareas más difíciles?", "¿Cómo es el flujo de trabajo?" o "¿Cuáles son las tareas más comunes?".

Un instructor, un observador y cinco ingenieros electrónicos de nuestro socio industrial para la fabricación de placas de inducción participaron en la entrevista para identificar las tareas. La entrevista fue dirigida por el instructor, quien realizó las preguntas abiertas. Un observador tomó notas y grabó la entrevista para un posterior análisis. La entrevista se realizó en una sala de reuniones de nuestro socio industrial.

Teniendo en cuenta las respuestas, la grabación y las notas, generamos una primera lista de tareas. Algunas tareas en esta lista fueron: "Generar una nueva placa de inducción", "Validar una placa de inducción", "Modificar un componente en particular en un módulo de una placa de inducción", "Modificar un inversor en todos los módulos en los que es utilizado", "Detectar que componente es el más utilizado en un conjunto de placas" o "Eliminar un componente de todas las placas en las que aparece".

Posteriormente, los usuarios finales priorizaron las tareas en función de la frecuencia y la relevancia de las tareas en su trabajo. Por ejemplo, la tarea "Eliminar un componente" obtuvo menos prioridad que "Conocer los componentes de una placa de inducción". Finalmente, un ingeniero de SPL clasificó el conjunto de tareas, en función de su naturaleza, ordenadas por los usuarios finales. Los tres tipos de tareas explicadas anteriormente de una herramienta de modelado de variabilidad (configuración, ámbito y visualización) se utilizan en la clasificación. El ingeniero de SPL seleccionó tareas representativas para cada tipo.

Cada tarea seleccionada por el ingeniero de SPL se reescribió como una tarea ejecutable en MT+CVL. Por ejemplo, para la tarea "¿Cuántas placas de inducción incluyen cierto modelo de inversor?", una tarea ejecutable adecuada sería "¿Cuántas placas de inducción incluyen el inversor INV016034 entre sus componentes?".

Como resultado del proceso anterior se generaron seis tareas ejecutables, dos para cada tipo de tarea de la herramienta de modelado de variabilidad[2]:

- T1: La placa de inducción IH013 tiene un problema con el módulo MOD008 y este módulo debe ser reemplazado por el módulo MOD014. En las otras placas de inducción, el módulo MOD008 no debe reemplazarse.
- T2: El inversor INV016034 en el módulo MOD017 en la placa de inducción IH021 no funciona correctamente. El módulo debe ensamblar el inversor INV019034. Esta sustitución debe afectar a todas las placas de inducción con el módulo anterior.
- T3: La placa de inducción IH021 utiliza en el módulo MOD073 el inversor INV015034. El parámetro VMAX de este inversor es incorrecto. Se debe crear un nuevo inversor clonando el inversor que funciona de forma incorrecta. El nuevo inversor tiene su parámetro VMAX igual a 42. La sustitución debe afectar a todas las placas de inducción con el módulo anterior.
- T4: El módulo MOD021 en la placa de inducción IH003 debe reemplazar el inversor INV015042 por el nuevo inversor INV016042. Este reemplazo no debe afectar a otras placas de inducción.
- T5: Detectar todos los componentes en la placa de inducción IH021.
- T6: ¿Cuál es el módulo más utilizado de entre lo siguientes módulos MOD021, MOD014, MOD017, MOD101 ?.

Las tareas (T1) y (T2) son del tipo configuración, (T3) y (T4) son de tareas de ámbito y, finalmente, (T5) y (T6) son de tareas de visualización.

6.4.3 Diseño del estudio

La figura 6.11 muestra, destacando con un fondo gris, todas aquellas características que forman parte del estudio. Algunas características incluidas son *Estudio de caso* o *Pregunta de investigación*. Por contra las características no presentes en este estudio han sido "suavizadas" como *Estudiante* o *Experimento*.

[2]La nomenclatura de los componentes ha sido modificada para preservar la información confidencial. La información omitida no es relevante para la investigación.

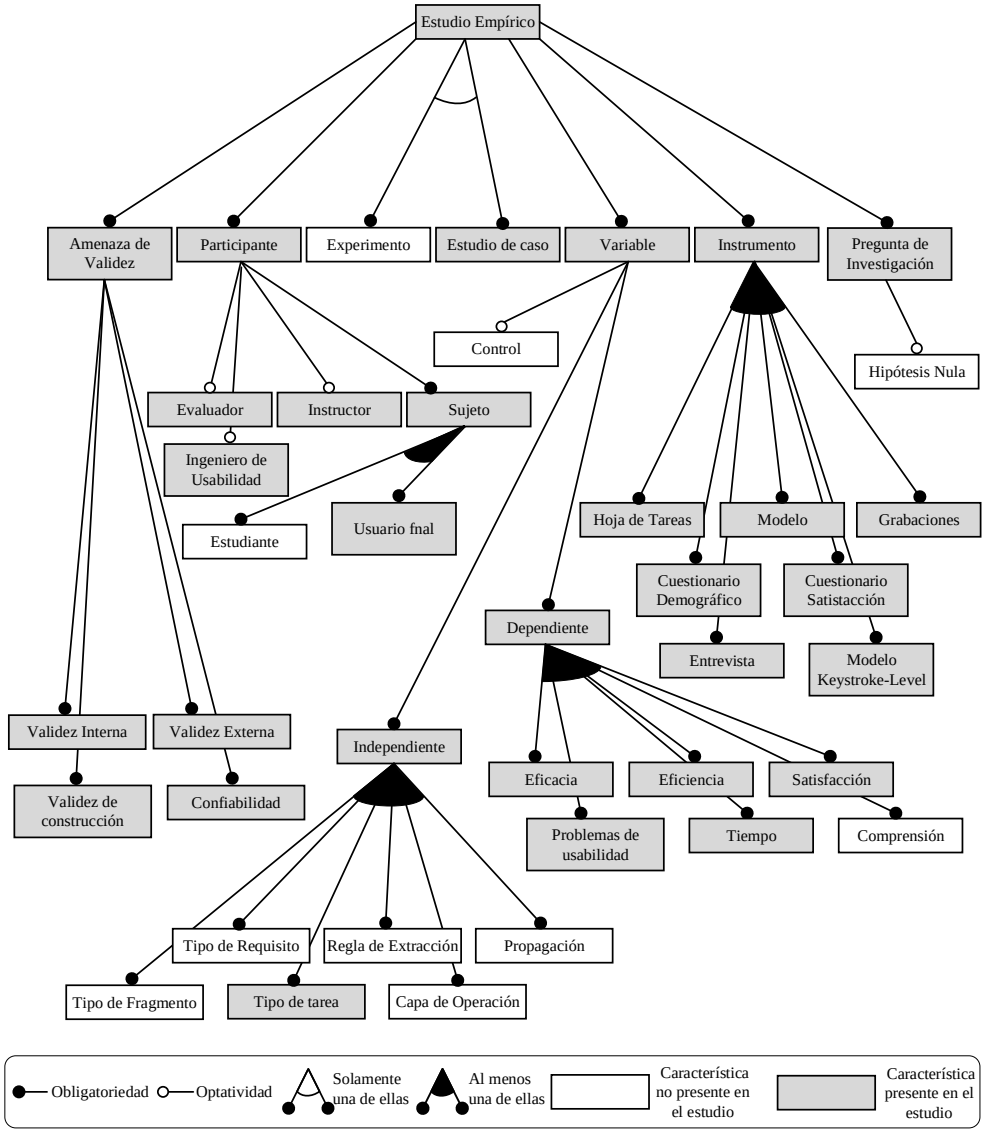


Figura 6.11: Modelo de características del estudio empírico para evaluar la usabilidad de MT+CVL.

Objetivo

El objetivo de la investigación descrita en este capítulo es analizar la usabilidad de una herramienta MT+CVL en función del tipo de tarea desarrollada (configuración, ámbito y visualización) por usuarios finales con la citada herramienta. Las medidas utilizadas en nuestra investigación para lograr el objetivo anterior son eficacia, eficiencia, satisfacción y tiempo. La eficacia, eficiencia y satisfacción son dimensiones que forman parte de la definición de usabilidad [137] y son ampliamente utilizadas en el ámbito de la investigación [138]. Por otro lado, el tiempo es medido para realizar la evaluación sin usuarios [139].

Participantes

La evaluación involucró a los usuarios finales de la herramienta, se trata de ingenieros de nuestro socio industrial. El estudio empírico para el análisis de la usabilidad se desarrolló con 5 usuarios finales. Las razones para esta elección fueron las siguientes: en primer lugar, el número de usuarios finales disponibles para realizar el estudio empírico y, en segundo lugar, la investigación en el campo de interacción hombre-máquina justifica usar cinco sujetos en las pruebas de usabilidad para obtener el 80 % de los problemas de usabilidad [140].

Además de los sujetos, también participaron un instructor, un evaluador y un ingeniero de usabilidad. El instructor proporcionó información sobre la realización de los ejercicios, aclaró dudas durante la realización de las tareas y dirigió las entrevistas. El evaluador tomó notas para análisis posteriores. Finalmente, el ingeniero de usabilidad desarrolló la evaluación sin usuarios (test de inspección).

Definición de variables

En esta subsección se enumeran y definen todas las variables relevantes en el estudio de caso.

Variables independientes: La variable independiente es el tipo de tarea a realizar, que es una variable nominal con tres valores: configuración, ámbito y visualización. Para más información ver 6.3 y 6.4.1.

Variables dependientes: En nuestro estudio empírico, los usuarios finales tuvieron que realizar seis tareas y rellenar un cuestionario de satisfacción. La realización de las tareas nos permitió medir la eficacia, la eficiencia y la satis-

facción. Por otro lado, la evaluación sin usuarios nos permitió medir el tiempo teórico para realizar las tareas. Las variables dependientes se definen como:

- La eficacia se define como el porcentaje de tarea realizada correctamente por el usuario final. Las tareas son descompuestas en una serie de pasos, y cada paso tiene un porcentaje ponderado con respecto a todo la tarea.
- La eficiencia es la relación entre la eficacia y el tiempo empleado (en minutos) para realizar el ejercicio.
- La satisfacción es la percepción que tiene un usuario final al utilizar la MT+CVL. Se mide usando un cuestionario de satisfacción al finalizar el conjunto de las seis tareas.
- Problemas de usabilidad, se trata de una enumeración de los problemas detectados cuando los usuarios finales desarrollan las tareas.
- El tiempo se calcula de forma teórica por un ingeniero de usabilidad utilizando una evaluación por inspección, en nuestro caso un Análisis de Acción (*Action Analysis*) [141].

Instrumentos

En esta subsección se muestra una descripción de todos los instrumentos utilizados para el desarrollo del estudio empírico.

Cuestionario demográfico: El cuestionario demográfico está compuesto por un conjunto de preguntas que permiten identificar el perfil de los sujetos participantes en el estudio. La información solicitada en el cuestionario a cada sujeto es: nivel de educación, tiempo de trabajo en su departamento actual (en años), edad, género, tiempo dedicado al desarrollo de software (diariamente), conocimiento sobre el entorno de desarrollo Eclipse y conocimiento sobre herramientas de modelado para generación de software.

Hoja de tareas: A cada usuario final participante en el estudio empírico se le proporcionó una hoja con 6 tareas a realizar. Dichas tareas han sido descritas en la sección 6.4.2.

Grabaciones de vídeo: El desarrollo del estudio fue grabado para todos los usuarios finales. Estas grabaciones nos permitieron calcular los valores de la eficacia y eficiencia y, por otro lado, detectar y analizar los posibles problemas de usabilidad.

Cuestionario de satisfacción: Este cuestionario de satisfacción fue *System Usability Scale, SUS*. SUS fue utilizado para determinar la satisfacción subjetiva del usuario con la herramienta MT+CVL. El cuestionario estaba compuesto por diez preguntas con una escala de Likert. En el SUS original se reemplazó la palabra "sistema" por "herramienta de modelado de la variabilidad". SUS, con solo diez preguntas, arroja resultados confiables [142]. Las preguntas del SUS abordan diferentes aspectos de la reacción del usuario a la herramienta como por ejemplo: "Encontré la herramienta de modelado de variabilidad innecesariamente compleja", "Me sentí muy seguro usando la herramienta de modelado de variabilidad"; en lugar de preguntarle al usuario con el objetivo de evaluar las características específicas del sistema (por ejemplo, apariencia visual, organización de la información, etc.).

Entrevista: El último UEM utilizado en la evaluación con usuarios fue la entrevista. Los objetivos de esta entrevista fueron dos: (1) determinar la comprensión por parte del usuario final de la herramienta de modelado y (2) obtener datos cualitativos de los comentarios de los usuarios. Las preguntas de la entrevista fueron preguntas abiertas y preguntas cerradas. Las preguntas cerradas se dirigieron a verificar la comprensión de las tareas en MT+CVL por parte de los usuarios finales. Por ejemplo, el instructor mostró dos imágenes al usuario final con el estado de la herramienta MT+CVL después de una tarea y el usuario final tuvo que elegir que imagen es la representativa del estado de la herramienta. El objetivo de las preguntas abiertas era detectar las partes del MT+CVL que eran más problemáticas, junto con las causas reales de los problemas [143]. Por ejemplo, una pregunta era "¿Cuáles han sido las tareas más difíciles para ti?".

Análisis de acción: Permite a un ingeniero de usabilidad, sin necesidad de utilizar usuario finales, predecir el tiempo que, de forma teórica, llevará el desempeño de una tarea en unas condiciones específicas.

6.4.4 Procedimiento

El procedimiento experimental está dividido en dos procesos diferenciados: (1) evaluación sin usuarios y (2) evaluación con usuarios. En los siguientes apartados se describen ambos.

Evaluación sin usuarios

El objetivo específico de esta fase de evaluación es encontrar problemas de usabilidad con un UEM sin usuarios finales. Fue elegido un método de esta naturaleza (sin usuarios finales) porque es complementario con un método de evaluación con usuarios finales [133, 139].

1. Task: T1		**Time**
1.1	**Sub-task: Select the induction hob**	
1.1.1	Initiate the task (decide to do)	1.2 s
1.1.2	Remember the induction hob reference	1.2 s
1.1.3	Find the induction hob	1.2 s
1.1.4	Point to induction hob	1.1 s
1.1.5	Double click on induction hob	0.4 s
1.1.6	Notice the selected induction hob in the editing window	1.2 s
1.2	**Sub-task: Select the module**	
1.2.1	Remember the module reference	1.2 s
1.2.2	Find the module	1.2 s
1.2.3	Point to module	1.1 s
1.2.4	Click on module	0.2 s
1.3	**Sub-task: Replace the module**	
1.3.1	Remember the new module reference	1.2 s
1.3.2	Find the new module	1.2 s
1.3.3	Point to new module	1.1 s
1.3.4	Click with the right button on the new module	0.2 s
1.3.5	Find the option replace	1.2 s
1.3.6	Point to option replace	1.1 s
1.3.7	Click on option replace	0.2 s
1.4	**Sub-task: Apply only to one induction hob**	
1.4.1	See the dialog box	1.2 s
1.4.2	Think the right choice	1.2 s
1.4.3	Find the right choice	1.2 s
1.4.4	Point to chosen button	1.1 s
1.4.5	Click on chosen button	0.2 s

Tabla 6.1: Predicción para la realización de la tarea T1 con *Keystroke-Level Model*.

El método de inspección elegido es Análisis de Acción (*Action Analysis*). Este método permite predecir el tiempo para completar tareas, utiliza el enfoque formal que a menudo se denomina modelo de nivel de pulsaciones (*Keystroke-Level Model*). El *Keystroke-Level Model* predice el tiempo de ejecución de la tarea desde un diseño específico y un escenario de tarea específico. El *Keystroke-Level Model* lo realiza un ingeniero de usabilidad en su lugar de trabajo. Este

método requiere un esfuerzo humano notable. Por contra, requiere un solo evaluador para realizarlo [133].

Para realizar el *Keystroke-Level Model*, un ingeniero de usabilidad descompuso cada tarea en un conjunto de subtareas. Más tarde, cada subtarea se descompone en una secuencia de acciones. Se asocia una duración a cada una de estas acciones y luego, se suman [141].

El *Keystroke-Level Model* tiene dos fases. La primera fase es determinar que pasos físicos y mentales realiza un usuario para completar una o más tareas con la herramienta a evaluar para predecir el tiempo que el usuario necesita para realizar la tarea. Para hacer esto, se asocia una duración a cada una de estas acciones o secuencia de operadores (físicos o mentales), y luego se suman. Esta duración se calcula utilizando el tiempo promedio que le cuesta a un usuario experto completar la acción, como lo sugieren los valores de tiempo de referencia mostrados en [141]. Por ejemplo, una pulsación de tecla en un teclado estándar se considera un movimiento físico que dura 0,28 segundos, mientras que, apuntar con el ratón a un objetivo en la pantalla cuesta 1,1 segundos.

La segunda fase consiste en analizar los pasos anteriores buscando problemas. Algunos problemas de usabilidad que el *Keystroke-Level Model* puede revelar son que se requieren demasiados pasos para realizar una tarea simple, o lleva demasiado tiempo realizar la tarea, o hay demasiado que aprender sobre la interfaz, etc. [139]. Además, se obtiene la cantidad de tiempo que el usuario necesita para realizar cada tarea. En nuestro estudio, un ingeniero de usabilidad aplicó *Keystroke-Level Model* a todas las tareas mostradas en la sección 6.4.2. Por ejemplo, la tarea T1 está compuesta por cuatro subtareas y el tiempo total previsto para realizar la tarea es de 21,1 segundos (consultar la Tabla 6.1).

Evaluación con usuarios

Los objetivos de esta fase son la evaluación de las medidas de usabilidad (eficacia, eficiencia y satisfacción) y la identificación de problemas de usabilidad. Para lograr estos objetivos, se utilizan los siguientes UEMs: cuestionario demográfico, medición del rendimiento (mediante realización de tareas que nos permiten medir eficacia y eficiencia), cuestionario de satisfacción y entrevistas. Estos UEMs se caracterizan por la participación de los usuarios finales. La evaluación con los usuarios se desarrolló de la siguiente forma para cada uno de los usuarios de forma individual:

1. Los usuarios finales recibieron información sobre las metas y objetivos de la evaluación. Les dijeron que no es una prueba de sus habilidades. También se les informó que su interacción sería grabada.

2. Los usuarios finales asistieron a un pequeño tutorial sobre MT+CVL.

3. Se solicitó a los usuarios finales que completaran un cuestionario demográfico antes de desarrollar las tareas.

4. Posteriormente, los usuarios finales recibieron una serie de instrucciones claras que eran específicas para la medición del rendimiento (para la realización de tareas). Se les aconsejó que intentaran realizar las tareas sin ayuda, y que solo deberían pedir ayuda si se sentían incapaces de completar la tarea por sí mismos.

5. Se les solicitó a los usuarios finales que completaran las seis tareas detalladas en la sección 6.4.2. Para evitar un posible efecto techo (*effect ceiling*), no hubo un límite de tiempo para completar las tareas.

6. Posteriormente, se les solicitó a los usuarios finales que completaran un cuestionario SUS de satisfacción.

7. Finalmente, se solicitó a los usuarios finales que respondieran una entrevista sobre la herramienta MT+CVL.

6.5 Resultados

En esta sección se muestran los resultados obtenidos tanto en la evaluación sin usuarios como en la evaluación realizada con la ayuda de usuarios finales.

6.5.1 Resultados de evaluación sin usuarios finales

Una vez realizado la evaluación con *Keystroke-Level Model* (se pueden observar las predicciones temporales en la Tabla 6.2) se puede concluir que no se han encontrado tareas que requieren demasiados pasos para ser realizadas, tareas que consuman un elevado tiempo para ser realizadas, ni es necesario un enorme esfuerzo cognitivo para realizarlas.

Task: **T1**							
ST 1.1	ST 1.2	ST 1.3	ST 1.4				**Total:**
6.3 s	3.7 s	6.2 s	4.9 s				21.1 s
Task: **T2**							
ST 2.1	ST 2.2	ST 2.3	ST 2.4	ST 2.5			**Total:**
6.3 s	7.3 s	3.7 s	6.2 s	4.9 s			28.4 s
Task: **T3**							
ST 3.1	ST 3.2	ST 3.3	ST 3.4	ST 3.5	ST 3.6		**Total:**
6.3 s	10.9 s	3.5 s	7.06 s	2.5 s	4.9 s		35.16 s
Task: **T4**							
ST 4.1	ST 4.2	ST 4.3	ST 4.4	ST 4.5	ST 4.6		**Total:**
6.3 s	7.3 s	7.3 s	7.3 s	2.5 s	4.9 s		35.6 s
Task: **T5**							
ST 5.1	ST 5.2	ST 5.3	ST 5.4	ST 5.5	ST 5.6		**Total:**
6.3 s	3.7 s	7.3 s	6 s	6 s	6 s		35.3 s
Task: **T6**							
ST 6.1	ST 6.2	ST 6.3	ST 6.4	ST 6.5	ST 6.6	ST 6.7	**Total:**
4.9 s	8.5 s	8.5 s	8.5 s	8.5 s	8.5 s	1.2 s	48.6 s

Tabla 6.2: Tiempos calculados para la realización de tareas con *Keystroke-Level Model*.

6.5.2 Resultados de evaluación con usuarios finales

En este apartado se muestran los resultados de la evaluación que contaron con la participación de usuarios finales.

Cuestionario demográfico

	Gender	Age	Education Level	Job	Time in Job (years)	Hours a day working with software induction hobs	Knowledge IDE Eclipse	Experience with modelling tools
user 1	M	33	A	E	12	7	No	No
user 2	M	45	A	E	5	8	Yes	Yes
user 3	M	31	A	E	2	7	Yes	Yes
user 4	M	30	A	E	5	8	Yes	No
user 5	M	35	A	E	10	3	No	No

Education Level A:engineer degree; *Job* E: Electronic engineer

Tabla 6.3: Características de los usuarios finales.

La Tabla 6.3 muestra los resultados que califican demográficamente a los sujetos participantes en el estudio. Todos ellos son ingenieros electrónicos, trabajan entre 3 y 8 horas desarrollando software, el 60 % no tienen conocimiento sobre la herramienta de desarrollo Eclipse y el 60 % nunca ha desarrollado software con una herramienta de modelado.

Eficacia y eficiencia

El objetivo de este paso de evaluación fue evaluar cómo de bien o mal se realizaron las tareas en la herramienta MT+CVL por los usuarios finales. Específicamente, medimos la eficacia y la eficiencia del usuario realizando unas tareas dadas, utilizando para ello un UEM denominado medición del desempeño (*Performance Measurement*).

	User	Task	Unassit. Task Effectiv. (%)	Assist. Task Effectiv. (%)	Time (min)	Completion rate/Task time	Asist.
Configuration	1	T1	100 %	0 %	0.73	136.36 %	0
		T2	100 %	0 %	1.63	61.22 %	0
	2	T1	33 %	66 %	0.16	206.25 %	1
		T2	20 %	25 %	0.16	125.00 %	1
	3	T1	100 %	0 %	0.62	162.16 %	0
		T2	100 %	0 %	1.25	80.00 %	0
	4	T1	100 %	0 %	0.60	166.67 %	0
		T2	100 %	0 %	2.23	44.78 %	0
	5	T1	100 %	0 %	0.50	200.00 %	0
		T2	100 %	0 %	0.72	139.53 %	0
		Mean	85 %	9 %	0.86	132.20 %	0.2
		Std desv	31 %	21 %	0.66	55.47 %	0.42
		Min	20 %	0 %	0.16	44.78 %	0
		Max	100 %	0 %	2.23	206.25 %	1
Scope	1	T3	0 %	61 %	4.80	0.00 %	1
		T4	16 %	0 %	0.57	28.24 %	0
	2	T3	100 %	0 %	1.20	83.33 %	0
		T4	16 %	0 %	1.13	14.12 %	0
	3	T3	60 %	0 %	0.87	69.23 %	0
		T4	100 %	0 %	3.12	32.09 %	0
	4	T3	100 %	0 %	3.20	31.25 %	0
		T4	100 %	0 %	1.35	74.07 %	0
	5	T3	60 %	0 %	2.18	27.48 %	0
		T4	100 %	0 %	0.72	139.53 %	0
		Mean	65 %	6 %	1.91	49.93 %	0.1
		Std desv	41 %	19 %	1.39	41.52 %	0.32
		Min	0 %	0 %	0.57	0.00 %	0
		Max	100 %	61 %	4.80	139.53 %	1
Visualization	1	T5	36 %	0 %	0.82	44.08 %	0
		T6	100 %	0 %	1.05	95.24 %	0
	2	T5	100 %	0 %	0.92	109.09 %	0
		T6	100 %	0 %	1.80	55.56 %	0
	3	T5	100 %	0 %	6.68	14.96 %	0
		T6	100 %	0 %	0.90	111.11 %	0
	4	T5	100 %	0 %	1.75	57.14 %	0
		T6	47 %	0 %	2.32	20.29 %	0
	5	T5	100 %	0 %	1.12	89.55 %	0
		T6	100 %	0 %	2.03	49.18 %	0
		Mean	88 %	0 %	1.94	64.62 %	0
		Std desv	25 %	0 %	1.75	34.85 %	0.00
		Min	36 %	0 %	0.82	14.96 %	0
		Max	100 %	0 %	6.68	111.11 %	0

Tabla 6.4: Resultados para la eficacia y la eficiencia.

En este UEM, los usuarios realizaron un conjunto predefinido de tareas de prueba (ver 6.4.2) mientras se recopilaban los datos de tiempo y errores cometidos. Los datos cuantitativos incluyen tiempos de rendimiento, tasas de error, tareas completadas o cantidad de asistencias requeridas para completar una tarea. Esta información permite el cálculo de la eficiencia y la eficacia. Los problemas de usabilidad vendrán de las notas que el evaluador ha sacado durante la prueba y del visionado de la grabación de la sesión. La eficacia considera el porcentaje de tareas realizadas correctamente sin asistencia, el porcentaje de tareas correctas que han requerido asistencia y la frecuencia de las asistencias requeridas por los usuarios. El valor de las asistencias indica la cantidad de veces que los usuarios pidieron ayuda para realizar las tareas. El valor de eficiencia es la relación entre el porcentaje de tareas correctas realizadas sin asistencia y el tiempo para finalizar estas tareas de acuerdo con el formato de industria común *(Common Industry Format, CIF)* para los informes de prueba de usabilidad *(Usability Test Reports)* [144].

En términos de frecuencia de asistencia, la Tabla 6.4 muestra que las tareas de configuración y ámbito son aquellas para las cuales los usuarios requirieron asistencia. El Usuario 2 requirió asistencia en dos ocasiones en tareas de configuración, y el Usuario 1 requirió asistencia en una tarea de ámbito. El Usuario 2 logró realizar la tarea de configuración después de la asistencia, en comparación, el Usuario 1 no logró progresar después de la asistencia con la tarea de ámbito. Por otro lado, las tareas de visualización no requieren asistencia. Esto indica que las tareas de visualización fueron las tareas menos difíciles.

Con respecto a las tareas terminadas correctamente, cuatro usuarios realizaron correctamente las tareas de configuración, tres usuarios completaron correctamente las tareas de visualización. Por otro lado, solo el Usuario 4 realizó correctamente las tareas de ámbito. Esto revela que las tareas de ámbito fueron las más difíciles.

La Tabla 6.4 muestra que los usuarios finales lograron altos valores de eficacia y eficiencia al ejecutar las tareas de configuración. Los datos muestran que 8 de 10 tareas de configuración (dos tareas por usuario) se realizaron correctamente. Por otro lado, los valores de eficacia y eficiencia para las tareas de ámbito son los más pequeños. Solo la mitad de las tareas de ámbito fueron completadas por los usuarios. Finalmente, para las tareas de visualización, los usuarios finales lograron un alto valor para la eficacia, pero el valor de la eficiencia es pequeño. De acuerdo con esto, un aspecto importante a considerar es el tiempo empleado por el Usuario 3 para realizar la tarea de visualización T5 y el tiempo empleado por el Usuario 4 para completar el 47 % de la tarea de visualización T6. Estos valores indican que las tareas más difíciles o problemáticas

son las tareas de ámbito. Por el contrario, los usuarios finales realizaron con alto desempeño las tareas de configuración. Por otro lado, los usuarios finales realizaron correctamente las tareas de visualización, pero les llevó demasiado tiempo si consideramos los valores calculados con *Keystroke-Level Model* (ver sección 6.5.1).

Satisfacción

	Normalized results	
1. I think that I would like to use this Variability Modelling tool frequently.	75 %	Yes
2. I found the Variability Modelling tool unnecessarily complex.	75 %	No
3. I thought the Variability Modelling tool was easy to use.	60 %	Yes
4. I think that I would need the support of a technical person to be able to use this Variability Modelling tool.	70 %	No
5. I found the various functions in this Variability Modelling tool were well integrated.	70 %	Yes
6. I thought there was too much inconsistency in this Variability Modelling tool.	85 %	No
7. I would imagine that most people would learn to use this Variability Modelling tool very quickly.	75 %	Yes
8. I found the Variability Modelling tool very cumbersome to use.	85 %	No
9. I felt very confident using the Variability Modelling tool.	60 %	Yes
10. I needed to learn a lot of things before I could get going with this Variability Modelling tool.	75 %	No
Total:	73 %	

Tabla 6.5: Resultados para la satisfacción.

Los usuarios finales tras finalizar la realización de las tareas rellenaron un cuestionario SUS. Los datos recopilados con este cuestionario deben introducirse en una hoja de cálculo para ser procesados. El cuestionario SUS está compuesto por diez preguntas con una escala *Likert* del 1 al 5. La contribución del puntaje de cada ítem varía de 0 a 4. Para los ítems 1, 3, 5, 7 y 9 (los ítems con redacción positiva) la contribución al marcado del cuestionario es el valor seleccionado por el usuario final menos 1. Para los ítems 2, 4, 6, 8 y 10 (los ítems de redacción negativa), la contribución es 5 menos el valor seleccionado por el usuario final. Finalmente se multiplica la suma de los puntuaciones por 2.5 para obtener el valor total del cuestionario SUS [145].

La Tabla 6.5 muestra los valores medios de los resultados con el cuestionario SUS. Los resultados muestran que los usuarios finales clasificaron la herramienta MT+CVL como "buena", según sugiere la escala mostrada en [146].

6.6 Problemas de Usabilidad

Los ingenieros (usuarios finales) de nuestro socio industrial han interiorizado los conceptos principales de CVL como la sustitución de *placements* por *replacements*. Las tareas asociadas con estos conceptos T1 y T2 (ver 6.3) fueron realizadas correctamente por los ingenieros de nuestro socio industrial. Indicaron que la herramienta permite reutilizar sistemáticamente fragmentos existentes al tiempo que se evita la redundancia innecesaria.

Sin embargo, hemos detectado algunos problemas de usabilidad *(Usability Problems, UP)*. Algunos de los UPs detectados son genéricos, podrían encontrarse en cualquier tipo de software y no están directamente relacionadas con los conceptos y operaciones de CVL. Por ejemplo, el tamaño de algunas etiquetas y botones, la discordancia entre los iconos y la acción representada. Este tipo de problemas quedan fuera del alcance de este capítulo.

Para cada una de los UPs relacionados con CVL extraídos del análisis de los datos, hemos generado una plantilla similar a la propuesta por [138], que indica la ID del UP, el nombre del UP (Nombre), la descripción de UP (Descripción), la clasificación del UP (Criterio ergonómico), el método de evaluación de usabilidad utilizado para detectar el UP (Fuente de UEM) y el número de ocurrencias para ese UP particular (Número de ocurrencias). Para determinar el criterio ergonómico de cada UP, hemos atendido a la clasificación de criterios ergonómicos de Bastien y Scapin [147]. Además, para cada UP, proponemos una solución para mitigar el problema y un consejo para los futuros diseñadores de herramientas. Estas soluciones y consejos se expresan en términos de los elementos CVL, son independientes del dominio lo que facilita su aplicación en cualquier herramienta que use CVL, independientemente del dominio de la SPL:

UP1: Nombre: La sintaxis concreta de los fragmentos lleva a los usuarios finales a perder puntos de variación en los modelos.

Descripción: Si diferentes fragmentos tienen la misma sintaxis concreta en el editor de modelos resueltos y en la biblioteca, los usuarios finales no se dan cuenta de las diferencias entre los *replacements* de la biblioteca. Por otro lado, si un fragmento tiene una sintaxis concreta diferente en

el editor de modelos resueltos y en la biblioteca, entonces los usuarios finales no consideran el *replacement* del modelo resuelto como un punto de variación.

Criterio ergonómico: Orientación - Agrupación / distinción de elementos.

Fuente de UEM: Entrevista

Número de ocurrencias: 3

Solución propuesta: Cada *replacement* en los modelos resueltos debe marcarse para ser reconocido como un punto de variación. Es decir, todos los puntos de variación compartirán una marca para indicar que se puede modificar. Además, diferentes *replacements* para un punto de variación compartirán un elemento de representación base común (para indicar que puede intercambiarse) pero también tendrán un elemento de representación individual (para distinguir entre diferentes *replacements*).

Consejo: El lenguaje de variabilidad debería mejorar su sintaxis concreta para resaltar el papel de los fragmentos como puntos u opciones de variación.

UP2: **Nombre:** La navegación desde la biblioteca de *replacements* al modelo resuelto no es directa.

Descripción: El usuario final generalmente busca los *replacements* de la biblioteca de *replacement* y luego quiere abrir un modelo determinado resuelto usando ese *replacement*. Sin embargo, esto no es sencillo y la navegación debe realizarse con el apoyo de un filtro. Por lo tanto, los *replacements* deberían poder presentar una lista de modelos resueltos que los usan para permitir que el usuario final abra los modelos de manera rápida.

Criterio ergonómico: Adaptabilidad - Flexibilidad.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 4

Solución propuesta: La inclusión de una forma directa de navegación desde un *replacement* particular a un modelo resuelto utilizando ese *replacement*. En particular, agregamos la funcionalidad de doble clic para mostrar la lista de modelos resueltos que usan ese *replacement* en concre-

to. Luego, el usuario puede elegir de ese subconjunto de modelos resueltos utilizando directamente el *replacement* seleccionado.

Consejo: La biblioteca de *replacements* debe contener soporte para identificar modelos resueltos dado un *replacement* particular.

UP3: **Nombre:** Falta de soporte para seleccionar el alcance de un cambio de propiedad realizado sobre un *replacement*.

Descripción: En el contexto de una modificación de las propiedades de *replacement* (que se usa en varios modelos de producto), el usuario final desea controlar el alcance de la propagación de los cambios realizados. Es decir, cuando un *replacement* que se usa en varios modelos resueltos cambia, el usuario debe tener el control sobre qué modelos resueltos (utilizando el *replacement* modificado) deberían verse afectados por el cambio.

Criterio ergonómico: Adaptabilidad - Flexibilidad.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 4

Solución propuesta: Cuando el usuario cambia el valor de una propiedad de un *replacement* que es utilizado por más de un modelo resuelto, mostramos una lista de modelos resueltos. Esta lista incluye los modelos resueltos que usan el *replacement* que se está modificando y el usuario puede seleccionar los modelos resueltos que se verán afectados por el cambio. Si se seleccionan todos, el *replacement* se modifica directamente. Si solo se seleccionan algunos, el *replacement* se duplica y el *replacement* nuevo se modifica y se agrega a los modelos resueltos seleccionados por el usuario (para que el resto de los modelos resueltos permanezcan sin cambios).

Consejo: Las herramientas deben incluir apoyo para la conciencia de las ramificaciones de los cambios de *replacement*.

UP4: **Nombre:** Falta de *feedback* cuando se crean nuevos *replacements* implícitamente.

Descripción: En el contexto de una creación implícita de un *replacement* (por ejemplo, en la descripción previa del problema, cuando el cambio se extiende solo a algunos modelos resueltos, es necesario crear un *replacement*), el usuario desea que se le notifique esa creación que se ha produ-

cido de forma implícita. De lo contrario, el usuario no está seguro de si la operación se realizó correctamente y si todos los modelos resueltos se actualizaron correctamente o no.

Criterio ergonómico: Orientación - *Feedback* inmediato.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 6

Solución propuesta: Dada la complejidad de algunas operaciones (como la modificación de propiedades que implican la creación de nuevos *replacements*) proponemos la inclusión de un informe confirmando los cambios después de la ejecución de una tarea compleja. Es decir, después de realizar una tarea compleja, mostramos un informe que indica qué elementos estaban involucrados y qué modificaciones se realizaron sobre ellos.

Consejo: Las herramientas deben proporcionar *feedback* al usuario cuando se crea un nuevo *replacement* implícitamente.

UP5: **Nombre:** Falta de soporte para crear un nuevo *replacement* explícitamente.

Descripción: En el contexto de una derivación de producto, el usuario final a veces necesita crear un *replacement* nuevo, e intenta hacerlo por medio de una entrada de menú contextual o incluso mediante atajos de teclado. Falta un método explícito para crear *replacements*, el único procedimiento para crear *replacements* nuevos es por medios implícitos. Consiste en cambiar un *replacement* y no extender los cambios a todos los modelos resueltos que lo usan. Además, el método explícito para crear *replacements* debe abordar los requisitos de reutilización y redundancia (no permitir duplicados, y permitir que el usuario final cree el nuevo *replacement* a partir de uno existente y luego modificarlo).

Criterio ergonómico: Adaptabilidad - Flexibilidad.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 4

Solución propuesta: Hemos incluido dos medios para crear nuevos *replacements*. Primero, cada biblioteca de *replacements* incluye un botón "nuevo" para crear un *replacement* nuevo desde cero. En segundo lugar,

se crea un menú contextual para cada fragmento de modelo que permite la creación de un nuevo *replacement* utilizando uno seleccionado como punto de partida o *replacement* base. Cuando se selecciona, el *replacement* se duplica y la nueva copia se abre para ser modificada por el usuario. Siempre que se guarde un nuevo *replacement* después de realizar una edición, es necesario verificar si hay duplicados, asegurándose de que no haya dos *replacements* idénticos en la biblioteca. Si se detecta un duplicado, se informa al usuario y se le presenta información para abordar el problema (cada *replacement* se presenta con la lista de modelos resueltos que lo utilizan y el usuario puede modificar cualquiera de los duplicados o fusionarlos).

Consejo: Las herramientas deben contener soporte para la detección de clones de fragmentos modelo.

UP6: **Nombre:** Falta de soporte para comparar *replacements*.

Descripción: En el contexto de una configuración de producto, el usuario final necesita determinar qué *replacement* de entre varios *replacements* satisface mejor sus necesidades actuales. Es decir, el usuario final desea realizar comparaciones entre *replacements* para determinar cuál debe usarse en ese producto en particular.

Criterio ergonómico: Adaptabilidad - Flexibilidad.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 1

Solución propuesta: Proponemos incluir medios para comparar dos *replacements*. En particular, hemos implementado una vista basada en tablas para presentar varios *replacements* con todas sus propiedades y resaltar las diferencias. Además, también mostramos los modelos resueltos que usan cada uno de los *replacements* que se comparan. Al usar esta utilidad de comparación, el usuario puede encontrar *replacements* que cumplan con requisitos particulares y determinar si alguno de los *replacements* existentes se ajusta a los citados requisitos o encontrar el mejor candidato para ser utilizado como la base de un nuevo *replacement* (utilizando la creación de nuevos *replacements* propuestos en UP5).

Consejo: Las herramientas deben incluir soporte para comparar fragmentos de modelo.

UP7: Nombre: Falta de operación de reemplazo en la biblioteca.

Descripción: En el contexto de una configuración de producto, la operación de reemplazo permite al usuario final realizar sustituciones de un *replacement* que se utiliza en un modelo resuelto (abierto en el editor) por *replacements* de la biblioteca. Luego, los cambios se pueden extender a otros modelos usando ese *replacement* particular que está siendo sustituido. Cuando el usuario desea realizar una operación de reemplazo (cambiar un *replacement* por otro) y propagar el cambio a todos los modelos resueltos utilizando esa sustitución, no tiene sentido abrir un único modelo resuelto (de todos los modelos resueltos que utilizan esa sustitución) y realizar el reemplazo en él. Por lo tanto, debemos permitir que el usuario realice una operación de reemplazo a nivel de biblioteca y no solo a nivel de modelo de producto.

Criterio ergonómico: Adaptabilidad - Flexibilidad.

Fuente de UEM: Tareas y entrevista.

Número de ocurrencias: 5

Solución propuesta: Proponemos incluir una operación de reemplazo que se puede aplicar directamente desde la biblioteca. Es decir, todas las instancias de un *replacement* particular (en modelos resueltos) se sustituyen por un *replacement* diferente. Por lo tanto, uno de los *replacements* ya no se usa en ningún modelo resuelto y se sustituye por otro seleccionado por el usuario. Para realizar esta acción, el usuario puede abrir el menú contextual de un reemplazo y seleccionar la opción "sustituir por". Luego, el usuario selecciona el *replacement* que se usará como sustituto.

Consejo: Las herramientas deben permitir la operación de reemplazo a nivel de biblioteca.

6.7 Amenazas a la validez

Esta sección describe las amenazas que no pudimos evitar pero que mitigamos y las amenazas que no pudimos abordar. Como ya ha sido comentado en anteriores capítulos, usamos la clasificación de amenazas a la validez de [116].

Validez de construcción: Esta amenaza fue tratada usando protocolos bien establecidos [138]. Los instrumentos utilizados en nuestra investigación son ampliamente aceptados en la comunidad de investigación en el ámbito de la

interacción persona-computador. Además, nuestro estudio no tiene el objetivo de evaluar una herramienta específica, buscamos evaluar los tres tipos de tareas que caracterizan (configuración, ámbito y visualización) que caracterizan a una herramienta de modelado aumentada con variabilidad.

Validez interna: Aunque el número de sujetos puede parecer relativamente pequeño, como ya ha sido comentado anteriormente la investigación en el ámbito interacción humano-computador aconseja usar cinco sujetos en una prueba de usabilidad para detectar el 80 % de los problemas de usabilidad [140].

Validez externa: El número de usuarios es muy bajo pero la naturaleza de los usuarios finales (ingenieros que desarrollan software en un entorno industrial) permite considerar los resultados como relevantes para otros modeladores y otros desarrolladores de herramientas de modelado aumentadas con CVL. Dado que la herramienta utilizada en este estudio es una herramienta concreta de nuestro socio industrial, la generalización de los hallazgos debe realizarse con precaución.

Confiabilidad: Para minimizar esta amenaza, los valores de las medidas estudiadas y las respuestas de los sujetos se han analizado utilizando las grabaciones y las anotaciones realizadas durante al realización de tareas. Para la medición de la satisfacción, usamos el cuestionario SUS, que es un cuestionario confiable según se afirma en [142].

6.8 Conclusión

Creemos que los resultados de la evaluación de usabilidad que hemos realizado son relevantes para los desarrolladores de software basados en modelos, para el proceso de estandarización de variabilidad de OMG y para los proveedores de herramientas de variabilidad de la siguiente manera:

- Desde el punto de vista de los desarrolladores de software basados en modelos, como es el caso de nuestro socio industrial, los resultados de la evaluación de usabilidad sugieren que CVL puede complementar sus herramientas de modelado actuales para formalizar y configurar la variabilidad (según los resultados obtenidos para la eficacia y la eficiencia). La biblioteca de fragmentos modelo de CVL puede permitirles cambiar de un enfoque *Clone & Own* a una reutilización sistemática de fragmentos modelo.

- Desde el punto de vista del proceso de estandarización de variabilidad actual del OMG, este documento proporciona evidencia de que la propuesta de CVL actual debería ampliarse para proporcionar una sintaxis concreta para los conceptos de fragmento de modelo. Es decir, la propuesta actual de CVL introduce los conceptos de *placement* de modelo y *replacement* de modelo, pero la propuesta carece de una sintaxis concreta para denotar los límites del modelo de fragmento. Esta falta de sintaxis concreta hace que los modeladores pierdan puntos de variación en los modelos (ver UP1 en sección 6.6).
- Finalmente, desde el punto de vista de los fabricantes/vendedores de herramientas de modelado, los resultados de la evaluación de usabilidad revelan que los modeladores requieren nuevas capacidades de edición para trabajar con fragmentos de modelo independientes tales como creación explícita (ver UP4, UP5 en sección 6.6), comparación de fragmentos (ver UP6 en sección 6.6), filtros basados en fragmentos (ver UP2 en sección 6.6) y propagaciones de cambios (ver UP3 y UP7 en sección 6.6).

7

Comprensión de fragmentos de modelo

Índice

7.1 Resumen del capítulo

La capacidad de gestión de la variabilidad del software se ha convertido en crucial para superar la complejidad y variedad de sistemas software. Para conseguir optimizar esta gestión, una representación comprensible de la variabilidad cobra enorme importancia. Sin embargo, en trabajos desarrollados anteriormente, se han detectado dificultades para comprender la variabilidad en un entorno industrial. Específicamente, los expertos en el dominio tenían dificultad para comprender la variabilidad en los fragmentos de modelo software para desarrollar el software de sus productos. Por esta razón, el objetivo de este capítulo es investigar en profundidad estas dificultades mediante la realización de un experimento. En dicho experimento los sujetos deben comprender la variabilidad de un sistema con el fin de lograr las configuraciones deseadas del producto. Nuestros resultados revelaron nuevas ideas sobre la configuración de productos que sugieren algunas líneas futuras para mejorar el enfoque del modelado de la variabilidad y, como consecuencia, promover la adopción de estas ideas en la industria.

7.2 Introducción

Con la constante evolución en el desarrollo software, los artefactos de software se han vuelto esenciales y más complejos. Un desafío fundamental en casi cualquier negocio es cómo administrar la variabilidad de su software para optimizar el proceso de desarrollo (es decir, mejorar la reutilización del código). La variabilidad es un concepto ampliamente estudiado en el ámbito de las SPL para apoyar el desarrollo y el mantenimiento de familias de productos de software [47, 1]. Los enfoques de modelado de variabilidad general (que son independientes del lenguaje en el que se especifican los productos de software) incluyen *Feature-Oriented Domain Analysis* (FODA) [45], *Orthogonal Variability Model* [1] y *Common Variability Language* (CVL) [56]. FODA se usa ampliamente para la gestión de la variabilidad mediante la descripción de características *(features)*. OVM es un lenguaje y una metodología para superponer la variabilidad sobre cualquier artefacto de desarrollo de software sin interferir en sus contenidos. CVL se recomienda para ser adoptado como estándar por el *Architectural Board* del *Object Management Group* (OMG) para gestionar la variabilidad. Estos enfoques pueden gestionar la variabilidad de los modelos de productos software mediante la descripción de características en términos de fragmentos de modelo.

Como se ha comentado en el capítulo 6, en el trabajo para evaluar la usabilidad detectamos que los expertos de dominio de nuestro socio industrial (división de fabricación de placas de inducción del grupo BSH) tenían dificultades para comprender la variabilidad en fragmentos de modelo para producir el software para la enorme cantidad de modelos de placas de inducción que producen (bajo las marcas Bosch y Siemens entre otros). Dado que la capacidad de gestionar la variabilidad en los modelos que utilizan una representación comprensible es crucial [80], buscamos investigar las dificultades para comprender la variabilidad de los fragmentos del modelo. Por esta razón, el objetivo de este capítulo es examinar las dificultades para comprender la variabilidad en los fragmentos del modelo software para la configuración del producto.

Para alcanzar el anterior objetivo realizamos un experimento en el que se expresaron puntos de variación *(variation points)* en modelos de un Lenguaje Específico de Dominio (DSL). Usando estos modelos, los sujetos tuvieron que realizar sustituciones de fragmentos de modelo para alcanzar una configuración de producto objetivo.

Los resultados obtenidos muestran cuatro hallazgos principales que son relevantes para el estudio del modelado de variabilidad general (FODA, OVM y CVL). Primero, la investigación previa descrita en el capítulo 6 consideró que usar la sintaxis concreta de los fragmentos del modelo como criterio de configuración era una fuente de soluciones incorrectas, pero este experimento revela que es posible utilizar este criterio para reducir significativamente el tiempo requerido para configurar productos. En segundo lugar, los participantes combinan intuitivamente los fragmentos del modelo antes de realizar la configuración del producto. Con esta operación se mejora la eficiencia en el proceso de configurar productos. Sin embargo, los enfoques generales sobre el modelado de variabilidad no son compatibles con estas combinaciones de fragmentos. En tercer lugar, los sujetos obtuvieron los peores resultados en las sustituciones de fragmentos que implican fragmentos recursivos en el modelo. Sin embargo, hasta la fecha la investigación sobre la configuración del producto ha descuidado el aspecto relacionado con los fragmentos de modelo recursivos. Otro aspecto a destacar es que los sujetos realizan pasos de configuración redundantes porque creen que deben realizar sustituciones de fragmentos de modelo para cada punto de variación, aunque ya contenga los elementos de la configuración que satisfacen el modelo objetivo. Normalmente, los materiales de aprendizaje disponibles para gestionar la variabilidad carecen de instrucciones para evitar estos pasos de configuración redundantes. Estos hallazgos proporcionan información sobre los próximos pasos para mejorar la forma de estudiar y gestionar el modelado de variabilidad.

7.3 Background sobre Configuración de Productos

Esta sección ilustra la relación entre la especificación de variabilidad y los fragmentos del modelo. Además, esta sección muestra ejemplos de sustituciones de fragmentos de modelo para alcanzar la configuración del modelo de producto deseado.

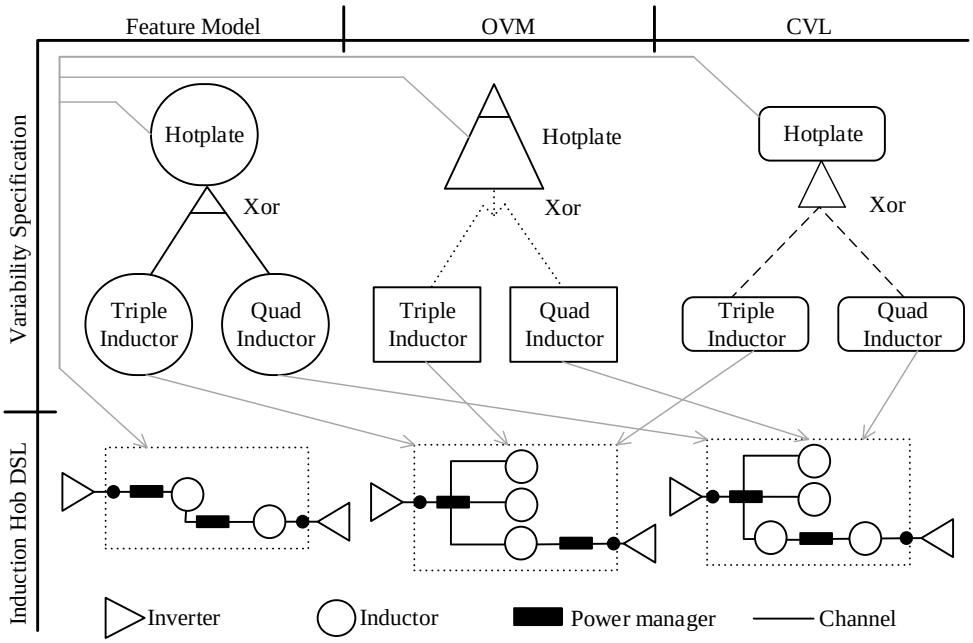


Figura 7.1: Relación entre especificación de variabilidad y fragmentos de modelo.

La mitad superior de la Figura 7.1 muestra una jerarquía de características que representan la especificación de variabilidad de un producto correspondiente a una placa de inducción usando los tres enfoques de modelado de variabilidad diferentes: Modelado de características (*Feature Model*), OVM y CVL. Estos enfoques siguen la idea de variantes superpuestas [58] en las que cada característica se relaciona con un fragmento de modelo que se expresa en los mismos términos de las instancias del producto (véanse las flechas de la Figura 7.1 que relacionan cada característica con un fragmento de modelo, que se expresa usando el DSL para placas de inducción). La superposición de una característica en un fragmento de modelo podría representar un punto de variación (ver la característica del *Hotplate* de la figura 7.1), o una opción disponible para establecer la configuración de un punto de variación (se pueden observar las

características del *Inductor Triple* e *Inductor Cuádruple* de la Figura 7.1 en el que se puede seleccionar uno de ellos para configurar las características del *Hotplate*).

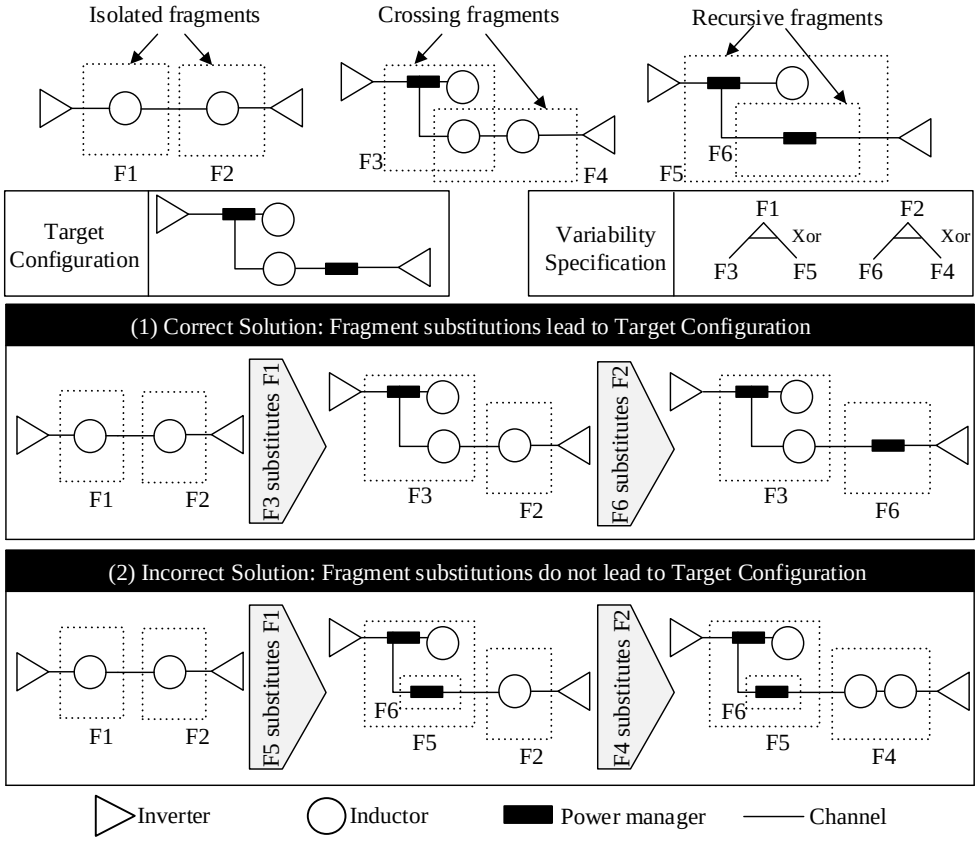


Figura 7.2: Ejemplos de configuraciones de producto correctas e incorrectas.

Después de la superposición de las características en los fragmentos del modelo, estos fragmentos del modelo pueden no compartir ningún elemento (fragmentos aislados) como muestra la parte superior-izquierda de la Figura 7.2. Por el contrario, también es posible que después de la superposición algunos elementos sean compartidos (fragmentos cruzados o *crossover*) como muestra la parte superior-central de la Figura 7.2. Además, todos los elementos de un fragmento de modelo pueden estar compartidos con otro fragmento de modelo (fragmentos recursivos) como representa la parte superior-derecha de la Figura 7.2.

Para conseguir configurar un producto determinado (por ejemplo, la parte central de la Figura 7.2 muestra una configuración de placa de inducción que se desea configurar), se realizan sustituciones de fragmentos de modelo. Una sustitución de fragmento de modelo es una operación que puede sustituir cualquier fragmento de modelo (un conjunto de elementos de modelo arbitrarios y las referencias entre ellos) con cualquier otro fragmento de modelo descrito en el mismo DSL de acuerdo con la especificación de variabilidad. La especificación de variabilidad que se muestra en la parte central de la Figura 7.2 indica que el fragmento modelo F1 puede sustituirse con el contenido de cualquiera de los fragmentos de modelo F3 o F5.

La parte inferior de la Figura 7.2 muestra dos ejemplos de sustituciones de fragmentos de modelo. Por un lado, la Figura 7.2 (1) muestra un ejemplo que consigue una solución correcta ya que el modelo de producto obtenido después de las sustituciones de fragmentos se corresponde con la configuración del producto objetivo. Esta solución correcta abarca dos sustituciones de fragmentos de modelo de la siguiente manera: primero el fragmento de modelo F3 sustituye el contenido del fragmento de modelo F1 y, en segundo lugar, el fragmento modelo F6 sustituye el contenido del fragmento de modelo F2. Por otro lado, el ejemplo que se muestra en la Figura 7.2 (2) ilustra una solución incorrecta porque la configuración del producto objetivo no se logra después de las sustituciones de los fragmentos.

En la url `http://svit.usj.es/variabilitytool.htm` se muestra un vídeo donde se realiza la configuración de un producto de placas de inducción mediante sustituciones de fragmentos de modelo en un entorno industrial.

7.4 Diseño del experimento

Como quedó reflejado en la sección 4.4 del capítulo 4, cada uno de los estudios empíricos desarrollados para al realización de este trabajo de tesis es representado con un modelo de características que refleja las propiedades del citado estudio. La figura 7.3 muestra destacando con un fondo gris todas aquellas características que forman parte del estudio. Algunas características incluidas son *Experimento* o *Pregunta de investigación*. Por contra, las características no presentes en este estudio han sido "suavizadas" como *Ingeniero de Usabilidad* o *Estudio de caso*. En este apartado se va a describir con detalle el diseño del experimento.

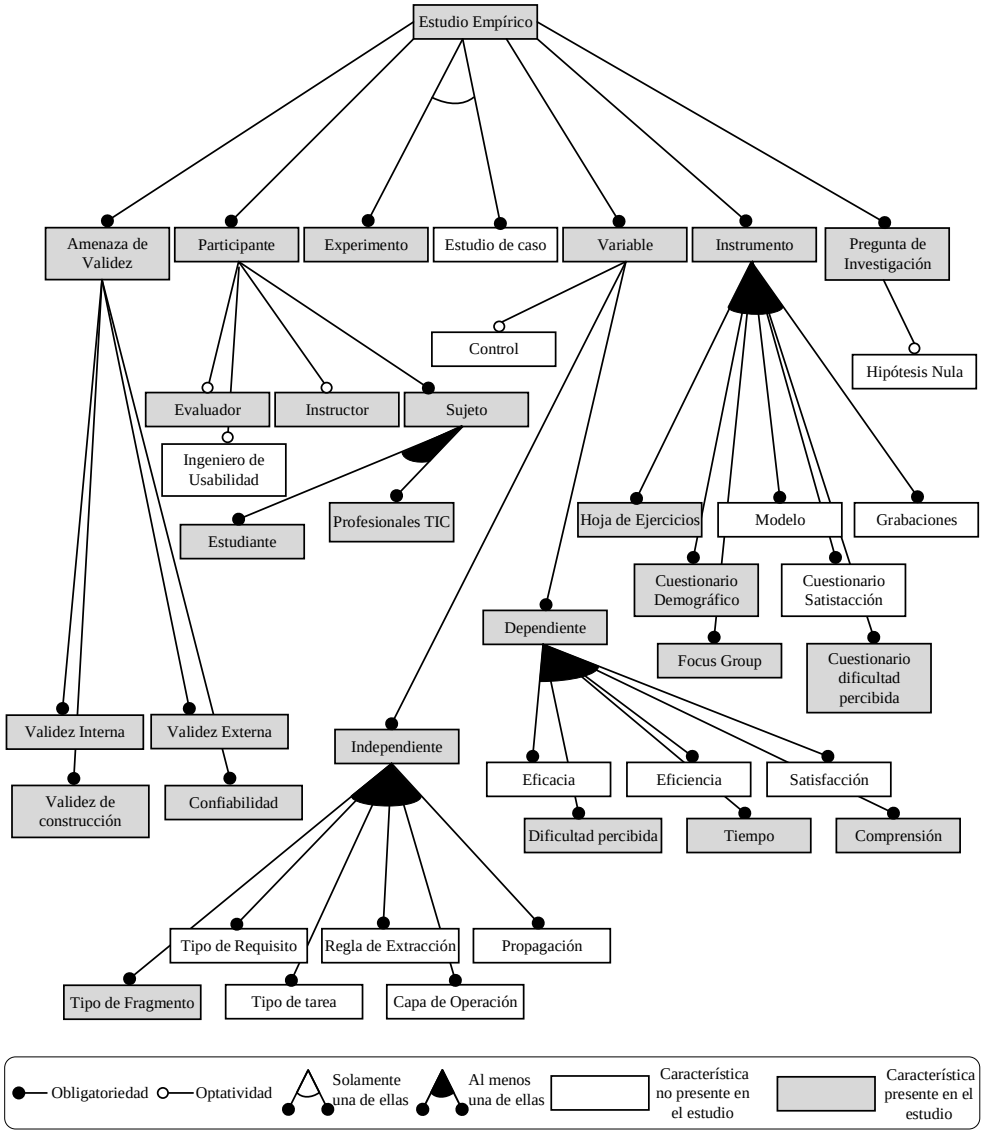


Figura 7.3: Modelo de características del experimento para evaluar la comprensión de fragmentos de modelo.

7.4.1 Objetivo del estudio y preguntas de investigación

El objetivo principal de la investigación descrita en este capítulo es determinar si hay dificultades en la comprensión de la variabilidad de fragmentos de modelo en la configuración de productos y si existen diferencias en la comprensión de fragmentos de modelo aislados, cruzados y recursivos para la configuración de productos. Nuestro experimento sigue el método presentado por Wohlin en [123], el objetivo del experimento fue:

Analizar si hay dificultades en la comprensión de fragmentos de modelo y las dificultades que tienen los usuarios con la comprensión de fragmentos de modelo aislados, cruzados y recursivos para configurar productos.

Con el propósito de incrementar el conocimiento sobre fragmentos de modelo mediante una evaluación empírica.

Desde el punto de vista de modeladores de productos a partir de fragmentos de modelo.

En el contexto de profesionales pertenecientes a empresas de desarrollo software y estudiantes de Ingeniería Informática.

Para conseguir el objetivo descrito anteriormente, buscamos responder a las siguientes preguntas de investigación:

RQ1: ¿Hay dificultades para comprender la variabilidad en los fragmentos de modelo para la configuración de productos?

RQ2: ¿Hay diferencias en la comprensión de fragmentos de modelo aislados, cruzados y recursivos en la configuración de productos?

Hasta donde sabemos, no existen teorías sobre la comprensión de fragmentos de modelo para la configuración de producto. Por esta razón, no hemos creado hipótesis relacionadas con las anteriores preguntas de investigación [148].

Para responder a las preguntas de investigación anteriores, utilizamos un diseño experimental. Las variables dependientes en nuestro experimento fueron el valor de comprensión (medido como el porcentaje de configuraciones de productos realizadas de forma correcta), el tiempo dedicado para realizar las configuraciones y la dificultad percibida por los sujetos al realizar configuraciones de producto. La variable independiente fue el tipo de fragmentos de modelo.

7.4.2 Participantes

Los sujetos fueron estudiantes de grado de Ingeniería Informática en la Universidad San Jorge de Zaragoza (España) y profesionales de empresas de desarrollo de software de Zaragoza. Había 20 estudiantes y 12 profesionales. Del total de sujetos que realizaron el experimento, 27 eran hombres (84 %) y 5 eran mujeres (16 %). La edad media de los sujetos era de 24.2 años. En el cuestionario demográfico, todos los estudiantes declararon que nunca habían trabajado como desarrolladores de software o como ingenieros de software. Por otro lado, todos los profesionales indicaron que tenían conocimiento sobre el modelado de software, conocimiento sobre los diagramas de características y estaban familiarizados con Lenguajes Específicos de Dominio. Además, todos los profesionales habían desarrollado software durante una media de 6.5 años.

Además de los citados sujetos, también participaron un instructor, dos observadores y dos expertos en modelado de la variabilidad. El instructor explicó con diapositivas el DSL utilizado para la realización de ejercicios, aclaró dudas durante el experimento y gestionó los *focus group*. Los observadores tomaron notas para análisis posteriores y comprobaron que la información era completada de una forma correcta por los sujetos. Finalmente, los expertos en modelado de la variabilidad, diseñaron los ejercicios, la corrección y corrigieron los ejercicios. Los expertos en modelado de la variabilidad no formaron parte del artículo de investigación al que dio lugar el experimento.

7.4.3 Definición de variables

En esta subsección se enumeran y definen todas las variables relevantes en el experimento.

Variables independientes

Realizamos un experimento de factor único donde la variable independiente es el tipo de fragmento de modelo, se trata de una variable nominal con tres valores: aislados, cruzados y recursivos. En la sección 7.3 se muestra una explicación detallada del significado de estos tres valores.

Variables dependientes

En nuestro experimento, los sujetos deben realizar nueve ejercicios de configuración para alcanzar un producto objetivo. Tres ejercicios corresponden al trabajo con fragmentos aislados, tres ejercicios utilizan fragmentos cruzados y tres ejercicios utilizan fragmentos recursivos. En cada ejercicio se deben realizar una serie de sustituciones de fragmentos que permiten desde un modelo base alcanzar el producto objetivo. En la realización de ejercicios medimos el valor de comprensión, el tiempo empleado y la dificultad percibida. Las variables dependientes se definen como:

- El valor de comprensión se define como el porcentaje de configuraciones de producto realizadas de forma correcta.
- El tiempo se define como el tiempo empleado (en minutos) para la realización de cada ejercicio.
- La dificultad percibida es la percepción que tienen los sujetos de la complejidad de un ejercicio. Se mide usando una escala *Likert*. Los sujetos deben completar un valor para la dificultad percibida para cada ejercicio.

7.4.4 Instrumentos

En esta subsección se muestra una descripción de todos los instrumentos utilizados para el desarrollo del experimento.

Cuestionario demográfico

El cuestionario demográfico tenía el objetivo de poder obtener información general sobre los sujetos y sus antecedentes. La información solicitada fue la edad, el género, el nivel académico, antecedentes como desarrolladores de software, experiencia en el ámbito de la Ingeniería de Software, conocimiento sobre modelado de software, conocimiento sobre especificaciones de variabilidad y conocimiento sobre DSL.

Para examinar objetivamente el conocimiento previo de los sujetos. Se les hicieron tres preguntas de comprensión sobre lenguajes de modelado genéricos. Cada pregunta tenía un enunciado y cuatro respuestas posibles: *Correcta*, *Incorrecta*, *No se puede responder con la información del modelo*, y *No lo sé.*

Material de entrenamiento

Este instrumento está compuesto por una copia en formato físico de las diapositivas utilizadas por el instructor para explicar el DSL utilizado en el experimento y los conceptos necesarios para realizar sustituciones en los modelos para alcanzar un producto objetivo.

Hoja de ejercicios

A cada sujeto participante en el experimento se le proporcionó una hoja de tareas por cada ejercicio a realizar. Cada ejercicio consiste en la configuración de un producto objetivo partiendo de modelo base y utilizando sustituciones con un conjunto de fragmentos de modelo determinado. Además del citado ejercicio cada hoja de tareas contiene un campo de texto para que el sujeto escriba la solución al ejercicio, y dos campos de texto para indicar el tiempo de inicio del ejercicio y el tiempo de finalización.

Es importante resaltar que los ejercicios fueron similares en complejidad (considerando número de elementos, límites y número de sustituciones) para lograr la misma configuración del producto objetivo utilizando fragmentos de modelo aislados, cruzados y recursivos.

Hoja de escala Likert

Esta hoja está compuesta por 9 escalas *Likert* y 9 áreas de texto, es decir, una escala y un área de texto por cada ejercicio. La escala comprende valores entre 1 y 7: 1 es muy fácil hasta 7 considerado muy difícil. Este valor de la escala se usa para calcular la dificultad percibida. En el área de texto, los sujetos podían escribir cualquier opinión que tuvieran sobre el proceso de configuración de productos utilizando los distintos tipos de fragmento de modelo.

Focus Group

La finalidad de realizar un *focus group* fue obtener datos cualitativos del proceso de configuradión de productos y la opinión de los sujetos sobre las soluciones de los ejercicios [125].

Los materiales utilizados en este experimento (el consentimiento para procesar los datos, el cuestionario demográfico, el material de entrenamiento, los ejer-

cicios, las preguntas abiertas y las notas del *focus group*) están disponibles en `http://svit.usj.es/productconfigurationexperiment`.

7.4.5 Procedimiento

Para desarrollar el experimento seleccionamos un diseño cruzado donde los sujetos aplicaban tres tratamientos (fragmentos aislados, cruzados y recursivos). Los ejercicios se dividieron en tres grupos dependiendo del tipo de fragmento involucrado en su resolución. En primer lugar, ejercicios con fragmentos de modelo aislados. En segundo lugar, los ejercicios con fragmentos de modelo cruzados. Finalmente, los ejercicios donde la superposición del características conformaba fragmentos de modelo recursivos. Para cada producto objetivo, hubo un ejercicio con fragmentos aislados, un ejercicio con fragmentos cruzados y un ejercicio con fragmentos recursivos. Al existir tres productos objetivo y tres ejercicios (uno por cada tipo de fragmento involucrado) para cada producto objetivo, hace un total de nueve ejercicios.

Los sujetos se dividieron, por razones de disponibilidad, en dos grupos para la realización del experimento. El primer grupo estaba formado por estudiantes del grado de Ingeniería Informática de la Universidad San Jorge, el segundo grupo estaba formado por profesionales de empresas de desarrollo de software. En ambos casos el desarrollo del experimento fue análogo y se desarrollo con los siguientes pasos:

1. En primer lugar, el instructor explicó las característcas del experimento.
2. Se pidió a los sujetos que completaran un cuestionario demográfico y firmaran un formulario de consentimiento para procesar los datos obtenidos.
3. Después de completar el cuestionario, el instructor (usando diapositivas) explicó el DSL de placas de inducción utilizado en los ejercicios. A los sujetos se les entregaron copias físicas de las diapositivas, estas copias podían ser consultadas por los sujetos durante la realización de los ejercicios.
4. Los sujetos realizaron los ejercicios. Para hacer cada ejercicio, se pidió a los participantes que configuraran el modelo DSL por medio de sustituciones de fragmentos. Después de la configuración del producto, el modelo resultante debe coincidir con el modelo objetivo del ejercicio para lograr la solución correcta (ver la Sección 7.3). No se impusieron restricciones de tiempo para desarrollar los ejercicios.

5. Tras la realización de cada ejercicio se anota el tiempo de finalización del ejercicio. Un observador comprueba la corrección de los datos.

6. Se solicitó a los sujetos que completaran una escala *Likert* sobre la dificultad percibida para cada ejercicio y que anoten cualquier comentario sobre la realización del ejercicio. Las respuestas de la escala *Likert* se usaron para calcular la dificultad percibida en la configuración de productos.

7. Antes de ser entregado al sujeto el siguiente ejercicio un observador verificaba que el sujeto hubiera completado la información requerida en la hoja de escala *Likert*.

8. Finalmente, se realizó un *focus group* [4] donde los sujetos expresaron sus opiniones sobre el proceso de ejecución de los ejercicios. Los observadores anotaron las opiniones expresadas por los sujetos.

7.5 Resultados

En esta sección, primero presentamos los efectos de diferentes fragmentos de modelos en el valor de comprensión y el tiempo dedicado a completar los ejercicios. Posteriormente, presentamos la dificultad percibida por los sujetos al realizar los ejercicios con los diferentes fragmentos del modelo.

Para analizar los resultados, utilizamos como variables dependientes: el valor de comprensión (como el porcentaje de ejercicios realizados correctamente), el tiempo necesario para realizar los ejercicios y la dificultad percibida por los sujetos. De acuerdo con una prueba de Shapiro-Wilk [149], el valor de comprensión, el tiempo para realizar los ejercicios y la dificultad percibida siguen una distribución normal.

Antes de realizar nuestro análisis, verificamos si la variable de control "tipo de participante", es decir, estudiante o profesional, tuvo una influencia en el valor de la comprensión y en el tiempo para realizar las tareas. Ya que "tipo de participante"no tuvo influencia, decidimos no incluirlo en las pruebas estadísticas.

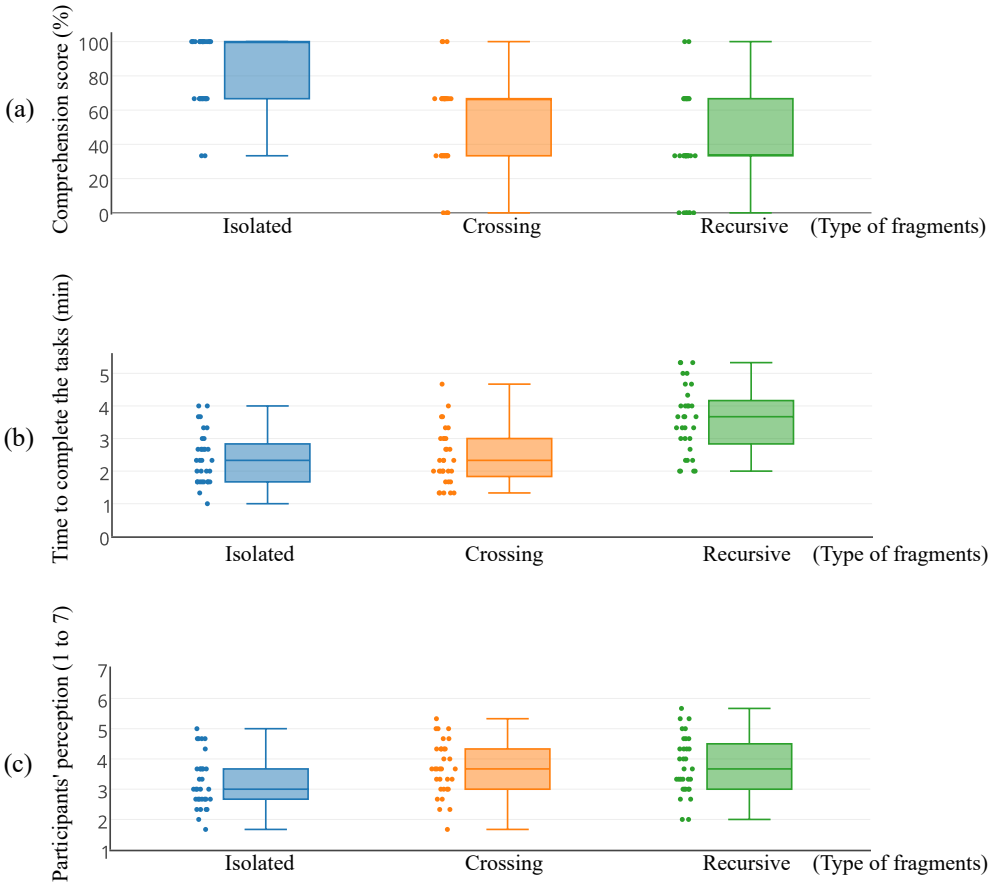


Figura 7.4: (a) Valor de comprensión. (b) Tiempo empleado en la resolución de ejercicios. (c) Dificultad percibida.

7.5.1 Valor de comprensión

Se realizó un test ANOVA [138] para comparar el valor de comprensión obtenido para los distintos tipos de fragmento del modelo. Hubo un efecto significativo de los diferentes tipos de fragmento del modelo en el valor de comprensión para $p<0.05$ para fragmentos aislados, fragmentos cruzados y fragmentos recursivos [$F(2,93) = 22.729$, $p = 0.000$]. La Figura 7.4 (a) muestra que los ejercicios con fragmentos aislados fueron los más fáciles de entender y realizar correctamente (82.29 %), seguidos por los ejercicios compuestos por fragmentos cruzados(51.04 %) y, finalmente, fragmentos recursivos (40.62 %). Realizamos t-tests para averiguar qué tipos de fragmentos de modelo tenían diferencias significativas entre sí en términos de comprensión. Las diferencias entre el valor de comprensión para fragmentos aislados y los valores para fragmentos cruzados y fragmentos recursivos fueron significativas. Sin embargo, no hubo diferencias significativas entre el valor de comprensión entre fragmentos cruzados y fragmentos recursivos.

Aplicamos el test *t-student* para analizar el efecto de la diferencia entre los sujetos. Aunque los profesionales obtuvieron mejores resultados que los estudiantes para los diferentes tipos de ejercicios, como ya se ha comentado anteriormente, el efecto de los sujetos no fue significativo.

7.5.2 Tiempo

Parea analizar el tiempo necesario para completar los ejercicios utilizamos un test ANOVA. Los resultados mostraron que hubo un efecto significativo de los diferentes modelos de fragmentos sobre el tiempo necesario para realizar los ejercicios [$F(2,93) = 17.512$, $p = 0.000$]. La Figura 7.4 (b) muestra que los sujetos necesitaron más tiempo para realizar los ejercicios con fragmentos recursivos (una media de 3.55 minutos); los ejercicios que contenían fragmentos aislados y fragmentos cruzados fueron realizados por los sujetos en la misma cantidad de tiempo (una media de 2.41 minutos). Realizamos t-tests para averiguar que tipos de fragmentos de modelo diferían significativamente unos de otros en términos de tiempo. El t-test mostró que no había diferencias significativas entre los fragmentos aislados y los fragmentos cruzados. Por el contrario, las diferencias entre los ejercicios de fragmentos recursivos y el resto de ejercicios fueron significativos.

Con respecto al efecto de los diferentes sujetos, los estudiantes realizaron los ejercicios más rápido que los profesionales. Los estudiantes emplearon de media (2.26, 2.16 y 3.48 minutos) y los profesionales (2.61, 2.88 y 3.72 minutos)

para la realización de ejercicios con fragmentos de modelo aislados, cruzados y recursivos, respectivamente. El t-test muestra que los tiempos medios no difirieron significativamente cuando analizamos los tiempos entre diferentes sujetos.

7.5.3 Dificultad percibida

Se realizó un test ANOVA para analizar la dificultad percibida por los sujetos en la realización de ejercicios con diferentes fragmentos del modelo. Hubo un efecto significativo del tipo de modelo de fragmentos sobre la dificultad percibida con $p<0.05$ para fragmentos aislados, fragmentos cruzados y fragmentos recursivos [$F(2,93) = 3.561$, $p = 0.032$]. La Figura 7.4 (c) muestra que los ejercicios con fragmentos recursivos (3.79) y fragmentos cruzados (3.70) fueron más difíciles de entender que los ejercicios con fragmentos aislados (3.23). Realizamos t-test para conocer la percepción de los usuarios de qué modelo fragmentos eran significativamente diferentes entre sí. Las diferencias entre la dificultad percibida de los sujetos entre fragmentos de modelo aislados con el resto de fragmentos de modelo (cruzados y recursivos) fueron significativas.

7.5.4 Comentarios escritos por los sujetos

Los sujetos tenían la posibilidad de escribir libremente cualquier comentario que ellos consideraran de interés sobre los ejercicios realizados. En un ejercicio con fragmentos cruzados, un participante dijo: "No entiendo que sucederá con la intersección en la primera sustitución". Otro sujeto pensó que un fragmento es similar a un booleano: "He supuesto que un inductor y otro se niegan entre sí, y eso es lo mismo que no utilizar ninguno".

Para los ejercicios con fragmentos recursivos, un sujeto dijo: "Las sustituciones con fragmentos internos tienen un comportamiento ambiguo". Otro sujeto dijo: "Con diferentes niveles, si necesita introducir piezas en otras piezas, se hace más difícil". En referencia a las sustituciones de fragmentos, un sujeto dijo: "Cuando había muchos puntos de interconexión, no sabía si la sustitución era correcta".

7.5.5 Focus group

Cuando los sujetos de cada grupo terminaron los ejercicios, se realizó un *focus group*. Un sujeto preguntó: "¿Debía llenar cada hueco (punto de variación)?". Otro sujeto preguntó: "¿Puedo usar la misma pieza (fragmento) varias veces?". Un sujeto declaró: "Creo que la sustitución de fragmentos es correcta si las formas geométricas coinciden entre sí". Otro sujeto dijo: "Yo uní un fragmento con otro fragmento porque sus formas encajaban ".

El análisis de los datos permite responder a las preguntas de investigación de la siguiente manera:

- *RQ1: ¿Hay dificultades para comprender la variabilidad en los fragmentos de modelo para la configuración de productos?* Como se puede apreciar en la Figura 7.4 (a), no todos los ejercicios de configuración del producto fueron resueltos correctamente. Específicamente, en el 42,01 % de los ejercicios no se pudo realizar una configuración de producto que coincidiera con el producto objetico a lograr.
- *RQ2: ¿Hay diferencias en la comprensión de fragmentos de modelo aislados, cruzados y recursivos fragmentos en la configuración de productos?* Solo los ejercicios que incluyen fragmentos de modelo aislados para la configuración del producto fueron comprendidos de una forma satisfactoria por la mayoría de sujetos (82.29 %). Por el contrario, los ejercicios que incluyen fragmentos de modelo cruzados o fragmentos de modelo recursivos obtuvieron los peores resultados (51.04 % y 40.62 % respectivamente). En la siguiente sección analizan estos resultados y se resaltan los hallazgos de esta investigación.

7.6 Análisis de los resultados

Algunos sujetos declararon que la formas geométricas de los fragmentos del modelo eran significativamente importantes para la configuración del producto. Específicamente, los sujetos consideraron un fragmento de modelo como una opción válida si su forma geométrica se ajusta a la forma geométrica de un punto de variación. Esto es especialmente importante porque en algunos casos, los sujetos dieron a la forma geométrica más prioridad que a la propia especificación de variabilidad, este criterio de forma geométrica puede convertirse en una fuente de errores. Este criterio produjo configuraciones incorrectas en el 27,5 % de los ejercicios en los que se ha utilizado. La sintaxis concreta de fragmentos de modelo genera confusión en las configuraciones de productos,

esta conclusión coincide con resultados de investigaciones anteriores [27]. Esta confusión hizo que los sujetos considerasen opciones válidas y las usaron en una configuración de producto aunque esas opciones no eran válidas según la especificación de variabilidad.

Aunque el criterio anterior de las formas geométricas, en ocasiones, es una fuente de errores, también medimos en este experimento, el tiempo que tardaban los usuarios en realizar los ejercicios. Detectamos que los ejercicios donde era aplicado el criterio de forma geométrica y se obtenía una solución correcta, el tiempo empleado en la configuración del producto se reducía en un 38 %. Previamente hemos considerado el criterio de forma geométrica como una fuente de soluciones incorrectas, pero esto el experimento revela que este criterio también podría reducir significativamente el tiempo requerido para configurar los productos. Estos resultados sugieren que los enfoques de modelado de variabilidad general deberían idear un mecanismo para alinear la forma geométrica y la especificación de variabilidad con el fin de aumentar la eficiencia y la eficacia en los procesos de configuración de productos.

Los sujetos también mencionaron que combinaron intuitivamente fragmentos de modelos (creando un nuevo fragmento de modelo) para inyectar la combinación en un punto de variación. Esta situación se reveló en la corrección en los ejercicios de 6 sujetos; crearon nuevos fragmentos al unir otros fragmentos. El modelado de variabilidad debería incluir combinaciones de fragmentos en la biblioteca. Los sujetos dijeron que estas combinaciones de fragmentos de modelo les ayudaron a resolver más fácilmente los ejercicios. Los resultados revelan que los sujetos que combinaron el modelo fragmentos (34,4 % de los sujetos), redujeron tanto el número de sustituciones (una media del 33,18 %) para alcanzar la configuración del producto objetivo y el número de errores (un promedio del 14 %) con respecto a los sujetos que no combinaron los fragmentos del modelo. Sin embargo, esta operación no es compatible con los enfoques de modelado de variabilidad general (*Feature Modeling*, OVM y CVL). Como nuestros resultados revelan que esta operación para combinar fragmentos de modelos es positiva para la configuración del producto, los nuevos enfoques de modelado de variabilidad, y en general los enfoques de modelado, deberían considerar la inclusión de esta operación.

Los resultados muestran que los profesionales, sin que estadísticamente sea significante, mejoraron el porcentaje de soluciones correctas en aproximadamente un 10 %, sin embargo, empeoraron el valor del tiempo para realizar los ejercicios respecto a los estudiantes. Este resultado está en línea con anteriores investigaciones [80]. La investigación había demostrado que los sujetos con experiencia resuelven las tareas correctamente, pero que su conocimiento puede

aumentar sus dudas, por lo tanto, requieren más tiempo. Nuestros resultados también muestran que la experiencia hace incrementar la percepción de la dificultad. Si analizamos el porcentaje de sustituciones correctas de fragmentos y la significancia estadística del valor de comprensión, todos los sujetos obtuvieron resultados similares. Esto puede indicar que el modelado de variabilidad no solo facilita su adopción por profesionales, también facilita la adopción por parte de usuarios noveles (estudiantes).

Los resultados también muestran que los sujetos obtuvieron los mejores resultados cuando se utilizan fragmentos del modelo aislados (la media de las soluciones correctas es 82.29 %). Este resultado coincide con trabajos de investigación anteriores [81], que utilizaron fragmentos de modelos aislados para realizar configuraciones de productos. Los fragmentos de modelo cruzados obtuvieron una media del 51,04 % de soluciones correctas. En la comunidad científica exsiten estudios conocedores de la dificultad para entender los modelos de fragmento cruzados [88, 82, 86]. Los fragmentos del modelo recursivos obtuvieron los peores resultados (una media de 40,62 % de soluciones correctas). En general, los sujetos obtuvieron soluciones incorrectas cuando se incrementaban los niveles de recursividad para alcanzar la configuración objetivo. Según nuestro conocimiento, los modelos de fragmento recursivos no han sido tratados por la comunidad científica. Los resultados de este experimento sugieren que los fragmentos de modelo recursivos son la principal fuente de errores para alcanzar una configuración de producto objetivo. Por lo tanto, es necesario realizar nuevos experimentos para investigar más a fondo los fragmentos de modelo recursivos en el proceso de configuraciones de productos.

Cuando se corrigieron los ejercicios, notamos que 12 sujetos realizaron sustituciones de fragmentos que fueron correctas pero innecesarias (los sujetos sustituyeron un fragmento con un fragmento que era exactamente el mismo). En otras palabras, los sujetos hicieron sustituciones de fragmentos redundantes porque pensaban que los puntos de variación tienen que ser siempre sustituido con una de sus opciones disponibles. Ellos consideraron las sustituciones como obligatorias aunque el punto de variación ya contiene los elementos del modelo del producto objetivo. Una posible explicación para estas sustituciones redundantes de fragmentos es el material utilizado en los tutoriales, que enfatiza que un conjunto de sustituciones elimina elementos de un fragmento a sustituir e inyecta el fragmento sustituto. Por lo tanto, el material utilizado a modo de tutorial debería destacar el hecho de que la sustitución de un fragmento no es obligatoria para cada fragmento susceptible de ser sustituido para lograr una configuración de producto. Según los sujetos, el diseño del árbol de la especificación de variabilidad (por ejemplo, la especificación de variabilidad que

puede ser observada en la mitad superior de la Figura 7.1) refuerza la idea de los sujetos de que deberían elegir una opción para cada punto de variación. Los resultados muestran que el 5,9 % de los ejercicios incluyeron sustituciones de fragmentos redundantes, que aumentan en un 6,93 % las sustituciones necesarias de fragmentos para alcanzar la configuración del producto objetivo. Es importante resaltar que no encontramos instrucciones para evitar esta redundancia en el materiales [1, 56, 45] donde han sido abordados enfoques de modelado de variabilidad, y que fueron utilizados en este experimento para entrenar a los sujetos. Por lo tanto, los resultados de este experimento sugieren que los materiales de entrenamiento para abordar el modelado de la variabilidad deberían ser aumentados de forma explícita para evitar esta redundancia.

#	Result	Type	Next step
1	Concrete syntax of model fragments misleads product configuration.	C [27]	New versions of concrete syntax of model fragments should align geometric shape and variability specification in order to take advantage of the time improvement.
2	Geometric shape criterion reduces the time required (38 % less time) to perform product configurations.	N	
3	Model fragment combination improves the efficiency (33.18 % less steps) of product configurations and reduce the incorrect solutions (14 %).	N	General variability modeling approaches should consider the inclusion of the combination operation as a complement for product configuration.
4	Highest percentage of correct solutions in the isolated model fragments (82.29 % of correct solutions).	C [81]	Specific experiments need to be conducted to further investigate recursive model fragments in product configurations.
5	Crossing model fragments are difficult to understand (51.04 % of correct solutions).	C [88, 82, 86]	
6	Recursive model fragments are the major source of incorrect solutions (40.62 % of correct solutions).	N	
7	Redundant configuration steps are produced by misunderstanding variation points as mandatory substitutions.	N	Training materials of the variability modeling approaches should be extended to explicitly avoid this redundancy.

Tabla 7.1: Resumen de resultados del experimento con fragmentos de modelo.

La Tabla 7.1 resume los principales hallazgos del trabajo experimental descrito en este capítulo que son relevantes para abordar de una forma general el modelado de variabilidad. Cada resultado presentado en la citada tabla está etiquetado como tipo C (el hallazgo confirma los resultados de trabajos de investigación previos) o tipo N (nuevo descubrimiento revelado por este trabajo). Finalmente, la tabla también resume nuestras sugerencias que deben ser tomadas en los siguientes pasos para abordar el modelado de variabilidad de una forma general.

7.7 Amenazas a la validez

Para describir las amenazas a la validez del experimento descrito en este capítulo utilizamos la clasificación propuesta por [116], que distingue cuatro tipos de amenazas a la validez y que nos permiten conocer las limitaciones de nuestro experimento.

Validez de construcción: este aspecto de validez refleja la medida en que las medidas operativas que se estudian representan lo que los investigadores tienen en cuenta y lo que se investiga sobre la base de la investigación preguntas. En este trabajo, los ejercicios propuestos no tienen una respuesta verdadera/falsa. Por lo tanto, es muy difícil para los sujetos responder correctamente si no entienden de una manera clara y concisa cuáles son las respuestas a los ejercicios. Además, para minimizar este tipo de amenaza, los ejercicios, respuestas y correcciones fueron diseñados por dos expertos en modelado de variabilidad. Estos expertos han desarrollado herramientas de modelado de variabilidad en entornos industriales (en el dominio de las placas de inducción y en el dominio de software de control de trenes). Su participación se limitó a la descrita anteriormente y no estuvieron involucrados directamente en el artículo de investigación generado a partir del documento. Finalmente, las medidas utilizadas en nuestra investigación son el valor de la comprensión (porcentaje de configuración realizado de forma correcta), tiempo empleado en la resolución de los ejercicios y dificultad percibida capturada utilizando una escala Likert. Estas medidas son ampliamente aceptadas en comunidad de investigación del ámbito de la Ingeniería del Software [81, 80].

Validez interna: existe el riesgo de que el factor que se investiga pueda ser afectado por otros factores descuidados. Para mitigar esta amenaza, en el experimento descrito en este capítulo, se explicó el DSL utilizado a los sujetos. Las diapositivas utilizadas durante la citada explicación se les dieron a los sujetos, para que la falta de comprensión de DSL no fuese un problema en la realización de los ejercicios.

Validez externa: este tipo de amenaza a la validez aborda hasta qué punto es posible generalizar los resultados, y en qué medida los resultados son extrapolables o generalizables para otros casos. El experimento fue realizado por estudiantes y profesionales, y la participación de estudiantes puede ser una debilidad. Sin embargo, usar estudiantes como sujetos en lugar de ingenieros de software no es un problema importante [84, 150] siempre y cuando las preguntas de investigación no estén específicamente enfocadas hacia los "profesionales" o ingenieros del software. Consideramos que los conceptos de modelado de variabilidad tratados en este capítulo también son relevantes pa-

ra los estudiantes. Por lo tanto, los estudiantes son sujetos aceptables. Otro problema importante es el pequeño número de participantes (32). Sin embargo, dos roles importantes fueron cubiertos por los sujetos. Una rol es el de desarrollador de software sin conocimiento profesional de modelado (estudiantes), y el otro rol es el ingeniero de software con conocimiento de modelado (profesionales). En el procedimiento de análisis, hemos utilizado un intervalo de confianza $p<0.05$ donde las conclusiones son representativas del 95 %. Esto significa que siguiendo una distribución normal, los resultados serán verdaderos el 95 % de las veces. Dado que la DSL utilizado en este experimento es un lenguaje muy sencillo en un dominio específico, creemos que la generalización de los resultados deben realizarse con precaución. El DSL seleccionado es apropiado para una sencilla comprensión por parte de los sujetos. Sin embargo, se deben realizar otros experimentos con otros DSL para validar nuestros resultados y recomendaciones.

Fiabilidad: este aspecto se refiere a qué medida en que los datos y el análisis dependen de los investigadores específicos. Para reducir esta amenaza, dos expertos en modelado de variabilidad diseñaron los ejercicios, respuestas y correcccciones. Por otro lado, realizamos esta investigación utilizando métodos que son ampliamente aceptados por la comunidad investigadora de la Ingeniería del Software.

7.8 Conclusiones

En trabajos anteriores al presentado en este capítulo [27], detectamos que los expertos de dominio en el ámbito del desarrollo de software para placas de inducción de nuestro socio indusrial (grupo BSH) tuvieron dificultades para entender la variabilidad para configurar el firmware de sus productos (placas de inducción bajo las marcas Bosch y Siemens). Específicamente, los expertos en el dominio tenían dificultades para comprender la variabilidad en fragmentos de modelo que resultan de la superposición de características [58], siguiendo la especificación de variabilidad en un modelo DSL.

En el trabajo presentado en este capítulo, llevamos a cabo un experimento en el cual los participantes lidian con la variabilidad en los fragmentos de modelo para lograr las configuraciones de producto deseadas. Los ejercicios se dividieron en tres grupos según el tipo de fragmentos producido por la superposición de las características en el modelo DSL: fragmentos de modelo aislados, fragmentos de modelo cruzados y fragmentos de modelo recursivos. Medimos el valor de comprensión, el tiempo empleado para la realización de los ejercicios

y la dificultad percibida por los sujetos participantes en el experimento. Además, obtuvimos la opinión de los sujetos sobre los ejercicios mediante preguntas abiertas y un *focus group*.

Nuestros resultados muestran que los hallazgos son relevantes para la investigación de los enfoques generales para el modelado de la variabilidad (FODA, OVM y CVL). Específicamente, los resultados muestran cuatro nuevos hallazgos revelados por este trabajo:

- El criterio de forma geométrica reduce el tiempo requerido para realizar las configuraciones de producto.
- La combinación de fragmentos de modelo mejora la eficacia en la configuración de productos y reduce las soluciones incorrectas.
- Los fragmentos de modelo recursivos son la principal fuente de soluciones incorrectas.
- Los pasos de configuración redundantes se producen por malentendidos con el funcionamiento de los puntos de variación: son considerados como sustituciones obligatorias.

Y tres hallazgos que confirman los resultados de trabajos de investigación anteriores:

- La sintaxis concreta de los fragmentos del modelo engaña en los procesos de configuración de productos.
- El porcentaje más alto de soluciones correctas se produce cuando se trabaja con fragmentos de modelo aislado.
- Los fragmentos de modelo cruzados son difíciles de comprender.

Teniendo en consideración los hallazgos anteriores, sugerimos los aplicación de las siguientes ideas para mejorar el enfoque sobre el modelado de variabilidad: (1) nueva sintaxis concreta de fragmentos de modelo, (2) inclusión de la operación de combinación de fragmentos de modelo, (3) investigar más a fondo los fragmentos de modelo recursivos en las configuraciones del producto y (4) aclaración en los materiales de aprendizaje sobre el enfoque de modelado de variabilidad. Abordar estas ideas contribuiría a promover la adopción de los nuevos enfoques de gestión de la variabilidad en la industria.

8

Gestión de errores

Índice

8.1 Resumen del capítulo

La gestión de errores puede ser compleja en la práctica en entornos industriales. En estos entornos, miles de productos software comparten características en su configuración. A pesar de la importancia y la complejidad que conlleva la corrección de errores, todavía faltan datos empíricos sobre las dificultades encontradas en las Líneas de Productos Software industriales (SPLs) cuando se desarrollan estas tareas. Este capítulo tiene como objetivo evaluar el rendimiento de los ingenieros de software cuando corrigen errores y propagan estas correcciones a otros productos configurados en el contexto de una SPL industrial.

Diseñamos y llevamos a cabo un estudio empírico para recopilar datos con respecto a las tareas de corrección de errores en el contexto de una SPL de la divisón de placas de inducción en el grupo BSH, uno de los mayores fabricantes de electrodomésticos de Europa.

Al realizar el estudio empírico, hallamos que la eficacia, la eficiencia y la satisfacción de los ingenieros de software alcanzaron buenos valores. Mediante las entrevistas realizadas también encontramos dificultades relacionadas con las *features* no utilizadas, las operaciones de clonación realizadas de forma no intencionada, la detección de *features* modificadas y la propagación de las correcciones realizadas cuando en el origen del error se producen interacciones entre distintas *features*. Las dificultades identificadas son relevantes para saber cómo aplicar mejor las SPLs en la industria.

8.2 Introducción

Las Líneas de Productos Software se han utilizado en una gran cantidad de dominios como una forma para lograr mejoras de calidad, reutilización extensiva y productividad de desarrollo [2]. En las SPLs, los ingenieros de línea de producto pueden crear una familia de productos configurando cada producto mediante la selección y personalización de diferentes características (*features*). En las SPLs en entornos industriales, la corrección de errores puede ser compleja para los ingenieros de líneas de productos, ya que en el catálogo de productos, miles de productos pueden compartir características entre ellos.

La mayoría de los trabajos existentes desarrollados mediante estudios empíricos en el ámbito de las SPLs y relacionados con la gestión de errores se centran en analizar las técnicas de prueba para buscar errores, pero no se ocupan de cómo solucionar esos errores. Por ejemplo, en [97] se realizó una evaluación en

un entorno industrial utilizando una técnica de test de SPL por pares llamada MoSo-PoLiTe. En [99] se propone una técnica aleatoria de test para SPL. En [94] se evalúa una técnica de test que genera incrementalmente pruebas para SPLs. Existen otros enfoques que han estudiado la fiabilidad de las mejoras en la evolución de la SPL a través de estudios empíricos, un ejemplo es mostrado en [96]. En este último trabajo de investigación se analiza hasta qué punto las partes comunes y los puntos de variación cambian con el tiempo. El análisis se centra en la línea de productos de Eclipse entre 2007 y 2010, teniendo en cuenta las tendencias de errores, las tendencias de cambio y fallos relacionados como variables. Otros enfoques como el caso mostrado en [98] han evaluado cómo el uso de patrones puede mejorar la gestión de errores.

El objetivo de este capítulo es evaluar la gestión de errores de una SPL en la división de placas de inducción en el grupo BSH. BSH es el mayor fabricante de electrodomésticos en Europa y uno de los principales fabricantes del sector en todo el mundo. Su división de placas de inducción ha estado produciendo placas de inducción (el portfolio de la marca está compuesto por Bosch y Siemens entre otros) durante los últimos 15 años.

Para evaluar la forma de solucionar errores, planificamos un estudio de caso con los ingenieros de la división de placas de inducción como sujetos. Los ingenieros tuvieron que realizar un conjunto de tareas para solucionar un error en un producto configurado y propagar, cuando las circunstancias lo requieren, la solución a otros productos configurados. Los resultados obtenidos muestran la aparición de dificultades para solucionar errores en las SPLs con respecto a las *features* no utilizadas, las *features* que aparecen por clonación de forma involuntaria, la detección de *features* modificadas y la propagación de la solución propuesta cuando el origen del error está en la interacción entre *features*. Estas dificultades detectadas son relevantes para las aproximaciones de SPLs y su tratamiento puede contribuir a promover la adopción de SPLs en entornos industriales.

8.3 SPLs y Errores

Las SPLs requieren herramientas que les permitan a los usuarios configurar productos que se ajusten a los requisitos comerciales o a las necesidades del mercado y que puedan ser configurados al variar las *features* que componen cada producto. Hasta ahora, muchas herramientas de modelado de variabilidad como pure::variants (`www.pure-systems.com`) y Gears (`www.biglever.com`) se han creado para poder trabajar bajo el paradigma de las SPLs. Aunque

los detalles de implementación son diferentes, estas herramientas incluyen los siguientes elementos clave:

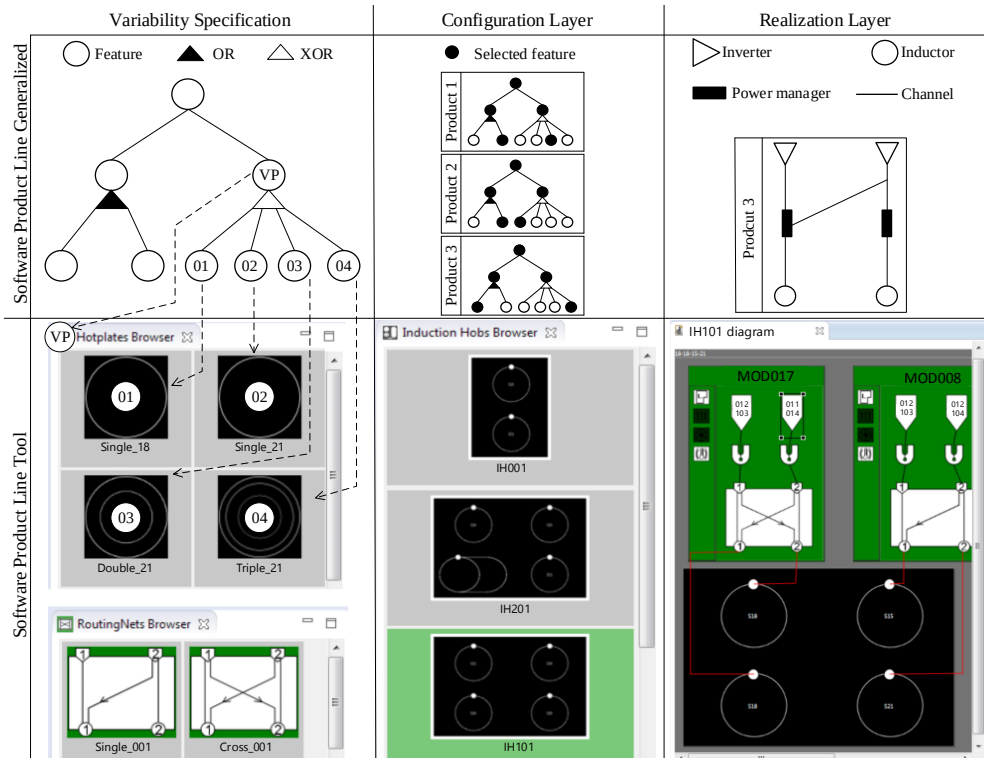


Figura 8.1: Herramienta SPL.

Especificación de variabilidad. Representa la información de todas las posibles oportunidades de variación de una SPL (puntos de variación). Una representación ampliamente utilizada para los puntos de variación son los modelos de características (*feature models*) [45], que representan todas las posibles oportunidades de variación en términos de características (*features*) y relaciones entre ellas como un conjunto jerárquicamente organizado. Por ejemplo, una especificación de variabilidad que usa un modelo de característica se muestra en el lado superior izquierdo de la Figura 8.1. Para cada punto de variación (ver VP en la figura), la herramienta de configuración del producto muestra un conjunto de opciones que se pueden seleccionar.

El lado inferior izquierdo de la Figura 8.1 muestra la herramienta que BSH usa para configurar las placas de inducción (*induction hobs, IHs*) en la que

se muestra un catálogo de *hotplates* disponible, ya que los *hotplates* actúan como puntos de variación en el modelo de caracterísitcas. El catálogo contiene una gran cantidad de elementos que especifican las posibilidades existentes de configuración para ese punto de variación en el modelo de características. Como muestra la Figura 8.1, el catálogo de placas de inducción está formado por cuatro tipos diferentes de *hotplates*, debido a la especificación de cuatro características (ver las características de 01 a 04 en la figura) para el *hotplate* que actúa como punto de variación.

Capa de configuración (*Configuration Layer, CL*). Es un portfolio con diferentes productos configurados. Las diferencias que existen entre los diferentes productos configurados son las características que se seleccionan en la especificación de variabilidad. Por ejemplo, el lado superior-central de la Figura 8.1 muestra que el portfolio está formado por 3 productos diferentes, que tienen diferentes características seleccionadas en la especificación de variabilidad (ver los productos del 1 al 3 en la figura).

En la SPL de BSH, una representación gráfica representa cada placa de inducción que compone el portfolio de placas como muestra la captura de la herramienta BSH en el lado inferior-central de la Figura 8.1.

Capa de realización (*Realization Layer, RL*). Esta capa permite la personalización de las características que se han seleccionado para un producto determinado en la especificación de variabilidad. Las características son simplemente símbolos gráficos que se materializan en esta capa a fragmentos de código de un lenguaje de programación (por ejemplo Java) o modelan fragmentos que se expresan utilizando lenguajes de modelado de propósito general (por ejemplo UML) o lenguajes de modelado específicos de dominio (por ejemplo Lenguaje Específico de Dominio de Placas de Inducción, *Induction Hob Domain Specific Language, IHDSL*). A modo de ejemplo, el lado superior derecho de la Figura 8.1 muestra el modelo de *Product 3* que se obtiene materializando cada característica seleccionada en su fragmento de modelo. Específicamente, este modelo se expresa utilizando IHDSL, cuyos conceptos principales son los siguientes: inversor (*inverter*), inductor (*inductor*), gestor de potencia (*power manager*) y canal (*channel*).

El lado inferior derecho de la Figura 8.1 muestra una captura de la herramienta SPL de BSH que muestra un modelo con la realización de un producto (una placa de inducción) identificado como *IH101*.

Con el paso del tiempo, puede ser necesario corregir errores en el desarrollo industrial (por ejemplo, se ha informado de un error en un determinada placa

de inducción debido a problemas de rendimiento en un *hotplate*). En el contexto de una SPL, la corrección de errores podría abarcar las siguientes tareas:

- **Corrección de errores en la capa de configuración.** Esto implica la sustitución de una característica seleccionada por una diferente. Por ejemplo, reemplazar la característica seleccionada *single18 hotplate* con la característica *triple21 hotplate*.
- **Corrección de errores en la capa de realización.** Esto implica la modificación del código o los elementos de un producto configurado. Por ejemplo, cambiar el parámetro *VMAX* de un inversor a 42.
- **Propagación de corrección de errores.** Esto podría hacerse para extender la corrección a otros productos configurados. Por ejemplo, el producto *IH101* se ha corregido, pero también puede ser necesario propagar la solución proporcionada a otros productos (otras placas de inducción) que comparten características con el producto *IH101*.

8.4 Diseño del estudio de caso

En la Figura 8.2 quedan reflejadas las característcas principales del estudio de caso desarrollado para estudiar la corrección de errores en SPLs. Destacado con un fondo gris se muestran aquellas características que forman parte del estudio de caso (*Estudio de caso* o *Pregunta de investigación*). Por contra, las características no presentes en este estudio han sido "suavizadas" (*Estudiante* o *Experimento*).

8.4.1 Objetivo

El objetivo de este estudio empírico es evaluar el rendimiento de los ingenieros de software cuando corrigen errores y propagan estas soluciones a otros productos configurados en el contexto de una SPL de la división de placas de inducción de la compañía BSH. En entornos industriales reales las empresas se encuentran con la necesidad de administrar miles de características y productos (a modo de ejemplo, la herramienta de configuración de productos BSH administra aproximadamente 2^{100} configuraciones diferentes de productos). Por esta razón la corrección de errores puede ser difícil de gestionar por los ingenieros de Líneas de Productos Software.

Siguiendo las directrices descritas por Wohlin en [123], el objetivo de nuestro estudio de caso fue:

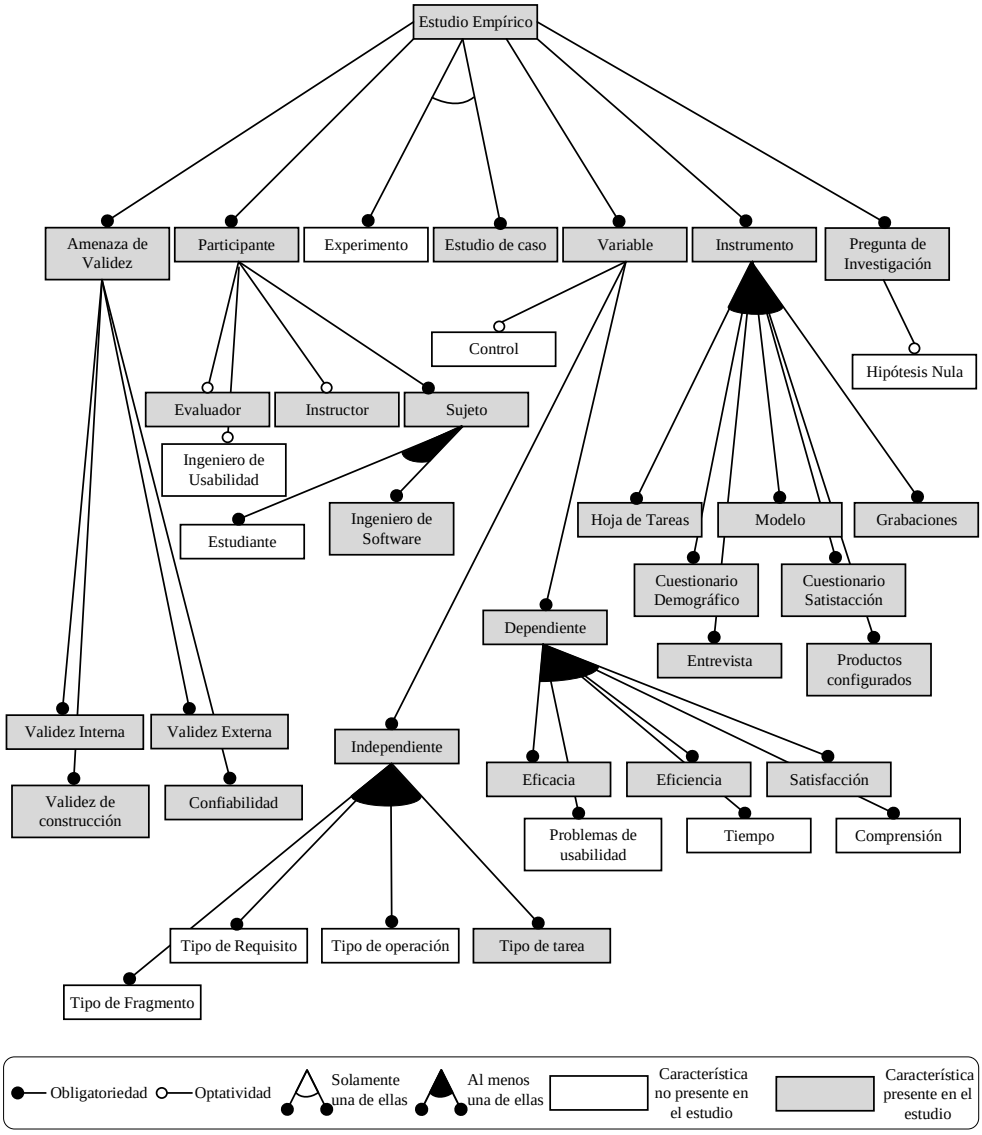


Figura 8.2: Modelo de características del estudio empírico para estudiar la gestión de errores sn SPL.

Analizar el rendimiento de los ingenieros en una SPL cuando corrigen errores;

Con la finalidad de completar el conocimiento empírico sobre el citado tema;

Con respecto a la capa donde se ha detectado el error y la necesidad de propagación (o no) de la solución al error;

Desde el punto de vista de ingenieros de software;

En el contexto de un SPL en la división de placas de inducción de la compañía BSH.

En relación al objetivo descrito anteriormente, nuestro estudio busca responder a las siguientes preguntas de investigación:

RQ1 ¿Cuál es la eficacia al corregir errores y propagar la solución en una familia de productos en una SPL?

RQ2 ¿Cuál es la eficiencia al corregir errores y propagar la solución en una familia de productos en una SPL?

RQ3 ¿Cuál es la satisfacción al corregir errores y propagar la solución en una familia de productos en una SPL?

Para encontrar la respuesta a las anteriores preguntas de investigación, utilizamos un estudio de caso. Las variables respuesta de nuestra investigación son eficacia, eficiencia y satisfacción. El factor es el tipo de tarea necesaria para corregir errores en una SPL. El citado factor tiene cuatro niveles, estos cuatro niveles son:

- Corrección de errores en la capa de configuración (CL).
- Corrección de errores en la capa de configuración con propagación de la solución a otros productos (CL-P).
- Corrección de errores en la capa de realización (RL).
- Corrección de errores en la capa de realización con propagación a otros productos (RL-P).

Cada uno de esos niveles se materializa en una tarea mostrada en la Tabla 8.1.

ID	Nombre	Descripción
CL	Solución de errores en la capa de configuración	Un error ha sido detectado in la placa de inducción IH011 con el módulo MOD006. El citado módulo debe ser sustituido por el módulo MOD014. El módulo MOD006 no debe ser sustituido en otras placas de inducción.
CL-P	Solución de errores en la capa de configuración con propagación a otros productos	Un error ha sido detectado en la placa de inducción IH017, en la citada placa el inversor INV014958 utilizado en el módulo MOD016 no funciona correctamente. El inversor debe ser reemplazado por el inversor INV015231. Esta solución debe ser propagada a todas las placas de inducción que contengan el inversor sustituido en la citada configuración.
RL	Solución de errores en la capa de realización	Un error ha sido detectado en el valor del parámetro VMAX en el módulo MOD020 de la placa de inducción IH002. El nuevo valor del parámetro debe ser modificaco a 39. Esta solución no debe ser propagada a otras placas de inducción.
RL-P	Solución de errores en la capa de realización con propagación a otros productos	Un error ha sido detectado en el valor del parámetro VMAX que utiliza el módulo MOD019 de la placa de inducción IH051. El valor del parámetro VMAX debe modificarse a 44. La solución adoptada debe afectar a cada placa de inducción que contiene en su configuración el módulo mencionado anteriormente.

Tabla 8.1: Tareas para la solución de errores.

8.4.2 Métricas

Las variables de respuesta se miden en función del rendimiento mostrado por los ingenieros de software solucionando errores en un conjunto de tareas en una SPL. Las citadas tareas se diferencian por el tipo de error a corregir en cada una de ellas.

La eficacia se define como el porcentaje de tarea del estudio realizado correctamente por el ingeniero sin recibir asistencia durante la realización de la misma. Para definir los porcentajes de tarea, la misma se descompone en un conjunto de subtareas siguiendo el método *Keystroke-Level Model*. Por ejemplo, si la mitad de estas subtareas dentro de una tarea se realizan sin recibir aistencia; la eficacia, en este caso, es del 50 %.

La eficiencia es el cociente entre la eficacia al realizar una tarea y el tiempo invertido en el desarrollo de la citada tarea de acuerdo con el *Common Industry Format* (CIF) para la realización de informes de pruebas de usabilidad [144].

Finalmente, la satisfacción se mide mediante un cuestionario de satisfacción completado por los ingenieros después de terminar las tareas desarrolladas para medir el rendimiento de los ingenieros solucionando errores. El cuestionario está compuesto por diez preguntas con una escala *Likert* con valores comprendidos entre 1 y 5.

8.4.3 Instrumentos

Para poder recopilar datos sobre la corrección de errores en una SPL y su posterior tratamiento se utilizaron los siguientes instrumentos:

Cuestionario demográfico. El cuestionario demográfico tiene el ojetivo de conocer las características de los sujetos que participan en el estudio empírico, incluye preguntas para identificar el perfil de cada sujeto participante. La información que se solicita a través del cuestionario es: el nivel educativo, el tiempo que llevan trabajando en su actual departamento (medido en años), edad, género, tiempo de trabajo diario en contacto con el software de las placas de inducción y el conocimiento atesorado sobre entornos de desarrollo software y herramientas de modelado software.

Hoja de tareas. A los sujetos se les proporciona una hoja de tareas para la corrección de errores en una SPL (cada hoja de tareas contiene una tarea de cada tipo acorde a los diferentes tipos descritos en la sección 8.3). La realización de este tipo de tareas permite la evaluación de como los sujetos realizan

la corrección de errores de en una SPL de placas de inducción utilizando tiempos de ejecución, porcentaje de tareas completadas y tiempo empleado para desarrollar la tarea por parte de los sujetos hasta que requirieron asistencia. Esta información permite el cálculo de la eficiencia y la eficacia.

Productos configurados. Como resultado de la realización de las tareas citadas anteriormente se han generado una serie de productos que nos permiten comprobar si las tareas han sido realizadas de forma correcta.

Grabaciones de vídeo. Dentro del estudio empírico, y contando con el consentimiento de los sujetos, se realizaron grabaciones del desempeño de los sujetos realizando tareas de corrección de errores y de las respuestas de los mismos a las preguntas realizadas tras la realización de las tareas y haber completado el cuestionario de satisfacción. Estas grabaciones nos permitieron calcular los valores de la eficacia y eficiencia y, por otro lado, conocer las opiniones de los sujetos sobre el proceso de corrección de errores en una SPL.

Cuestionario de satisfacción. El cuestionario de satisfacción utilizado fue *System Usability Scale* (SUS), permite medir de una forma subjetiva la satisfacción de los sujetos, en nuestro caso, en el proceso de corrección de errores en una SPL. El cuestionario está compuesto por diez preguntas con una escala de *Likert*. En el cuestionario SUS original, la palabra "system" fue reemplazada en nuestro caso por "herramienta SPL ". El cuestionario SUS, con solo diez preguntas, arroja resultados confiables para medir la satisfacción de un usuario con un proceso o herramienta [142]. El cuestionario SUS aborda diferentes aspectos de la reacción del usuario en el proceso de reparación de errores de la herramienta SPL desde un punto de vista de la totalidad (por ejemplo, "Encontré la herramienta SPL innecesariamente compleja ", "Me sintí muy seguro al usar la herramienta SPL") en lugar de preguntar al usuario por características muy específicas del sistema (por ejemplo, apariencia visual, organización de la información, etc.).

Entrevista. Al finalizar el proceso se desarrolla una entrevista semiabierta. Este instrumento nos sirve para determinar la comprensión de los participantes en el proceso de corrección de errores del SPL, y para detectar aquellas facetas de la corrección de errores en una SPL que es más problemático, entre otros, desde el punto de vista de interacción persona-computador junto con las causas reales de los problemas acaecidos [143]. Por ejemplo, una de las pregunta es "¿En qué fases de la tarea has encontrado una mayor dificultad?".

8.4.4 Participantes

Los sujetos en el estudio empírico fueron cinco ingenieros de desarrollo software de la división de placas de inducción de BSH. Estos ingenieros son expertos en el desarrollo de software para placas de inducción. Acorde a sus respuestas en el cuestionario demográfico han estado entre 1,5 y 12 años trabajando en el departamento de las placas de inducción (con una media de 6,7 años). A su vez, respondieron en el cuestionario que, habitualmente, pasan entre 1 y 8 horas diarias trabajando con el software de las placas de inducción (una media de 5 horas por día). Además, el 60 % de los sujetos afirmaron que eran competentes en el manejo de entornos de desarrollo de software integrados (como por ejemplo Eclipse).

Además de los sujetos descritos en el párrafo anterior, también participaron en el estudio un instructor y un observador. El instructor proporcionó la información necesaria sobre el manejo de la herramienta SPL, impartió un tutorial sobre la herramienta antes de realizar las tareas por parte de los sujetos, aclaró las posibles dudas acontecidas durante el desarrollo del estudio empírico y entrevistó a los sujetos cuando éstos finalizaron las tareas y completaron los cuestionarios. Por otro lado, el observador tomó notas durante todo el proceso, grabó las entrevistas y realizó el análisis posterior.

8.4.5 Procedimiento

En esta subsección se describe el procedimiento utilizado para desarrollar el estudio de caso. El procedimiento descrito a continuación se repitió con cada uno de los sujetos:

1. A los sujetos se les explicó los objetivos y el procedimiento a desarrollar durante el estudio. Además, se les informó que todo el proceso (realización de tareas, completado de formularios y entrevista) iba a ser grabado para su posterior tratamiento.

2. Los sujetos recibieron un pequeño tutorial sobre la herramienta SPL de BSH. Este tutorial fue explicado por el instructor.

3. Se solicitó a los sujetos que rellenaran el cuestionario demográfico.

4. Los sujetos recibieron instrucciones sobre el desempeño en la realización de tareas (*Performance measurament*). Se les aconsejó que intentaran realizar las tareas sin solicitar ayuda, y que, en la medida de lo posible,

solo solicitasen ayuda si se sentían incapaces de completar la tarea de forma autónoma.

5. Se solicitó a los sujetos que completaran cuatro tareas que aglutinan, por combinación de las mismas, el tipo de tareas de corrección de errores en una herramienta SPL identificadas al final de la sección 8.3. La Tabla 8.1 muestra la lista de tareas, su identificación, nombre y descripción. Estas tareas se usaron para calcular la eficacia y la eficiencia. Para evitar un posible efecto techo (*ceiling effect*), no se estableció tiempo límite para completar las tareas.

6. El observador anotó como fue el desempeño de los sujetos al realizar tareas de corrección de errores. Esta información se complementa con una grabaciones de vídeo de la sesión para su posterior análisis.

7. Posteriormente, se solicitó a los sujetos que rellenaran un cuestionario SUS. Este cuestionario se utilizó para calcular la satisfacción de los ingenieros durante el proceso de corrección de errores con la herramienta SPL de BSH.

8. En el siguiente paso, los sujetos fueron entrevistados por el instructor sobre el proceso de corrección de errores en las cuatro diferentes tareas.

9. Finalmente, el observador revisó las grabaciones de los ingenieros realizando las tareas para calcular la eficacia y efeciencia; se calculó la satisfacción en función de las respuestas obtenidas en el cuestionario SUS, se transcribió la entrevista de los usarios en la finalización del experimento y se analizaron los resultados obtenidos.

8.5 Resultados

En esta sección se describen los resultados obtenidos para las variables de respuesta. Los resultados se muestran en las Figuras 8.3 y 8.4.

8.5.1 Eficacia

La Figura 8.3 (a) muestra que la mayoría de las tareas fueron completadas correctamente por los ingenieros. Fijándonos en los números concretos, 15 de 20 tareas (cuatro tareas por usuario y cinco usuarios) se completaron y realizaron correctamente.

En términos de asistencias recibidas, un sujeto (Sujeto 3) requirió asistencia en la tarea CL y otro sujeto (Sujeto 2) requirió asistencia en la tarea RL-P. En el resto de las tareas no fue requerida asistencia por parte de los sujetos. Se debe añadir que la tarea CL-P fue terminada correctamente por dos sujetos, el resto de los sujetos no propagaron de forma correcta los cambios requeridos para solucionar el error detectado. Por otro lado, la eficacia media de la tarea CL-P fue del 87,4 %, mientras que en el resto de las tareas se obtuvieron valores para la media de la eficacia superiores al 90 %. Finalmente, la tarea RL fue realizada correctamente por todos los sujetos.

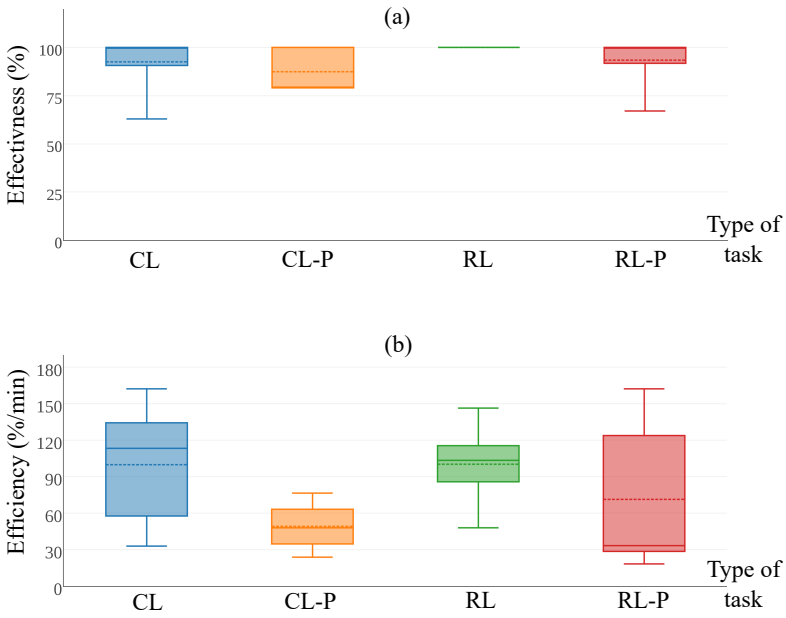


Figura 8.3: Resultados obtenidos para eficacia y eficiencia en el proceso de corrección de errores.

De acuerdo con los resultados descritos, podemos responder RQ1 afirmando que la eficacia para corregir errores y para propagar las soluciones en una familia de productos en una SPL obtiene un buen valor (cercano al 100 %).

8.5.2 Eficiencia

La Figura 8.3 (b) muestra que las tareas que involucraron solo un producto (CL y RL) se realizaron con mayor eficiencia que las tareas que implicaban propagación de la solución (CL-P y RL-P). También es destacable que cuando la propagación no era necesaria, el valor medio alcanzado para la eficiencia es muy similar en el proceso de solución de errores en ambas tareas, en el caso de la tarea correspondiente a la capa de configuración el valor fue de 99.83 %/min y el valor para las tareas en la capa de realización fue de 100.28 %/ min.

Cuando se requiere propagación para corregir el error, el valor medio de eficiencia es mejor en la tarea desarrollada en la capa de realización (71,39 %/min) que en la tarea desarrollada en la capa de configuración (49,1 %/min). Por otro lado, la desviación estándar es mayor que 35 %/min en todas las tareas excepto en la tarea CL-P (19.96 %/min). Finalmente, considerando exclusivamente las tareas con eficacia igual al 100 %, el valor medio alcanzado para la eficiencia en la tarea CL-P fue de 48,52 %/min, mientras que el valor de la media para la eficiencia en las otras tareas fue superior a 84,6 %/min.

De acuerdo con estos resultados, podemos responder RQ2 afirmando que la eficiencia para corregir errores y propagar la solución en una familia de productos en un SPL es cercana al 100 %/min cuando la eficacia es baja.

Questions	Type	Normalized Results
1. I think that I would like to use this SPL tool frequently.	P	65%
2. I found the SPL tool unnecessarily complex.	N	90%
3. I thought the SPL tool was easy to use.	P	75%
4. I think that I would need the support of a technical person to be able to use this SPL tool.	N	70%
5. I found the various functions in this SPL tool were well integrated.	P	90%
6. I thought there was too much inconsistency in this SPL tool.	N	75%
7. I would imagine that most people would learn to use this SPL tool very quickly.	P	75%
8. I found the SPL tool very cumbersome to use.	N	85%
9. I felt very confident using the SPL tool.	P	50%
10. I needed to learn a lot of things before I could get going with this SPL tool.	N	75%
	Total:	75%

SUS results. *P = Positively worded item, N = Negatively worded item*

Figura 8.4: Resultados para la satisfacción en el proceso de corrección de errores.

8.5.3 Satisfacción

La Figura 8.4 muestra la percepción de la satisfacción por parte de los sujetos en la utilización de la herramienta SPL, el valor medio alcanzado con el cuestionario SUS fue de 75 %. Este valor, de acuerdo con la investigación de Bangor en [146] para mejorar la interpretación de los valores alcanzados en el cuestionario SUS, califica la herramienta SPL como " buena ". Además, los ítems 3 y 5 se relacionan con la complejidad y la funcionalidad respectivamente alcanzaron el valor más alto (90 %). Por otro lado, el ítem 9 relacionado con la confianza que la utilización de la herramienta genera obtuvo el valor más bajo (50 %).

Tal y como cita J. Brooke en [145], a partir de los trabajos de Lewis [151] donde concluye que SUS estaba destinado a mostrar el valor de la satisfacción percibida de un sistema, el análisis del cuestionario indica que incorpora dos escalas. Una escala mide la capacidad de aprendizaje de un sistema, mientras que el el resto de la escala mide la usabilidad general del sistema. En el trabajo de Borsci [152] se confirmó de manera independiente la estructura de dos factores del SUS, lo que también muestra que esos factores (usabilidad y capacidad de aprendizaje) están correlacionados. La Figura 8.5 muestra los valores obtenidos para la capacidad de aprendizaje (valor medio de 72.5 %) y para la usabilidad (valor medio de 75.62 %).

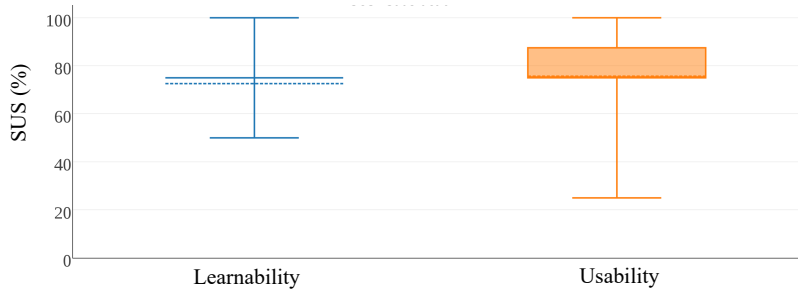


Figura 8.5: Resultados obtenidos para los factores capacidad de aprendizaje y usabilidad.

De acuerdo con estos resultados, podemos responder RQ3 afirmando que la satisfacción de corregir errores y propagar la solución en una familia de productos en una SPL obtiene un buen valor.

8.6 Discusión sobre los resultados obtenidos

Esta sección analiza en detalle los resultados a través de las entrevistas que el instructor hizo al final de las tareas y los datos recopilados en el desempeño de los sujetos para corregir errores y propagar las soluciones a un catálogo de productos en una SPL.

A los sujetos se les preguntó sobre las causas que les llevaron a requerir asistencia en determinadas ocasiones para finalizar las tareas y por las dificultades encontradas durante la realización de las tareas de corrección de errores, y finalmente, sobre sus respuestas en el cuestionario de satisfacción. Esta información nos permitió ahondar sobre las verdaderas causas sobre el rendimiento de los sujetos en una SPL en el ámbito de la generación de software para la generación de placas de inducción cuando corrigen los errores. Sus respuestas nos permiten mejorar el análisis de las preguntas de investigación de la siguiente manera:

RQ1 ¿Cuál es la eficacia para corregir errores y propagar la solución en una familia de productos en una SPL?

En la capa de configuración (CL), los sujetos tenían que corregir un error en un producto (una placa de inducción) al reemplazar una de sus características (*feature*) por una diferente. Mientras que cuatro sujetos corrigieron con éxito el error, un sujeto requirió la asistencia del instructor porque no recordaba los pasos necesarios en la herramienta SPL para reemplazar una determinada característica (*feature*) en un producto configurado.

En la capa de realización (RL), los sujetos tuvieron que corregir un error en un producto (una placa de inducción) al cambiar el valor de un parámetro que se establece en una característica (*feature*) seleccionada de su configuración. Todos los sujetos corrigieron con éxito el error sin necesitar ayuda por parte del instructor. Esto puede deberse al hecho de que los sujetos realizan desarrollan principalmente, en su trabajo diario, trabajos referidos a la capa de realización.

En la tarea de propagación en la capa de configuración (CL-P), tres sujetos no completaron la tarea con éxito porque propagaron la solución al error para un producto a otros productos que no requerían la citada solución. Al revisar los vídeos, detectamos que la dificultad surgía cuando los errores a corregir dependían de la interacción entre varias características (*features*). Específicamente, el error ocurrió en la tarea CL-P debido a la configuración simultánea de dos características (por ejemplo, la característica A) con otra (por ejemplo, la característica B). La estrategia aplicada por todos los sujetos para corregir el

error fue modificar una de las dos características y luego propagar la solución. Dos de los cinco sujetos solo propagaron la solución a los productos que tenían la configuración simultánea (característica A y característica B).

En la tarea de propagación en la capa de realización (RL-P), un sujeto no completó la tarea satisfactoriamente. El sujeto solucionó el error en una placa de inducción, y posteriormente abandonó la realización de la tarea porque no pudo propagar la solución en una familia de productos. En la entrevista posterior, el sujeto afirmó que para él era muy importante un mejor (y más claro) *feedback* cuando se crea una nueva característica como consecuencia inmediata al solucionar un error.

RQ2 ¿Cuál es la eficacia para corregir errores y propagar la solución en una familia de productos en una SPL?

En cada tarea, los sujetos dedicaron tiempo a probar si los cambios para corregir el error se habían realizado correctamente y tenían las consecuencias deseadas. Por ejemplo, el sujeto 4 declaró: *"No estoy seguro de si la propagación se realizó correctamente y necesito verificar si los cambios han sido materializados de forma correcta"*. Por otro lado, observando el rendimiento el sujeto 5, que logró los mejores valores de eficacia, se deduce que la causa fue el conocimiento detallado sobre las características de todos los productos configurados en el catálogo de la herramienta SPL de la empresa.

En la tarea de la capa de configuración (CL), el sujeto 3 enfatizó la importancia de cómo se nombran las diferentes características. El citado sujeto afirmó en la entrevista: *"Gasté mucho tiempo durante la realización de las tareas porque los nombres de los inversores son algo diferentes entre los utilizados por la herramienta SPL y los que utilizo en mi trabajo diariamente"*.

En la tarea de la capa de realización (RL), todos los sujetos corrigieron con éxito el error. Detectamos que cuatro de los cinco sujetos crearon clones de características (*features*) involuntariamente, es decir, en el proceso de corrección del error crean características (*features*) que ya existen en el catálogo de elementos de la SPL. Esto se debió a que los sujetos crearon fragmentos del modelo en la capa de realización para corregir el error que corresponde a las características que ya existen. La estrategia de corrección de errores aplicada por todos los sujetos en las tareas RL fue copiar y modificar la característica que se debía modificar y luego seleccionaron esta característica "corregida" para el producto en el que se detectó el error. En la entrevista, tres de los sujetos declararon que copiar una característica era una tarea engorrosa y propensa a errores. De hecho, el tiempo obtenido con respecto a la eficiencia confirma

que la creación manual de una característica es el paso que lleva más tiempo. Para evitar esta dificultad, los sujetos sugirieron que la herramienta SPL debería proporcionar automatización para copiar y modificar características y controlar la posible creación de clones innecesarios.

RQ3 ¿Cuál es la satisfacción de corregir errores y propagar la solución en la familia de productos en una SPL?

Todos los sujetos declararon que la herramienta SPL era *" una herramienta útil "*. Sin embargo, cuatro sujetos coincidieron en sus opiniones que la herramienta SPL debería informar cuando una característica no es utilizada por ningún producto después de la corrección de errores, es decir, una característica que deja de ser utilizada en la SPL pero que queda almacenada en la herramienta. Dado que la aparición de una característica no utilizada en el SPL es importante, sería enormemente interesante que los ingenieros puedan decidir, en estas situaciones, si la característica no utilizada se elimina, guarda o si su estado cambia para denotar esta nueva situación de la característica.

El ítem 9 en el cuestionario de satisfacción muestra que los sujetos no están muy seguros con la herramienta SPL. Dos de los cinco sujetos afirmaron que si cuando realizan modificaciones en una característica el nombre de la característica modificada, no se cambiaba en la capa de configuración después de la corrección de errores, podría ser una fuente de dificultades para otros usuarios de la SPL, ya que no van a detectar que la característica ha sido modificado en la capa de realización. Además, el sujeto 4 expresó su temor sobre una correcta propagación de los cambios y dudaba sobre una característica fundamental de la herramienta: el código generado por la herramienta SPL una vez configurados los productos.

8.7 Amenazas a la validez

En este apartado se hace una breve descripción de las ameazas a la validez del estudio de caso presentado en este capítulo, para ello utilizamos la clasificación de amenazas a la validez propuestas en [116]:

Validez de construcción: este tipo de validez refleja la medida en que las medidas operativas representan lo que los investigadores tienen en mente. Esta amenaza fue tratada usando protocolos bien establecidos en otros trabajos de investigación [138]. Los instrumentos utilizados en nuestra investigación son ampliamente aceptados en la comunidad de investigación del ámbito interacción persona-computador. Por otro lado, nuestro estudio no se centra en la

funcionalidad de herramientas específicas, nuestras tareas son genéricas dentro del ámbito de una SPL y para corregir errores en una SPL.

Validez interna: este tipo de validez se refiere a cuando el factor que se investiga puede ser afectado por otros factores no tenidos en consideración. Para mitigar esta amenaza, no tuvimos ninguna influencia sobre la elección de los sujetos por parte de por nuestro socio industrial BSH. Aunque el número de sujetos puede parecer relativamente pequeño, no permitiendo realizar un completo estudio estadístico, en la investigación de interacción persona-computador se aconseja usar cinco sujetos en la prueba de usabilidad para detectar el 80 % de los problemas de usabilidad [140].

Validez externa: este tipo de validez se refiere hasta qué punto es posible generalizar los hallazgos del estudio. La generalización estadística no es posible a partir de nuestro estudio de caso debido al bajo número de sujetos paerticipantes en el estudio empírico. Para poder realizar una generalización de nuestros resultados las tareas de solución de errores deberían ser evaluadas en otras herramientas de modelado y en diferentes contextos industriales. Las tareas de estudio utilizadas en el trabajo mostrado en este capítulo son típicas del desarrollo de una SPL. Por esta razón, estos resultados podrían ser relevantes para otros modeladores y otros desarrolladores de herramientas SPL. Dado que la herramienta utilizada en este estudio es una herramienta concreta de nuestro socio industrial, la generalización de los hallazgos debe realizarse con precaución.

Fiabilidad: este tipo de amenaza tiene en consideración en qué medida en que los datos y el análisis realizado dependen de lo específico investigadores. Para minimizar esta amenaza, los valores de las medidas estudiadas y las respuestas de los sujetos se han analizado utilizando las grabaciones realizadas durante el desarrollo del estudio. Para la medición de la satisfacción, usamos el cuestionario SUS, que es un cuestionario confiable según [142].

8.8 Conclusiones

Hasta ahora, los estudios empíricos sobre Líneas de Producto Software se han centrado en analizar técnicas de *testing* para buscar errores y el grado de fiabilidad, pero no hay datos empíricos sobre la corrección de errores en SPLs. En este capítulo se ha documentado un estudio de caso para evaluar la corrección de errores en el contexto de una SPL industrial del grupo BSH en su división de placas de inducción (BSH es el mayor fabricante de electrodomésticos en

Europa). El número de sujetos en este estudio de caso es pequeño, por lo que la generalización de los hallazgos debe realizarse con precaución.

Medimos la eficacia, eficiencia y satisfacción para determinar si hay dificultades en la corrección de errores en el contexto de una SPL en un entorno industrial. Los resultados muestran que estas tres variables obtienen buenos valores en los procesos de corrección de errores y propagación de la solución en una familia de productos en una SPL. La eficacia obtiene valores cercanos al 100 %, la eficiencia obtiene valores cercanos al 100 % cuando la eficacia es baja y la satisfacción es cercana al valor máximo posible.

El estudio de caso revela algunas dificultades para realizar la corrección de errores en la capa de configuración, en la capa de realización y en la propagación. Estas dificultades son relevantes para el estudio y utilización de SPLs y abordarlas con éxito contribuiría a promover la adopción de SPL en la industria.

Parte III

Conclusión

9

Conclusión y Trabajos Futuros

Índice

9.1 Resumen del capítulo

Este capítulo muestra los resultados presentados a lo largo de la tesis y concluye la misma. En primer lugar, se conectan las preguntas de investigación particulares presentadas en cada uno de los capítulos de la parte II (capítulos 5, 6, 7 y 8) con las cuatro preguntas de investigación propuestas en el inicio de la tesis. Posteriormente, se hace una recapitulación de la influencia de los resultados de esta tesis en el desarrollo de herramientas de generación de software acorde a MDD+SPL. En el siguiente apartado, se muestran los trabajos publicados y congresos internacionales donde han sido publicados y expuestos los trabajos que componen esta tesis. A continuación, se describe la investigación en curso y las ideas para trabajos futuros. Finalmente, se concluye la tesis.

9.2 Preguntas de investigación

Las cuatro preguntas de investigación presentadas como parte de la tesis en el capítulo 1 apartado 1.2 se han abordado a través de los trabajos de investigación presentados en los capítulos 5, 6, 7 y 8. A continuación, presentamos como se han conectado cada una de las preguntas de investigación con los trabajos de investigación presentados en esta tesis:

RQ1 ¿Cómo afecta la naturaleza de requisitos para generar modelos software en un entorno de desarrollo MDD+SPL?

- **Pregunta de investigación 1 en (ESEM´17)** [25]: ¿Existen diferencias cuando se usan diferentes niveles de enriquecimiento de requisitos con respecto a la eficacia en la creación de modelos de software?
- **Pregunta de investigación 2 en (ESEM´17)** [25]: ¿Existen diferencias cuando se usan diferentes niveles de enriquecimiento de requisitos con respecto a la eficiencia en la creación de modelos de software?
- **Pregunta de investigación 3 en (ESEM´17)** [25]: ¿La dificultad percibida es diferente cuando los ingenieros de software usan diferentes niveles de enriquecimiento de requisitos para construir modelos de software?

Respuesta a RQ1 : Para construir modelos software en un desarrollo fundamentado en MDD+SPL el enriquecimiento de requisitos con pares propiedad-valor se ha demostrado, mediante un experimento desarrollado

en un entorno industrial, como una mejora. Esta mejora quedó reflejada en las medidas obtenidas de eficacia, eficiencia y dificultad percibida. El experimento mostró que los pares propiedad-valor son utilizados por los ingenieros de software como puntos de control que sirven de ayuda en la generación de modelos software.

RQ2 ¿Cuál es la usabilidad en un desarrollo software basado en MDD+SPL?

Pregunta de investigación en (CSiMQ´15) [27]: ¿Las herramientas de modelado aumentadas con CVL son lo suficientemente intuitivas como para realizar las principales tareas desde la perspectiva del modelado de variabilidad (configuración, ámbito y visualización)?

Respuesta a RQ2 : El estudio empírico desarrollado para evaluar la usabilidad mostró, por los valores alcanzados para las medidas empleadas (eficacia, eficiencia y satisfacción) por los usuarios finales, que los citados usuarios realizaron correctamente las tareas propias en un desarrollo basado en MDD+SPL y reutilizaron de manera sistemática fragmentos de modelo para generar nuevos modelos. Por otro lado, la evaluación de usabilidad realizada desveló algunos problemas de usabilidad que, dependiendo de su naturaleza, deben ser abordados desde distintos puntos de vista: desarrolladores de software basado en modelos, procesos de estandarización de la variabilidad y fabricantes de herramientas de modelado.

RQ3 ¿Cómo es la comprensión para configurar productos desde fragmentos de modelo?

Pregunta de investigación 1 en (CAiSE´16) [29]: ¿Hay dificultades para comprender la variabilidad en los fragmentos de modelo para la configuración de productos?

Pregunta de investigación 2 en (CAiSE´16) [29]: ¿Hay diferencias en la comprensión de fragmentos de modelo aislados, cruzados y recursivos en la configuración de productos?

Respuesta a RQ3 : El experimento para evaluar la comprensión de fragmentos de modelo para la configuración de productos mostró, mediante la medición de comprensión, tiempo empleado para la realización de las configuraciones y dificultad percibida, resultados ya conocidos para los casos de fragmentos aislados y cruzados; por contra, desveló la necesidad de prestar atención al trabajo con fragmentos recursivos. Por otro lado, concluimos que es necesario prestar especial atención a la sintaxis con-

creta de los fragmentos de modelo, incluir la posibilidad de configurar fragmentos a nivel de biblioteca e incluir en los materiales de aprendizaje como abordar configuraciones redundantes.

RQ4 ¿Cómo se corrigen los errores un entorno de desarrollo software basado en MDD+SPL?

Pregunta de investigación 1 en (ESEM´16) [30]: ¿Cuál es la eficacia al corregir errores y propagar la solución en una familia de productos en una SPL?

Pregunta de investigación 2 en (ESEM´16) [30]: ¿Cuál es la eficiencia al corregir errores y propagar la solución en una familia de productos en una SPL?

Pregunta de investigación 3 en (ESEM´16) [30]: ¿Cuál es la satisfacción al corregir errores y propagar la solución en una familia de productos en una SPL?

Respuesta a RQ4 : Para determinar si hay dificultades en la corrección de errores en el contexto de una SPL en un entorno industrial medimos la eficacia, eficiencia y satisfacción. Los resultados mostraron que estas tres variables obtienen buenos valores en los procesos de corrección de errores y propagación de la solución en una familia de productos en una SPL. Por otro lado, el estudio empírico mostró la necesidad de prestar atención a situaciones como cuando algunas *features* no son utilizadas, cuando se crean clones de manera involuntaria o cuando, como consecuencia de solucionar un error, se genera de manera implícita una nueva *feature*.

9.3 Influencia de esta tesis en la industria

En este apartado se muestran (ver Tabla 9.1) los resultados de todos los estudios que conforman esta tesis y que han sido aplicados total o parcialmente en las herramientas desarrolladas por el grupo de investigación SVIT para el desarrollo de software según MDD+SPL.

El consejo con ID 1 está asociado a la dimensión *procesado de requisitos*, los consejos con ID comprendido entre 2 y 9 están asociados a la dimensión *usabilidad*, los consejos con ID comprendido entre 10 y 12 están asociados a la dimensión *comprensión* y, finalmente, los consejos con ID comprendido entre 13 y 18 están asociados a la dimensión *gestión de errores*.

Los resultados obtenidos en un determinado dominio, como es el desarrollo de software para placas de inducción, han sido posteriormente aplicados en otros dominios, como la fabricación de software para la construcción de trenes.

9.4 Publicaciones

Los resultados de investigación mostrados en esta tesis han sido presentados y discutidos en distintos fórums revisados por pares (*peer-review forums*). Las distintas publicaciones en las que el autor ha participado se listan a continuación:

1. Conferencias internacionales

   - Jorge Echeverría, Francisca Pérez, Carlos Cetina, y Óscar Pastor. *Comprehensibility of variability in model fragments for product configuration.* En Conference on Advanced Information Systems Engineering (CAiSE). Ljubljana, Slovenia. 2016.

     Calificación CORE: A. Calificación GII-GRIN-SCIE: II (*very good events*). Ratio de aceptación: 16.5 %.

   - Jorge Echeverría, Francisca Pérez, Andrés Abellanas, Jose Ignacio Panach, Carlos Cetina, y Óscar Pastor. *Evaluating bug-fixing in software product lines: An industrial case study.* En 10th International Symposium on Empirical Software Engineering and Measurement (ESEM). Ciudad Real, España. 2016.

     Calificación CORE: A. Calificación GII-GRIN-SCIE: II (*very good events*). Ratio de aceptación: 29 %.

   - Jorge Echeverría, Francisca Pérez, Carlos Cetina, y Óscar Pastor. *Assessing the performance of automated model extraction rules.* En Information Systems Development: Advances in Methods, Tools and Management (ISD). Larnaca, Chipre. 2017.

     Calificación CORE: A. Este artículo ha sido escogido por invitación, por encontrarse entre los mejores trabajos, para ser publicado por Springer en la serie "Lecture Notes in Information Systems and Organisation".

   - Jorge Echeverría, Francisca Pérez, José Ignacio Panach, Carlos Cetina, y Óscar Pastor. *The influence of requirements in software mo-*

ID	Paper	Consejo	Aplica
1	[25]	Enriquecer los requisitos con pares propiedad-valor que actúen como puntos de control para la creación de modelos software	Parcialmente
2	[27]	Mejora de la sintaxis concreta resaltando los fragmentos de modelo como puntos de variación y su naturaleza	Sí
3	[27]	Acceso desde un fragmento de modelo a todos los modelos que lo utilizan desde la biblioteca	Sí
4	[27]	Soporte para seleccionar el alcance del cambio en un fragmento de modelo	Sí
5	[27]	Generación de informes sobre los cambios producidos tras la realización de tareas complejas (incluyendo creación de fragmentos de modelo implícitamente)	Sí
6	[27]	Creación de fragmentos de modelo de forma explícita	Sí
7	[27]	Soporte para la detección de clones de fragmentos de modelo	Parcialmente
8	[27]	Operación de reemplazo de fragmentos de modelo a nivel de biblioteca	Parcialmente
9	[27]	Soporte para comparar fragmentos de modelo	Parcialmente
10	[29]	Figuras geométricas de puntos de variación son muy importantes: alinear forma geométrica del punto de variación y fragmento	Sí
11	[29]	Permitir combinar fragmentos en la biblioteca	Parcialmente
12	[29]	Los materiales de entrenamiento de modelado de la variabilidad deben mostrar que no es necesario realizar operaciones redundantes	Parcialmente
13	[30]	Clarificar cuando una característica aparece de forma simultánea a otra	Parcialmente
14	[30]	Feedback cuando al solucionar un error se crea una nueva feature	No
15	[30]	Nomenclatura en sintaxis concreta acorde al dominio	Sí
16	[30]	Feedback sobre la propagación de cambios cuando una feature es modificada en todos los productos que utilizan la citada feature	Sí
17	[30]	Automatización para copiar y modificar características y controlar la posible creación de clones innecesarios	Parcialmente
18	[30]	Informar sobre features no utilizadas por ningún producto	Sí

Tabla 9.1: Consejos a aplicar en el desarrollo MDD+SPL.

del development in an industrial environment. En 11th International Symposium on Empirical Software Engineering and Measurement (ESEM). Toronto, Canada. 2017.

Calificación CORE: A. Calificación GII-GRIN-SCIE: II (*very good events*). Ratio de aceptación: 19 %.

2. *Workshops* internacionales de alta relevancia para esta tesis

   - Jorge Echeverria, Jaime Font, Carlos Cetina, and Óscar Pastor. *Usability evaluation of variability modeling by means of common variability language*. CAiSEForum 2015 co-localizada con 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015). Estocolmo, Suecia. 2015.

     Ratio de aceptación: 15.12 % . De los 205 *papers* no seleccionados para la conferencia CAiSE 2015 los 31 *papers* con mayor puntuación fueron invitados a participar en CAiSEForum 2015 [153].

3. Revistas internacionales no indexadas en JCR

   - Jorge Echeverría, Jaime Font, Óscar Pastor, y Carlos Cetina. *Usability evaluation of variability modeling by means of common variability language*. Complex Systems Informatics and Modeling Quarterly (CSIMQ). 2015.

     La participación en CSIMQ 2015 fue por invitación a los mejores *papers* presentados en CAiSEForum 2015.

4. *Doctoral Symposium* internacionales

   - Jorge Echeverría. *Research on augmenting the MDD process with variability modeling*. 14th International Doctoral Symposium on Empirical Software Engineering (IDoESE 16). Publicado en ACM SIGSOFT Software Engineering Notes, Volume 41 Issue 6. Ciudad Real, España. 2016

9.5 Trabajos futuros

Con el objetivo último de mejorar el conocimiento del proceso de desarrollo software MDD+SPL desde el grupo de investigación SVIT se han planificado objetivos a corto y medio plazo. Dentro de los objetivos a corto plazo destaca el diseño, desarrollo y evaluación de resultados de un estudio empírico que culminará con la escritura de un artículo de investigación donde se comparan dos técnicas de desarrollo de software utilizando modelos: *Clone and Own (CaO)* versus MDD+SPL.

Las investigaciones científicas han comprobado que, es una práctica muy extendida en entornos industriales, la generación de software siguiendo el enfoque CaO: esta situación es habitual cuando una empresa tiene que abordar sus necesidades del mercado objetivo lanzando un nuevo producto que es similar, aunque no idéntico, a alguno de sus productos ya existente. En muchos casos, los artefactos de un producto existente se clonan y modifican para ajustarse a los nuevos requisitos [154, 155].

Después de analizar las obras principales relacionadas con el estudio empírico de MDD+SPL y CaO, concluimos algunas afirmaciones importantes que nos indicaron como proceder con el diseño de nuestro estudio. En primer lugar, encontramos una falta de evaluaciones realizadas en la industria. La gran mayoría de los trabajos utilizaron estudiantes o los propios autores como sujetos. No encontramos una comparación de MDD+SPL versus CaO. De acuerdo con los trabajos existentes, esta nueva contribución es un claro paso adelante para evaluar un desarrollo MDD+SPL en un contexto real de uso, cubriendo una brecha de los trabajos previos.

El objetivo de esta investigación en curso es conocer el desempeño de ingenieros de software cuando reutilizan activos (*assets*) en el proceso de desarrollo de productos software utilizando dos enfoques diferentes: MDD+SPL vs CaO. Hemos realizado un experimento de factor único donde la variable independiente es el tipo de enfoque (CaO vs MDD+SPL) para realizar cuatro tareas en el contexto de nuestro socio industrial BSH, en concreto en la división de creación de placas de inducción.

En el contexto del desarrollo de productos de software con activos reutilizables, las tareas se pueden clasificar como:

- Tarea sin nuevo activo: Esto implica el desarrollo de un producto de software sin crear un nuevo activo.

- Tarea con nuevo activo: Esto implica el desarrollo de un producto de software donde se debe crear un nuevo activo.
- Tarea para desarrollar un nuevo producto de software: Como resultado de esta tarea, se desarrolla un nuevo producto de software.
- Tarea para modificar un producto de software existente: Como resultado de esta tarea, se modifica un producto de software existente.

En la realización de este estudio empírico todos los sujetos participantes realizaron las mismas 4 tareas: cada una perteneciente a uno de los tipos descritos anteriormente.

Los resultados estadísticos obtenidos, a falta de una análisis pormenorizado y la correspondiente discusión científica, muestran que el desarrollo siguiendo el enfoque MDD+SPL obtiene mejores resultados que el desarrollo software siguiendo el enfoque CaO.

En un horizonte a medio plazo nuestras investigaciones estarán dirigidas a incrementar el conocimiento de desarrollo software MDD+SPL, con este fin hemos planificado dos objetivos:

1. Otro aspecto que complemente y mejore la investigación realizada para el desarrollo de esta tesis es la confirmación de los resultados obtenidos. Para ello es necesario que los diversos estudios empíricos descritos sean realizados de una forma análoga en contextos distintos a los descritos en esta tesis, es decir, en contextos de generación de software diferentes a la generación de software para placas de inducción en BSH y generación de software para sistemas ferrovarios en CAF.
2. Realizar un estudio empírico que nos permita medir como es aceptado el enfoque MDD+SPL en entornos de desarrollo software industrial. Con este fin diseñaremos un estudio empírico que nos permita estudiar como los usuarios aceptan este paradigma de desarrollo utilizando, por ejemplo, el Modelo de Aceptación de Tecnología (*Technology Acceptance Model, TAM*).

9.6 Conclusiones finales

Con la finalidad de facilitar la reproducción de los experimentos desarrollados para la realización de estas tesis y para facilitar el trabajo de otros investigadores que puedan construir sus trabajos de investigación sobre el nuestro, se han proporcionado enlaces web a nuestro repositorio público. En los enlaces se encuentran los materiales utilizados en los siguientes experimentos:

- Experimento para evaluar la comprensión en la configuración de productos sofware.
  - Url: `https://svit.usj.es/productconfigurationexperiment/`
  - Material disponible: Pre-cuestionario, material empleado en el tutorial, enunciados de los ejercicios, cuestionario sobre dificultad percibida y respuestas de todos los usuarios.
- Experimento para evaluar el procesado de requisitos.
  - Url: `https://svit.usj.es/requerimentinfluenceexperiment/`
  - Material disponible: Cuestionario demográfico, material empleado en el tutorial, hoja de tareas, respuestas de todos los usuarios (incluyendo modelos software).

Estos materiales pueden servir para ser reproducidos, realizar familias de experimentos o para que se utilicen los datos en experimentos con otro enfoque.

Nuestra experiencia en el grupo de investigación SVIT está poniendo de manifiesto que nuevos paradigmas de desarrollo software, como MDD+SPL tratado en esta tesis, se están mostrando de gran utilidad en la industria. Este hecho nos indica que se abren nuevas direcciones a explorar en el ámbito de la Ingeniería del Software. A su vez, nos invita a la realización de estudios que nos permitan recorrer estos caminos acompañados por la industria para mejorar el conocimiento de los mismos y optimizar la generación de software.

Finalmente, me gustaría agradecer al Centro de Investigación PROS y el Grupo SVIT el reto y la oportunidad que me han brindado de realizar mi investigación en entornos industriales.

Bibliografía

[1] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques.* Springer Science & Business Media, 2005.

[2] Paul Clements and John McGregor. Better, faster, cheaper: Pick any three. *Business Horizons*, 55(2):201–208, 2012.

[3] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1—-7:8, New York, NY, USA, 2013. ACM.

[4] Richard A. Krueger and Mary Anne Casey. Designing and conducting focus group interviews. *Social Analysis, Selected Tools and Techniques, Krueger, RA, MA Casey, J. Donner, S. Kirsch and JN Maack*, pages 4–23, 2002.

[5] Guillermo Pantaleo. *Calidad en el Desarrollo de Software.* Alfaomega Grupo Editor, 2013.

[6] R. Rabiser, D. Dhungana, W. Heider, and P. Grunbacher. Flexibility and End-User Support in Model-Based Product Line Tools. In *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference*, pages 508–511, 2009.

[7] Aleksandar Jakšić, Robert B. France, Philippe Collet, and Sudipto Ghosh. Evaluating the Usability of a Visual Feature Modeling Notation. In Benoît Combemale, David J. Pearce, Olivier Barais, and Jurgen J. Vinju, editors, *Software Language Engineering*, pages 122–140, Cham, 2014. Springer International Publishing.

[8] Xiaorui Zhang. *Developing model-driven software product lines.* PhD thesis, Faculty of Mathematics and Natural Sciences, 2014.

[9] Steffen Thiel and Andreas Hein. Modeling and Using Product Line Variability in Automotive Systems. *IEEE Softw.*, 19(4):66–72, July 2002.

[10] Dhirendra Pandey and Vandana Pandey. Requirement Engineering: An Approach to Quality Software Development. *International Journal of*

Global Research in Computer Science (UGC Approved Journal), 3(9):31–33, 2012.

[11] Tom Clancy. The standish group chaos report. *Project Smart*, 2014.

[12] V. Ambriola and V. Gervasi. Processing Natural Language Requirements. In *Proceedings of the 12th International Conference on Automated Software Engineering (Formerly: KBSE)*, ASE '97, pages 36–, Washington, DC, USA, 1997. IEEE Computer Society.

[13] Adrian Fernandez, Emilio Insfran, and Silvia Abrahão. *Modelo de Usabilidad Web Alineado con SQuaRE para Procesos de Desarrollo Dirigido por Modelos*. RA-MA Editorial, 01 2010.

[14] Barry W. Boehm, John R. Brown, and Hans Kaspar. Characteristics of software quality. 1978.

[15] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality.* ISO/IEC, 2001.

[16] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In Ana Moreira, Bernhard Schätz, Jeff Gray, Antonio Vallecillo, and Peter Clarke, editors, *Model-Driven Engineering Languages and Systems*, pages 1–17, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[17] Jim A. McCall, Paul K. Richards, and Gene F. Walters. Factors in software quality. volume i, ii, iii. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA, 1977.

[18] Sonia Montagud. Un Método para la Evaluación de la Calidad de Líneas de Productos Software basado en SQuaRE, 2009.

[19] Carla I. M. Bezerra, Rossana M. C. Andrade, and José Maria S. Monteiro. Measures for Quality Evaluation of Feature Models. In Ina Schaefer and Ioannis Stamelos, editors, *Software Reuse for Dynamic Systems in the Cloud and Beyond*, pages 282–297, Cham, 2014. Springer International Publishing.

[20] Sonia Montagud, Silvia Abrahão, and Emilio Insfran. A systematic review of quality attributes and measures for software product lines. *Software Quality Journal*, 20(3):425–486, Sep 2012.

[21] Adam Trendowicz, Teade Punter, et al. Quality modeling for software product lines. In *7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'03)*, 2003.

[22] Javier González-Huerta, Emilio Insfran, Silvia Abrahão, and John D. McGregor. Non-functional Requirements in Model-driven Software Product Line Engineering. In *Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages*, NFPinDSML '12, pages 6:1–6:6, New York, NY, USA, 2012. ACM.

[23] International Organization for Standardization (ISO). ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

[24] Sonia Montagud and Silvia Abrahão. Gathering Current Knowledge About Quality Evaluation in Software Product Lines. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 91–100, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[25] Jorge Echeverria, Francisca Pérez, José Ignacio Panach, Carlos Cetina, and Oscar Pastor. The Influence of Requirements in Software Model Development in an Industrial Environment. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2017, Toronto, ON, Canada, November 9-10, 2017*, pages 277–286, 2017.

[26] Jorge Echeverria, Francisca Pérez, Carlos Cetina, and Oscar Pastor. Assessing the Performance of Automated Model Extraction Rules. In *Information Systems Development: Advances in Methods, Tools and Management (ISD2017 Proceedings) Larnaca, Cyprus: University of Central Lancashire Cyprus.*

[27] Jorge Echeverria, Jaime Font, Oscar Pastor, and Carlos Cetina. Usability Evaluation of Variability Modeling by means of Common Variability Language. *CSIMQ*, 5:61–81, 2015.

[28] Jorge Echeverria, Jaime Font, Carlos Cetina, and Oscar Pastor. Usability Evaluation of Variability Modeling by means of Common Variability Language. In *Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering co-located with 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015), Stockholm, Sweden, June 10th, 2015.*, pages 105–112, 2015.

[29] Jorge Echeverria, Francisca Pérez, Carlos Cetina, and Oscar Pastor. Comprehensibility of Variability in Model Fragments for Product Configuration. In *CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*, pages 476–490, 2016.

[30] Jorge Echeverría, Francisca Pérez, Andrés Abellanas, Jose Ignacio Panach, Carlos Cetina, and Óscar Pastor. Evaluating Bug-Fixing in Software Product Lines: An Industrial Case Study. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 24:1–24:6, New York, NY, USA, 2016. ACM.

[31] Jorge Echeverria. Research on Augmenting the MDD Process with Variability Modeling. *ACM SIGSOFT Software Engineering Notes*, 41(6):1–6, 2016.

[32] Roelf J. Wieringa. *Design science methodology for information systems and software engineering.* Springer, 2014. 10.1007/978-3-662-43839-8.

[33] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[34] Stuart Kent. *Model Driven Engineering*, pages 286–298. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[35] Francisco Durán Muñoz, Javier Troya Castilla, and Antonio Vallecillo Moreno. Desarrollo de software dirigido por modelos. *FUOC. Fundación para la Universitat Oberta de Catalunya*, 2007.

[36] Carlos Cetina Englada. *Achieving Autonomic Computing through the Use of Variability Models at Run-time.* PhD thesis, Universidad Politécnica de Valencia, 2010.

[37] Óscar Pastor, Marcela Ruiz, and Sergio España. *From Requirements to Code: A Full Model-Driven Development Perspective*, pages 56–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[38] N. Anquetil, B. Grammel, I. Galvao, J.A.R. Noppen, S. Shakil Khan, H. Arboleda, A. Rashid, and A. Garcia. *Traceability for Model Driven, Software Product Line Engineering*, pages 77–86. Number Supplement. SINTEF, 6 2008.

[39] S. Sendall and W. Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, Sept 2003.

[40] Antoni Olivé. *Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[41] Jose Ignacio Panach, Nathalie Aquino, and Óscar Pastor. A proposal for modelling usability in a holistic MDD method. *Science of Computer Programming*, 86(Supplement C):74 – 88, 2014. Special issue on Software Support for User Interface Description Languages (UIDL 2011).

[42] Macario Polo Usaola. Desarrollo de software basado en reutilización. *FUOC. Fundación para la Universitat Oberta de Catalunya*, 2007.

[43] Ó. Díaz and S. Trujillo. Líneas de producto software. *Publicado en "Fábricas de Software: experiencias, tecnologías y organización"2*, pages 61–79, 2010.

[44] D. L. Parnas. On the Design and Development of Program Families. *IEEE Trans. Softw. Eng.*, 2(1):1–9, January 1976.

[45] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[46] Rafael Capilla, Jan Bosch, and Kyo Chul Kang, editors. *Systems and Software Variability Management, Concepts, Tools and Experiences.* Springer, 2013.

[47] Paul C. Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[48] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling Stepwise Refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.

[49] Linda M. Northrop. SEI's Software Product Line Tenets. *IEEE Softw.*, 19(4):32–40, July 2002.

[50] Gary Chastek and John McGregor. Guidelines for Developing a Product Line Production Plan. Technical Report CMU/SEI-2002-TR-006, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2002.

[51] Karina Villela, Adeline Silva, Tassio Vale, and Eduardo Santana de Almeida. A Survey on Software Variability Management Approaches. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 147–156, New York, NY, USA, 2014. ACM.

[52] Samuel Sepúlveda, Cristina Cachero, and Carlos Cares. Modelado de Características para Líneas de Producto de Software: una propuesta. In *International Workshop on Advances Software Engineering (IWASE'12)*, pages 16–20, 2012.

[53] J. D. McGregor, L. M. Northrop, S. Jarrad, and K. Pohl. Initiating software product lines. *IEEE Software*, 19(4):24–27, July 2002.

[54] Matías Pol'la, Agustina Buccella, Alejandra Cechich, and Maximiliano Arias. Un modelo de metadatos para la gestión de la variabilidad en líneas de productos de software. In *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Simposio Argentino de Ingeniería de Software (Buenos Aires, 2014)*, pages 158–172, 2014.

[55] Klaus Schmid and Isabel John. A customizable approach to full lifecycle variability management. *Sci. Comput. Program.*, 53(3):259–284, 2004.

[56] CVL Submission Team. Common variability language (CVL), OMG revised submission. http://www.omgwiki.org/variability/lib/exe/fetch.php?id=start&cache=cache&media=cvl- revised- submission.pdf., 2012.

[57] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. *COVAMOF: A Framework for Modeling Variability in Software Product Families*, pages 197–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[58] Krzysztof Czarnecki and Michał Antkiewicz. *Mapping Features to Models: A Template Approach Based on Superimposed Variants*, pages 422–437. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[59] Faheem Ahmed. Software Requirements Engineer: An Empirical Study about Non-Technical Skills. *JSW*, 7(2):389–397, 2012.

[60] Muneera Bano and Naveed Ikram. *Addressing the Challenges of Alignment of Requirements and Services: A Vision for User-Centered Method*, pages 83–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[61] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of linguistic techniques for Use Case analysis. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 157–164, 2002.

[62] Jorge J. García Flores. *Semantic Filtering of Textual Requirements Descriptions*, pages 427–433. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[63] Leonid Kof. Requirements Analysis: Concept Extraction and Translation of Textual Specifications to Executable Models. In *Proceedings of the 14th International Conference on Applications of Natural Language to Information Systems*, NLDB'09, pages 79–90, Berlin, Heidelberg, 2010. Springer-Verlag.

[64] Sergio España, Nelly Condori-Fernandez, Arturo González, and Óscar Pastor. An empirical comparative evaluation of requirements engineering methods. *Journal of the Brazilian Computer Society*, 16(1):3–19, 2010.

[65] Gonzalo Génova, José M. Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41, 2013.

[66] Irit Hadar, Iris Reinhartz-Berger, Tsvi Kuflik, Anna Perini, Filippo Ricca, and Angelo Susi. Comparing the comprehensibility of requirements models expressed in use case and tropos: Results from a family of experiments. *Information and Software Technology*, 55(10):1823 – 1843, 2013.

[67] Vidhu Bhala R. Vidya Sagar and S. Abirami. Conceptual modeling of natural language functional requirements. *Journal of Systems and Software*, 88:25 – 41, 2014.

[68] A. Ferrari, F. Dell'Orletta, G.O. Spagnolo, and S. Gnesi. Measuring and improving the completeness of natural language requirements. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8396 LNCS:23–38, 2014.

[69] J. Mund, D. Méndez Fernández, H. Femmer, and J. Eckhardt. Does Quality of Requirements Specifications Matter? Combined Results of Two Empirical Studies. volume 2015-November, pages 144–153, 2015.

[70] Souhaib Besrour, Lukman Bin Ab Rahim, and P. Dominic. Exploratory Study to Assess and Evaluate Requirement Specification Techniques Using Analysis Determination Requirements Framework. *Research Journal of Applied Sciences, Engineering and Technology*, 9(3):165–171, 2015.

[71] M. Robeer, G. Lucassen, J. M. E. M. v. d. Werf, F. Dalpiaz, and S. Brinkkemper. Automated Extraction of Conceptual Models from User Stories via NLP. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 196–205, Sept 2016.

[72] Talat Ambreen, Naveed Ikram, Muhammad Usman, and Mahmood Niazi. Empirical research in requirements engineering: trends and opportunities. *Requirements Engineering*, pages 1–33, 2016.

[73] Daniel Amyot, Hanna Farah, and Jean-François Roy. Evaluation of Development Tools for Domain-Specific Modeling Languages. In Reinhard Gotzhein and Rick Reed, editors, *System Analysis and Modeling: Language Profiles*, volume 4320 of *Lecture Notes in Computer Science*, pages 183–197. Springer Berlin Heidelberg, 2006.

[74] Goetz Botterweck, S. Thiel, D. Nestor, S. bin Abid, and C. Cawley. Visual Tool Support for Configuring and Understanding Software Product Lines. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 77–86, Sept 2008.

[75] Florian Heidenreich, Ilie Savga, and Christian Wende. On Controlled Visualisations in Software Product Line Engineering. In Steffen Thiel and Klaus Pohl, editors, *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*, SPLC (2), pages 335–341. Lero Int. Science Centre, University of Limerick, Ireland, 2008.

[76] P. Grunbacher, R. Rabiser, and D. Dhungana. Product Line Tools are Product Lines Too: Lessons Learned from Developing a Tool Suite. In *Automated Software Engineering. 23rd IEEE/ACM International Conference on*, pages 351–354, 2008.

[77] Bariic, A. and Amaral, V. and Goulao, M. Usability evaluation of domain-specific languages. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference on the*, pages 342–347, Sept 2012.

[78] Rick Rabiser, Paul Grunbacher, and Martin Lehofer. A Qualitative Study on User Guidance Capabilities in Product Configuration Tools. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 110–119, New York, NY, USA, 2012. ACM.

[79] Alexandr Murashkin, Michael Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. Visualization and Exploration of Optimal Variants in Product Line Engineering. In *Proceedings of the 17th Software Product Line Conference*, SPLC '13, pages 111–115. ACM, 2013.

[80] Iris Reinhartz-Berger, Kathrin Figl, and Øystein Haugen. Comprehending Feature Models Expressed in CVL. In *Model-Driven Engineering Languages and Systems*, volume 8767 of *Lecture Notes in Computer Science*, pages 501–517. Springer International Publishing, 2014.

[81] Iris Reinhartz-Berger and Kathrin Figl. Comprehensibility of Orthogonal Variability Modeling Languages: The Cases of CVL and OVM. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 42–51, New York, NY, USA, 2014. ACM.

[82] Jon Oldevik, Øystein Haugen, and Birger Møller-Pedersen. Confluence in Domain-Independent Product Line Transformations. In Marsha Chechik and Martin Wirsing, editors, *FASE*, volume 5503 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2009.

[83] D.L. Moody, P. Heymans, and R. Matulevicius. Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of i* Visual Syntax. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 171–180, Aug 2009.

[84] Iris Reinhartz-Berger and Arava Tsoury. Experimenting with the Comprehension of Feature-Oriented and UML-Based Core Assets. In *Enterprise, Business-Process and Information Systems Modeling*, volume 81 of *Lecture Notes in Business Information Processing*, pages 468–482. Springer Berlin Heidelberg, 2011.

[85] Ko-Hsun Huang, Nuno Jardim Nunes, Leonel Nobrega, Larry Constantine, and Monchu Chen. *Hammering Models: Designing Usable Modeling Tools*, pages 537–554. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[86] Andreas Svendsen, Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. Towards Evolution of Generic Variability Models. In Jörg Kienzle, editor, *Models in Software Engineering*, volume 7167 of *Lecture Notes in Computer Science*, pages 53–67. Springer Berlin Heidelberg, 2012.

[87] Wilco Engelsman and Roel Wieringa. Understandability of Goal-Oriented Requirements Engineering Concepts for Enterprise Architects.

In Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors, *Advanced Information Systems Engineering*, volume 8484 of *Lecture Notes in Computer Science*, pages 105–119. Springer International Publishing, 2014.

[88] Anatoly Vasilevskiy and Øystein Haugen. Resolution of Interfering Product Fragments in Software Product Line Engineering. In *Model-Driven Engineering Languages and Systems*, volume 8767 of *Lecture Notes in Computer Science*, pages 467–483. Springer International Publishing, 2014.

[89] Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wąsowski. Three Cases of Feature-Based Variability Modeling in Industry. In *ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2014.

[90] P. Caire, N. Genon, P. Heymans, and D.L. Moody. Visual notation design 2.0: Towards user comprehensible requirements engineering notations. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 115–124, July 2013.

[91] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Sarah Nadi, and Rohit Gheyi. The Love/Hate Relationship with the C Preprocessor: An Interview Study (Artifact). *DARTS*, 1(1):07:1–07:32, 2015.

[92] Alexander Egyed. Fixing Inconsistencies in UML Design Models. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 292–301, Washington, DC, USA, 2007. IEEE Computer Society.

[93] Bo Wang, Yingfei Xiong, Zhenjiang Hu, Haiyan Zhao, Wei Zhang, and Hong Mei. A Dynamic-Priority Based Approach to Fixing Inconsistent Feature Models. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *Model Driven Engineering Languages and Systems*, pages 181–195, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[94] E. Uzuncaova, S. Khurshid, and D. Batory. Incremental Test Generation for Software Product Lines. *IEEE Transactions on Software Engineering*, 36(3):309–322, May 2010.

[95] Mahdi Noorian, Alireza Ensan, Ebrahim Bagheri, Harold Boley, and Yevgen Biletskiy. Feature Model Debugging based on Description Logic Reasoning. In *DMS*, pages 158–164. Knowledge Systems Institute, 2011.

[96] Sandeep Krishnan, Robyn R. Lutz, and Katerina Goševa-Popstojanova. Empirical Evaluation of Reliability Improvement in an Evolving Software Product Line. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 103–112, New York, USA, 2011. ACM.

[97] Michaela Steffens, Sebastian Oster, Malte Lochau, and Thomas Fogdal. Industrial Evaluation of Pairwise SPL Testing with MoSo-PoLiTe. In *Proceedings of the Sixth International Workshop on VaMoS*, VaMoS '12, pages 55–62, New York, NY, USA, 2012. ACM.

[98] M. Ohira, A. E. Hassan, N. Osawa, and K. i. Matsumoto. The impact of bug management patterns on bug fixing: A case study of Eclipse projects. In *28th IEEE International Conference on Software Maintenance*, pages 264–273, Sept 2012.

[99] Lei Ma, Cyrille Artho, Cheng Zhang, and Hiroyuki Sato. Efficient Testing of Software Product Lines via Centralization . *SIGPLAN Not.*, 50(3):49–52, September 2014.

[100] L. Rincón, G. Giraldo, R. Mazo, C. Salinesi, and D. Diaz. Method to Identify Corrections of Defects on Product Line Models. *Electronic Notes in Theoretical Computer Science*, 314:61 – 81, 2015. CLEI 2014, the XL Latin American Conference in Informatic.

[101] Jean Melo, Claus Brabrand, and Andrzej Wąsowski. How Does the Degree of Variability Affect Bug Finding? In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 679–690, New York, NY, USA, 2016. ACM.

[102] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. Towards Predicting Feature Defects in Software Product Lines. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*, FOSD 2016, pages 58–62, New York, NY, USA, 2016. ACM.

[103] Megha, Arun Negi, and Karamjit Kaur. Method to Resolve Software Product Line Errors. In Saroj Kaushik, Daya Gupta, Latika Kharb, and Deepak Chahal, editors, *Information, Communication and Computing Technology*, pages 258–268, Singapore, 2017. Springer Singapore.

[104] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[105] Oscar Pastor. *Model-Driven Development in Practice: From Requirements to Code*, pages 405–410. Springer International Publishing, Cham, 2017.

[106] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering. *IEEE Transactions on Evolutionary Computation*, PP(99):1–1, 2017.

[107] Jean-Claude Royer and Hugh Arboleda. *Model-Driven and Software Product Line Engineering (ISTE)*. Wiley-IEEE Press, 1st edition, 2012.

[108] Hugo Arboleda, Rubby Casallas, and Jean claude Royer. Implementing an MDA Approach for Managing Variability. In *in Product Line Construction Using the GMF and GME Frameworks. 5th Nordic Workshop on Model Driven Software Engineering. August*, pages 67–82, 2007.

[109] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau, and Krzysztof Pietroszek. Model-driven Software Product Lines. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 126–127, New York, NY, USA, 2005. ACM.

[110] Kelly Garcés, Carlos Andres Parra, Hugo Arboleda, Andrés Yie, and Rubby Casallas. Administración de Variabilidad en una Línea de Producto de Software basada en Modelos. *RASI*, 4:3–12, 2007.

[111] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London, London, 2008.

[112] Robert K. Yin. *Applications of Case Study Research Second Edition (Applied Social Research Methods Series Volume 34)*. Sage Publications, Inc, December 2002.

[113] Robert Davison, Maris G. Martinsons, and Ned Kock. Principles of canonical action research. *Information Systems Journal*, 14(1):65–86, 2004.

[114] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. *Journal of Systems and Software*, 91:3 – 23, 2014.

[115] Leonor Buendía, Ma Colás Bravo, Hernández Pina Pilar, et al. *Métodos de investigación en psicopedagogía*. McGraw-Hill, 1998.

[116] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[117] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[118] Barry W. Boehm. *Software Engineering Economics*, pages 99–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[119] David Garlan. Software Architecture: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 91–101, New York, NY, USA, 2000. ACM.

[120] James J. Cappel. Entry-Level is Job Skills: A Survey of Employers. *Journal of Computer Information Systems*, 42(2):76–82, 2002.

[121] Dante Carrizo, Oscar Dieste, and Natalia Juristo. Systematizing Requirements Elicitation Technique Selection. *Information and Software Technology*, 56(6):644–669, June 2014.

[122] Svetan M. Ratchev, Kulwant S. Pawar, Esmond Urwin, and Dirk Mueller. Knowledge-enriched Requirement Specification for One-of-a-kind Complex Systems. *Concurrent Engineering: R&A*, 13(3):171–183, 2005.

[123] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[124] Javier Gonzalez-Huerta, Emilio Insfran, Silvia Abrahão, and Giuseppe Scanniello. Validating a model-driven software architecture evaluation and improvement method: A family of experiments. *Information and Software Technology*, 57:405–429, 2015.

[125] Richard A. Krueger and Mary Anne Casey. *Focus groups: A practical guide for applied research*. Sage publications, 2014.

[126] Jose Ignacio Panach, Sergio España, Oscar Dieste, Oscar Pastor, and Natalia Juristo. In Search of Evidence for Model-driven Development Claims. *Inf. Soft. Techn.*, 62(C):164–186, June 2015.

[127] Mohsen Asadi, Samaneh Soltani, Dragan Gašević, and Marek Hatala. The effects of visualization and interaction techniques on feature model configuration. *Empirical Software Engineering*, pages 1–38, 2014.

[128] Brady T. West, Kathleen B. Welch, and Andrzej T Galecki. *Linear Mixed Models: A Practical Guide Using Statistical Software, Second Ed.* Chapman and Hall/CRC Press, 2014.

[129] Jacob Cohen. Statistical power analysis for the behavior science. *Lawrance Eribaum Association*, 1988.

[130] Ø. Haugen. Common Variability Language (CVL)–OMG Revised Submission. OMG document ad. Technical report, 2012-08, 2012.

[131] Sana Ben Nasr, Nicolas Sannier, Mathieu Acher, and Benoit Baudry. Moving Toward Product Line Engineering in a Nuclear Industry Consortium. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 294–303, New York, NY, USA, 2014. ACM.

[132] Dimitri Van Landuyt, Steven Op de beeck, Aram Hovsepyan, Sam Michiels, Wouter Joosen, Sven Meynckens, Gjalt de Jong, Olivier Barais, and Mathieu Acher. Towards Managing Variability in the Safety Design of an Automotive Hall Effect Sensor. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 304–309, New York, NY, USA, 2014. ACM.

[133] Andreas Holzinger. Usability engineering methods for software developers. *Communications of the ACM*, 48(1):71–74, 2005.

[134] Meta Object Facility (MOF) 2.0 Core Specification, 2003.

[135] Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, Gøran K. Olsen, Andreas Svendsen, and Xiaorui Zhang. A Generic Language and Tool for Variability Modeling. Technical report, SINTEF, 2009.

[136] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. Adding Standardized Variability to Domain Specific Languages. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 139–148, Washington, DC, USA, Sept 2008. IEEE Computer Society.

[137] ISO. ISO 9241-11: Ergonomic requeriments for officce work with visual display terminals (VDTs) - Part 11: Guidance on usability. Technical report, International Organization for Standardization, Geneva, Switzerland., 1998.

[138] Nelly Condori-Fernández, José Ignacio Panach, Arthur Iwan Baars, Tanja E. J. Vos, and Oscar Pastor. An empirical approach for evaluating the

usability of model-driven tools. *Sci. Comput. Program.*, 78(11):2245–2258, 2013.

[139] Silvia Abrahão, Emilio Iborra, and Jean Vanderdonckt. Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool. In *Maturing Usability*, pages 3–32. 2008.

[140] Robert A. Virzi. Refining the test phase of usability evaluation: how many subjects is enough? *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 34(4):457–468, 1992.

[141] David Kieras. Using the keystroke-level model to estimate execution times. *University of Michigan*, 2001.

[142] Thomas S. Tullis and Jacqueline N. Stetson. A comparison of questionnaires for assessing website usability. In *Usability Professional Association Conference*, pages 1–12, 2004.

[143] Natalia Juristo and Xavier Ferre. How to Integrate Usability into the Software Development Process. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 1079–1080, New York, NY, USA, 2006. ACM.

[144] ANSI/NCITS. ANSI/NCITS-354 Common Industry Format (CIF) for Usability Test Reports. Technical report, NIST Industry USability Reporting, 2001.

[145] Jhon Brooke. SUS: A Retrospective. *Journal of Usability Studies*, 8(2):29–40, 2013.

[146] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[147] Dominique L. Scapin and J.M. Christian Bastien. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. *Behaviour & information technology*, 16(4-5):220–231, 1997.

[148] Saroj Kumar and Veera Karoli. *Handbook Of Business Research Methods.* Thakur Publishers, 2011.

[149] S. S. Shapiro and M. B. Wilk. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika*, 52(3/4):591–611, 1965.

[150] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Trans. Softw. Eng.*, 28(8):721–734, August 2002.

[151] James R. Lewis and Jeff Sauro. *The Factor Structure of the System Usability Scale*, pages 94–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[152] Simone Borsci, Stefano Federici, and Marco Lauriola. On the dimensionality of the System Usability Scale: a test of alternative measurement models. *Cognitive Processing*, 10(3):193–197, Aug 2009.

[153] Janis Grabis and Kurt Sandkuhl, editors. *Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering co-located with 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015), Stockholm, Sweden, June 10th, 2015*, volume 1367 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[154] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki. An Exploratory Study of Cloning in Industrial Software Product Lines. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 25–34, March 2013.

[155] Thilo Mende, Rainer Koschke, and Felix Beckwermert. An Evaluation of Code Similarity Identification for the Grow-and-prune Model. *J. Softw. Maint. Evol.*, 21(2):143–169, March 2009.

Our experience in the SVIT research group has revealed that new paradigms of software development, such as the combination of Model Driven Development and Software Product Lines, are very useful for the industry. This fact justifies the exploration of new approaches in the Software Engineering field. At the same time, this invites us to join the industry to carry out studies aimed to improve our knowledge and to optimize the automated generation of software from models in industrial domains.