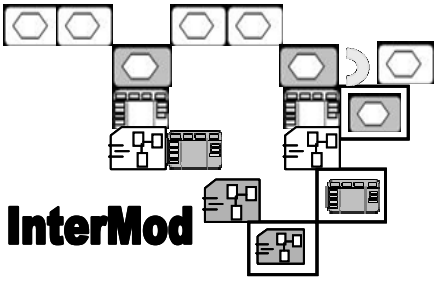
Departamento de Lenguajes y Sistemas Informáticos

Universidad del País Vasco

Tesis doctoral



InterMod: Un enfoque ágil, dirigido por modelos y centrado en el usuario, para desarrollar aplicaciones interactivas

---

Begoña Losada Pereda

2014

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

GALAN

InterMod: Un enfoque ágil, dirigido por modelos y centrado en el usuario, para desarrollar aplicaciones interactivas

---

Memoria de la tesis doctoral desarrollada por Begoña Losada Pereda y dirigida por la doctora Maite Urretavizcaya Loinaz, para optar al grado de doctora en Informática por la Universidad del País Vasco

---

San Sebastián, enero de 2014

A Libo, tan comprensivo, y a Irati y Josu, tan importantes

Agradecimientos

Deseo agradecer especialmente a mi tutora, Maite Urretavizcaya, su dedicación y soporte durante la realización de esta tesis. Han sido muchas las reuniones que hemos mantenido, necesarias para pulir este trabajo. Sin duda, su paciencia y consejo han sido fundamentales para llevarlo a cabo.

También, agradezco al grupo Galán que me acogió y ha facilitado mi trabajo, y de forma especial a Isabel Fernández de Castro y JuanMi López, por sus consejos que me han ayudado mucho. Igualmente, a mis compañeros en la Facultad de Informática que me han animado en este camino; entre ellos quiero citar a Alex García Alonso, Basilio Sierra y Maite Oronoz, y a aquéllos que estuvieron disponibles para responder a mis dudas o darme un consejo como Julio Abascal, Oscar Díez y Ana Molina.

En este camino, desde el inicio sentí el apoyo de las personas que componen la asociación AIPO, donde encontré buenos profesionales pero sobre todo buena gente.

Debo agradecer al director y secretaria del Departamento de Lenguajes y Sistemas Informáticos durante la confección de esta memoria, Xabier Arregi y Montse Hermo, por su buena labor y por conseguir en tiempos difíciles una liberación que me ha permitido terminar esta memoria.

Finalmente quiero agradecer también a Libo por ocuparse de la familia, demasiadas veces en soledad ultimamente, y a mis chicos por su cariño.

Resumen

En el actual contexto tecnológico surge la necesidad de cambiar el enfoque de desarrollo del software interactivo. Se ha demostrado que el enfoque tradicional por fases no se adapta bien a los cambios frecuentes que exige la complejidad semántica de estas aplicaciones, ya que en cada fase trata el proyecto completo e implica al usuario sólo al principio y en la entrega del mismo.

Hace ya algún tiempo que existe la tendencia ágil que enfoca un proyecto a la entrega pronta e iterativa de resultados parciales, dado que el mercado actual es muy competitivo y exige resultados rápidos. La integración continua y los ciclos cortos permiten aceptar los cambios de forma temprana. Sin embargo, los factores de fracaso en el uso de estos enfoques son varios. Entre ellos, destacan los problemas organizativos, la falta de mecanismos para la planificación y/o el seguimiento del proyecto, y la indefinición del papel fundamental del cliente.

La metodología propuesta en esta tesis busca resolver dos importantes problemas. Por una parte, adecuarse a las necesidades del usuario, posiblemente cambiantes, desde el principio al fin del proyecto. Un modelo específico para la recogida y evaluación temprana de los requisitos, y la integración de técnicas de usabilidad, serán cruciales para lograrlo. Por otra parte, formalizar el desarrollo del proyecto para que ayude al proceso. Las reglas de planificación y gestión propuestas en un desarrollo ágil por modelos lograrán superar este reto.

Esta tesis presenta la metodología InterMod para el desarrollo de aplicaciones interactivas que se fundamenta en el desarrollo ágil e integra los principios de la Ingeniería de la Usabilidad y el desarrollo dirigido por modelos.

Contenidos

Índice de Figuras

Índice de Tablas

CAPÍTULO 1. Introducción

La evolución en Ingeniería de Software (IS) ha sido creciente desde sus inicios. Los adelantos en hardware, software y comunicaciones han provocado cambios sociales en el plano personal y cambios organizativos en el ámbito de las empresas de software.

La informática ha dejado de ser un campo limitado a unos pocos profesionales con usuarios poco entendidos e implicados. Los usuarios actuales disponen de hardware, sistemas de red potentes y están habituados a nuevas formas de interacción. Continuamente, surgen propuestas de aplicaciones interactivas cuyo software suele ser complejo en su semántica, con numerosos requisitos que cubren muchas necesidades. Su éxito está condicionado por su utilidad y oportunidad, pero también influyen otros factores como su robustez y usabilidad. En la calidad de un producto software influye tanto el cumplimiento de los requisitos de la aplicación como la cobertura de las necesidades de los usuarios ("ISO 9000:2005 - Quality management systems -- Fundamentals and vocabulary," 2005).

Los ciclos de vida que promueven un desarrollo secuencial estricto que comienza con la especificación completa de los requisitos, continúan con el diseño e implementación y finalizan con las pruebas del sistema, no son adecuados para adaptarse a los cambios frecuentes y las nuevas necesidades que surgen a lo largo de un desarrollo software. Muestra de ello son los proyectos que han fracasado y provocaron la llamada "crisis del software ("The Standish Group International, Inc. The CHAOS Report" 1995), obligando a adaptar las

metodologías a los nuevos tiempos. A raíz de la expansión del desarrollo ágil, encontramos que los autores tienden a clasificar las metodologías de IS en dos categorías:

- Las metodologías clásicas "*heavyweight*" o dirigidas por un plan, que precisan un análisis de requisitos completo "*upfront*" para comenzar el proyecto, además de documentación exhaustiva y planes detallados.
- Las metodologías ágiles o "*lightweight*", que se adaptan a cada situación del proceso en función de los cambios que van surgiendo, y siguen en menor o mayor medida los 12 principios ágiles ("Agile Alliance: The Twelve Principles of Agile Software," 2001).

Mientras que estos dos enfoques llevan a discusión la conveniencia de uno u otro, algunos autores, se decantan por el equilibrio (Boehm and Turner, 2003). Ellos indican que los desarrolladores dirigidos por un plan deben ser ágiles, pero también que los desarrolladores ágiles deben ser disciplinados. La clave del éxito, dicen, es encontrar el equilibrio entre la agilidad y el enfoque clásico, y este equilibrio varía de proyecto a proyecto según las circunstancias y riesgos. En esta misma línea, Olagbegi y Haddad examinan varios casos de estudio y concluyen que los desarrollos ágiles no obtienen resultados más productivos que otros métodos de desarrollo (Olagbegi and Haddad, 2008). Pese a ello, el movimiento hacia las metodologías ágiles, más flexibles y adaptables que las clásicas, y hacia métodos de desarrollo más centrados en el usuario es general y masivo. Leffingwell justifica está tendencia por los beneficios de negocio que proporcionan (Leffingwell, 2011).

El concepto de usabilidad se define como el nivel con el que un producto se adapta a las necesidades del usuario y puede ser utilizado por el mismo para lograr unas metas con efectividad, eficiencia y satisfacción en un contexto específico de uso ("ISO/IEC 25000:2005 - Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE", 2005). Si la Ingeniería de Software tiene por objetivo aplicar mecanismos y procesos de ingeniería al desarrollo de productos software para mejorar su calidad, la Ingeniería de la Usabilidad (IngUS) comprende un conjunto de modelos, técnicas y métodos cuyo objetivo es mejorar la usabilidad de un producto interactivo, y se aplica en la disciplina llamada Interacción Persona Computador-Human Computer Interaction (HCI), cuya visión más extendida es el Diseño Centrado en el Usuario (DCU). La integración de los

principios del DCU en la IS se ha venido debatiendo con intensidad en los últimos años, por ejemplo en el grupo de trabajo de IFIP WG 2.7/13.4 http://www.se-hci.org. Debido a las necesidades especiales de los sistemas interactivos, es preciso incorporar al ciclo de vida las cuestiones de usabilidad. Un software con mala usabilidad puede reducir su productividad y aceptación (Hellmann et al., 2010). Desde este punto de vista, la IngUS afirma que una interfaz de usuario que ha pasado por varias evaluaciones de usabilidad en su desarrollo necesitará menos cambios posteriormente (Cooper et al., 2007) (Hellmann et al., 2010). Esto es, aplicando evaluaciones de usabilidad durante la construcción del sistema se observará y tratará la interacción con los usuarios para determinar la forma de hacerlo más usable (Dix et al., 2003). Las evaluaciones de usabilidad, realizadas por usuarios y/o expertos en usabilidad mediante las técnicas adecuadas, se pueden aplicar desde las primeras etapas del proceso de desarrollo (Mayhew, 1999).

Por otra parte, la utilización de modelos en el diseño de sistemas complejos es habitual en las disciplinas de ingeniería. En IS, los modelos son representaciones abstractas del software, en cuya construcción es crucial la participación de los implicados para el éxito del modelo y la fiabilidad del producto final. El Desarrollo Dirigido por Modelos - Model-Driven Development (DDM) tiene el potencial de incrementar la productividad y calidad del desarrollo. Así, los aspectos importantes de una solución se describen mediante abstracciones, que generan código automáticamente y son más intuitivas que las utilizadas en la codificación de un lenguaje de programación. El primer foco en el desarrollo software mediante DDM son los modelos y no los programas. De esta manera, se expresan los conceptos de una forma más cercana al dominio del problema, lo que hace que los modelos sean más sencillos de especificar, entender y mantener, y menos sensibles a las tecnologías y sus cambios (Selic, 2003). En la actualidad y a la par del auge del desarrollo ágil, se impulsan las prácticas del modelado ágil que incluyen la creación de varios modelos en paralelo, el modelado en incrementos pequeños que se debate y valida en código, y la utilización de herramientas simples para crear los modelos (Ambler, 2002-13a).

1.1 Problemas en el desarrollo ágil de software interactivo de calidad

Las nuevas propuestas metodológicas para el desarrollo de software han intentado paliar problemas que surgían en anteriores métodos. Actualmente, las Metodologías Ágiles (MA), el Diseño Centrado en el Usuario (DCU), y el Desarrollo Dirigido por Modelos (DDM) son las propuestas dominantes y se están dando pasos para lograr la integración de las MA tanto con el DCU (Hussain et al., 2009) (Memmel et al., 2007), como con DDM (Ambler, 2004) y con ambas (Luna et al., 2010). Los problemas principales que se deben solucionar para abordar esta integración se han agrupado aquí desde la perspectiva del proceso ágil de IS, de la IngUS y del DDM.

1.1.1 Ingeniería de Software como proceso ágil

Para introducir "agilidad" en los procesos de la IS se deben crear métodos que permitan tratar las siguientes cuestiones:

1. Necesidades reales de los usuarios. Primero, es preciso conocer y evaluar las peticiones o deseos de los usuarios y, después, interactuar continuamente con ellos, a lo largo del proceso, de modo que se permitan cambios y revisiones en cualquier momento.
2. Proyectos con elementos novedosos y factores desconocidos. Suele ser habitual que un proyecto no se pueda abordar completamente desde un principio debido al desconocimiento de todos los requisitos o a las dependencias entre ellos que se puedan observar en las ejecuciones de los productos parciales.
3. Definición de la unidad de progreso en el desarrollo incremental, ya que en estos procesos es dificil determinar el tamaño de un incremento.
4. Contextos variables, con cambios frecuentes en los requisitos y en la tecnología, como los sistemas web. En cualquier punto del proceso puede haber cambios que impliquen una pequeña variación o una revisión completa.
5. Puntos de evaluación en el proceso y tipo de evaluación a seguir. Una metodología ágil realiza numerosas evaluaciones que condicionan el desarrollo del proyecto, pero surgen

dudas respecto a cuándo realizar las evaluaciones, cómo y qué evaluar. Igualmente, no siempre está claro con quién evaluar.

6. Equilibrio entre la informalidad de los procesos ágiles y la calidad exigible en IS. La calidad de un producto software exige una forma de actuación correcta que debe regirse por unas normas, cuyo cumplimiento ágil no es sencillo.

1.1.2 Integración de los Métodos Ágiles con la Ingeniería de la Usabilidad

Algunos autores destacan algunas dificultades en la integración de las MA con la IngUS, que deben ser consideradas. Entre ellas:

1. Inconsistencias provocadas por cambios en la Interfaz de Usuario (IU). Los cambios en la arquitectura de software normalmente no tienen impacto en *lo que el usuario ve*, en la IU. Sin embargo, los cambios continuos en la IU debido a un diseño iterativo rápido, pueden ocasionar conflictos con las expectativas y comprensión del usuario. Esto causa inconsistencias y finalmente insatisfacción (Constantine, 2002).
2. Costo de las evaluaciones relativo a los estudios de usabilidad. Según Sy, debe haber cambios en la granularidad de los estudios de usabilidad (Sy, 2007). Los ciclos en las iteraciones de corta duración permiten completar diseños pequeños pero los tests de usabilidad se multiplican. Un proceso de evaluación de usabilidad completo tradicional al comienzo de cada iteración puede ser demasiado largo. Además, reclutar a los participantes y evaluadores, diseñar las pruebas y analizar los resultados de evaluación puede ser excesivamente largo y pesado en cada iteración. Es necesario racionalizar los estudios de usabilidad a lo largo del proceso.
3. Equilibrio de diseño y documentación a lo largo del proyecto. Las metodologías ágiles prescriben un tiempo mínimo de diseño y documentación al principio del proyecto, realizándose la parte necesaria del diseño en cada iteración. Sin embargo, esto causa problemas al emplear técnicas de usabilidad tradicionales, en las que el diseño y evaluación de toda la interfaz debería estar finalizado antes de cualquier implementación. Las iteraciones del Diseño Centrado en el Usuario (DCU) pueden llevar mucho tiempo

hasta llegar a una solución. Tal y como dice Memmel: "*A one-to-one integration of UCD processes and methods is in general inappropriate for an agile course of action*" (Memmel et al., 2007). Es preciso encontrar un equilibrio entre diseño y documentación, y "agilidad" en el proceso.

4. Definición piezas de diseño que se ajusten a las necesidades de los usuarios. La evaluación de pequeñas entregas con los implicados no asegura que el sistema total proporcione un modelo conceptual, navegacional o de contenido consistente. Sy indica que, previamente, hay que definir objetivos de diseño, mediante estudios de observación del contexto (Sy, 2007).
5. Adaptación a los cambios de forma coherente con los principios del DCU. Tradicionalmente, el trabajo de diseño y la evaluación de usabilidad de un proyecto se ha venido realizando antes de su implementación. La codificación comienza cuando concluye el proceso iterativo de diseño de la IU, esto es, el diseño del prototipo, la evaluación, análisis y rediseño. Si posteriormente se encuentra un fallo de diseño deberá corregirse en la siguiente versión del producto. Esto se contradice con los principios de las metodologías ágiles que permiten los cambios en cualquier momento del desarrollo ("Manifesto for Agile Software Development", 2001). Además el modelo ágil impulsa las iteraciones cortas y el mínimo diseño up-front.
6. Entregables funcionales y no funcionales. El proceso iterativo en HCI produce normalmente resultados en forma de diseños no funcionales y esto puede estar en disonancia con los principios ágiles. Sin embargo, Larman indica que el resultado de una iteración ágil no es necesariamente un software funcional (Larman, 2003).

1.1.3 Integración de Métodos Ágiles con el Desarrollo Dirigido por Modelos

La integración de las MA con el DDM tiene una serie de dificultades e incompatibilidades, principalmente ligadas con el reto ágil de entrega rápida de producto ejecutable (Luna et al., 2009). Los aspectos problemáticos más destacados son:

1. Modelizaciones redundantes. El enfoque DDM establece múltiples representaciones de los artefactos de un proceso de desarrollo de software para representar las diferentes vistas o niveles de abstracción del mismo concepto. Sin embargo, las MA valoran más la funcionalidad del producto a entregar que la documentación exhaustiva derivada de los modelos, por lo que critican el esfuerzo necesario en la realización de las diferentes perspectivas.
2. Consistencia ágil del modelado. En DDM se debe asegurar la consistencia entre los modelos de un proyecto, a pesar de los cambios que pueda sufrir a lo largo de su proceso. Esto es, cualquier cambio en un modelo (vista), debería reflejarse en los modelos (vistas) interrelacionados. Sin embargo, inferir la semántica de las abstracciones de alto nivel a partir de las de bajo nivel(por ejemplo el código), es mucho más difícil que generar abstracciones de bajo nivel a partir de las de alto nivel, y esto puede ser incompatible con la entrega rápida de un producto ejecutable.
3. Complejidad de los modelos. En algunos casos, los modelos pueden aumentar la complejidad del proceso de desarrollo en lugar de reducirla. Por ello, los modelos con mucho detalle serán difíciles de realizar y mantener (Bell, 2004) (Ambler, 2004). Cada tipo de modelo requiere un conjunto particular de habilidades para crearlo y manterlo, así como equipos expertos que entiendan las relaciones entre los modelos y el impacto de los cambios en ellos. Además, los desarrolladores precisan conocer las notaciones y tecnologías de transformación, y éstas pueden ser complejas. Por ejemplo, UML 2.0 tiene detractores pese a su potencial y uso masivo. Primero, con el objetivo de atender a todas las necesidades se ha vuelto enorme (Thomas and Barry, 2003). Segundo, algunas construcciones en UML 2.0 no tienen una semántica precisa, como por ejemplo los casos de uso. Esto complica la automatización de las herramientas de transformación.

1.2 Metodología propuesta en esta tesis

Esta tesis propone la metodología InterMod para el desarrollo de aplicaciones interactivas en el ámbito de las Metodologías Ágiles y el Desarrollo Dirigido por Modelos e incluye características y técnicas de la Ingeniería de la Usabilidad, para obtener productos software de calidad dirigidos por los deseos de los usuarios.

La manera en que se realiza la integración de la Ingeniería de Software con el Diseño Centrado en el Usuario y el Desarrollo Dirigido por Modelos produce las aportaciones de esta tesis. La metodología propuesta resuelve los tres objetivos siguientes:

1. Realizar un proceso de desarrollo de software ágil y robusto, que se adapte a los cambios y necesidades que puedan surgir en el mismo. Para conseguirlo, habrá que considerar tres aspectos cruciales: la realización y evaluación parcial, la revisión de los desarrollos y, la integración y refactorización continua. Un conjunto de reglas regirá el proceso ayudando a su planificación y asegurando su calidad. Se usarán diagramas visuales y herramientas software como mecanismos facilitadores.

2. Formalizar el desarrollo mediante modelos que describan los requisitos de usuario, la interfaz de usuario y la funcionalidad. Además, estos modelos serán "ágiles", primando la utilidad sobre la formalidad. Los modelos de un desarrollo se alimentarán entre sí, según propone el desarrollo dirigido por modelos. Así, se podrá generar un prototipo concreto a partir de la modelización abstracta de requisitos, independiente de plataforma y lenguaje de programación.

3. Considerar la usabilidad como una característica fundamental de la calidad del producto resultante. La evaluación de la interfaz con el usuario se tendrá en cuenta desde el comienzo del desarrollo y antes de su implementación. Además y desde el principio, el foco principal que guiará el proceso de desarrollo lo formarán las necesidades del usuario final. A lo largo del proceso las evaluaciones de estas necesidades serán puntos primordiales que ayudarán a dirigir el proceso.

1.2.1 Aportaciones para afrontar los problemas de integración.

En InterMod se ha propuesto un modo de integración ágil, coherente y justificado, de los principios de las Metodologías Ágiles con los del Diseño Centrado en el Usuario y el Desarrollo Dirigido por Modelos.

En esta propuesta, los problemas de integración se abordan de la siguiente forma:

1. La noción, especificación, formalización y clasificación del concepto *Objetivo de Usuario(UO),* permite organizar el proyecto ajustándolo al usuario final. Un UO cubre una necesidad de usuario y es la unidad de medida del progreso del proyecto. Por otra parte, por estar ligado desde su concepción al usuario final, su desarrollo incluirá evaluaciones que buscan cumplir sus necesidades reales.
2. El proceso ágil se realiza mediante tres tipos de modelos (M-1, M-2, M-3), que desarrollan los UOs según una nueva perspectiva del DCU que permite asegurar la calidad del resultado final. Además, un conjunto de reglas ayuda al equipo gestor en la toma de decisiones para planificar y desarrollar en paralelo los diferentes modelos. Contar con varios equipos de desarrollo permite acelerar la obtención de resultados, según proponen los modelos ágiles.
3. El modelo específico *SE-HCI* - Semantically Enriched Human-Computer Interaction (M-1) describe la interacción hombre-máquina, semánticamente enriquecida con aspectos de presentación y comportamiento. Este modelo, cercano a los propuestos en HCI, favorece la captación de los requisitos mediante descripciones de las interacciones usuario-sistema que producen efectos en la interfaz. Además, incluye la información necesaria para producir los primeros prototipos que ayudarán a evaluar el producto de manera temprana, y que servirá de base para construir el resto de los modelos. Por otra parte, el modelo propuesto es intuitivo y rápido de confeccionar; a ello colabora la propuesta de herramientas sencillas para su realización.
4. La propuesta de integración de técnicas de evaluación de la IngUS indica qué técnicas usar, cuándo y cómo aplicarlas. Esto asegura el cumplimiento de usabilidad de los UOs, y del proyecto en general.

La Figura 1.1 representa la integración de las tres disciplinas (MA, DCU, DDM) en InterMod, y los puntos de conexión comentados: los UOs, los tres modelos de desarrollo de un UO, el modelo SE-HCI, y el desarrollo y evaluación planteado según la nueva perspectiva DCU.

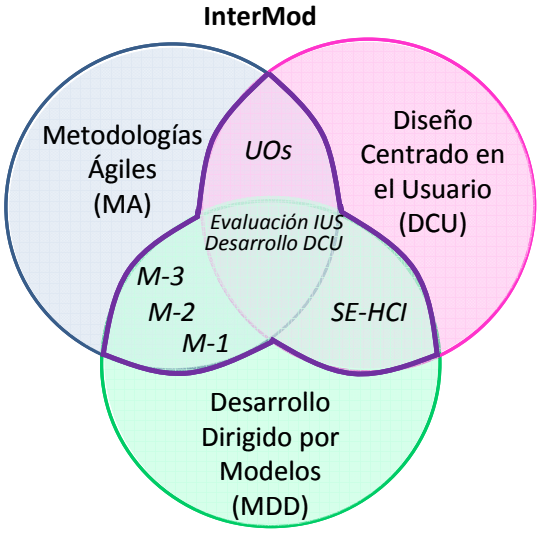


Figura 1.1 InterMod, una propuesta integradora de MA, DCU y DDM

1.2.2 Visión general del proceso metodológico

El proceso de desarrollo del proyecto (Figura 1.2) comienza con la captura, mediante técnicas de la IngUS, de las primeras necesidades o deseos de usuario, referentes al Sistema objetivo, ordenados por prioridad. El tiempo destinado al diseño y documentación al principio del proyecto se ciñe al desarrollo de los UOs iniciales, no a todo el proyecto. Además en este estadio se determina también la visión general del sistema y los aspectos globales del diseño que permitirán un desarrollo posterior, iterativo e incremental, con garantías de coherencia.

Cada iteración va formalizando los Objetivos de Usuario; para ello, los desarrolladores realizan las actividades de desarrollo e integración que crean y evalúan los modelos de desarrollo. En resumen, al inicio de cada iteración, se formalizan las nuevas necesidades de usuario en UOs. A continuación, el conjunto de reglas de planificación y gestión ayuda a tomar decisiones de desarrollo según una nueva perspectiva DCU que establece un orden concreto en la realización de los modelos. Esto es, se decide qué UOs abordar, qué actividad de desarrollo (DAs) o integración (IAs) realizar sobre los UOs escogidos para crear y evaluar los modelos adecuadamente, y cómo distribuir las actividades entre los equipos de trabajo.

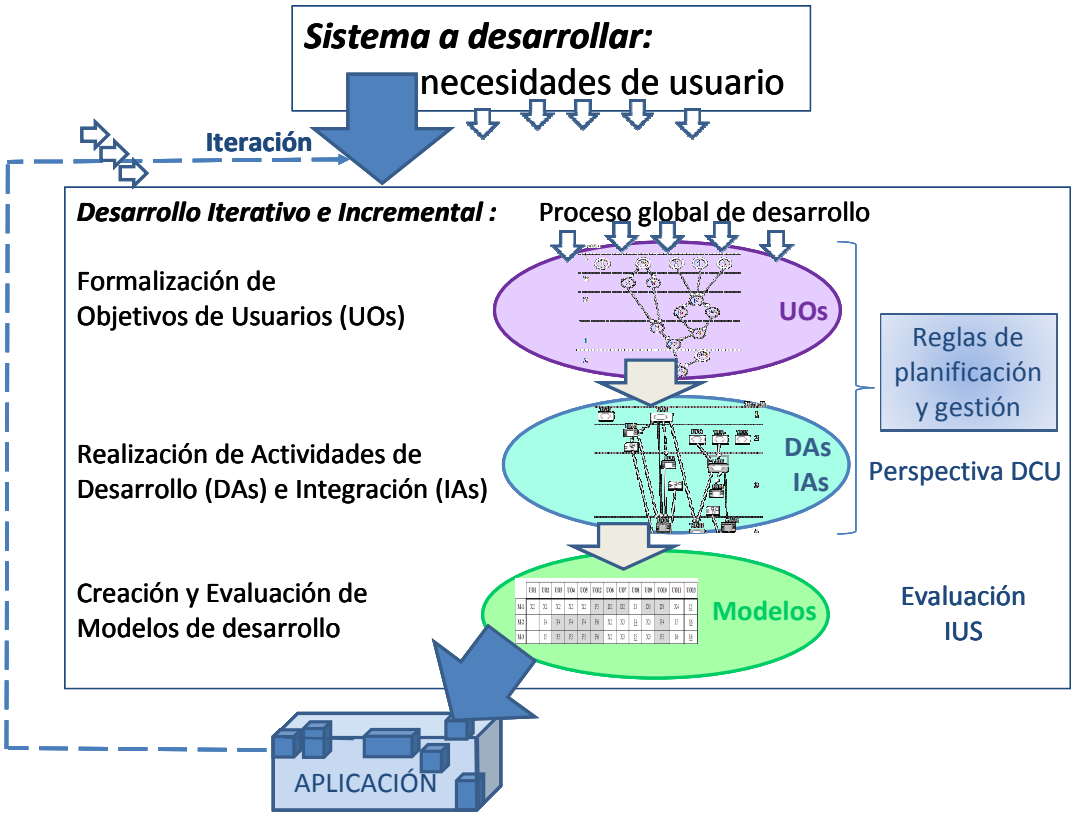


Figura 1.2 Esquema simplificado del proceso InterMod

El proceso de desarrollo, guiado por UOs, exige realizar integraciones y evaluaciones continuas de los modelos. De esta forma, los cambios en la IU serán validados por el usuario en el modelo integrado. Las evaluaciones de los modelos pueden ser internas, esto es con el equipo de desarrolladores, o bien con usuarios cuando se refieran a una petición directa de ellos. Los entregables propuestos en las primeras etapas de desarrollo de un UO son prototipos que sirven para evaluar los requisitos y, parcialmente, la presentación y la funcionalidad. La integración de técnicas de evaluación de IngUS en el proceso asegura la calidad en usabilidad de los UOs y del proyecto en general.

La gestión de modelos y la planificación del proceso buscan aprovechar las ventajas de la modelización sin repercutir negativamente en el desarrollo ágil. De esta manera, el proyecto (y los requisitos) progresa poco a poco, haciendo frente a lo desconocido en las etapas en que es posible su realización. En el enfoque de desarrollo basado en la nueva perspectiva DCU propuesta, la evaluación de los requisitos tiene prioridad sobre la evaluación de la interfaz y ésta tiene prioridad sobre la evaluación de la lógica de negocio. Sin embargo, esto es cierto

únicamente en el contexto de un UO de tal modo que, probablemente, no toda la IU estará desarrollada antes de comenzar la implementación. Las unidades de progreso del proyecto son los UOs, y éstos aumentan y se desarrollan poco a poco a lo largo del proceso a través de sus modelos.

1.3 Estructura de la tesis

La memoria de esta tesis tiene 6 capítulos. Tras la introducción, se revisa la evolución de los aspectos generales que corresponden a cada uno de los pilares que sostienen este trabajo: la Ingeniería de Software interactivo, la Ingeniería de la Usabilidad y el Desarrollo Dirigido por Modelos. A continuación, se describe con detalle la propuesta integradora para el desarrollo de aplicaciones interactivas. Al final del documento, se incluyen tres apéndices con información complementaria,

Esta es la estructura y contenido de esta memoria, en los siguiente capítulos:

- **Capítulo 2 Evolución en el proceso de desarrollo de software interactivo**: Este capítulo muestra la evolución de los ciclos de vida en el desarrollo de software interactivo, comenzando con los métodos tradicionales y haciendo hincapié en los modelos ágiles. Se mencionan las ventajas y desventajas de cada propuesta, y se indican sus áreas óptimas de aplicación. A continuación, se describen los modelos propuestos desde la Ingeniería de la Usabilidad y se resumen los intentos de su integración con la Ingeniería de software. Este capítulo de revisión se completa con la descripción de los diferentes enfoques y herramientas del Desarrollo Dirigido por Modelos, los entornos de desarrollo de interfaces de usuario dirigidos por modelos, y sus ventajas e inconvenientes.
- **Capítulo 3 InterMod, una metodología ágil para el desarrollo de aplicaciones interactivas:** Este capítulo presenta InterMod, desde la perspectiva del desarrollo ágil de aplicaciones interactivas. Se describen los conceptos de la nueva metodología y el modelo de proceso ágil propuesto. Además, incluye el conjunto de reglas que ayudan a la planificación del proceso y se muestra la documentación del progreso del proyecto mediante diagramas de creación de UOs y actividades.

- **Capítulo 4 Desarrollo y gestión de modelos propuestos por InterMod:** Este capítulo describe InterMod desde la perspectiva de los modelos involucrados en un proyecto (M-1, M-2 y M-3) y las técnicas de evaluación propuestas para cada uno de ellos. Se hace especial hincapié en el modelo de requisitos específico (el modelo SE-HCI) de la propuesta aquí presentada. Además, se incluyen las reglas necesarias para la gestión de modelos y se muestra la documentación del progreso de un proyecto mediante la creación y evaluación positiva de sus modelos.
- **Capítulo 5 Integración de la Ingeniería de la Usabilidad en InterMod. Desarrollo paso a paso de una aplicación para móvil**: Este capítulo trata la integración de la ingeniería de la usabilidad en InterMod con el caso de estudio de una aplicación para dispositivo móvil. Primero, se describe el desarrollo de la aplicación paso a paso, y después, se detalla cómo se ha realizado el proceso de integración.
- **Capítulo 6 Conclusion y líneas futuras de trabajo**: Finalmente, se presentan las conclusiones del enfoque presentado. Además, este capítulo muestra el trabajo en desarrollo. Por un lado se muestra, como trabajo más avanzado, la integración de la técnica *Test-Driven Development* en InterMod, y a medio plazo se está trabajando en la utilización de UOs y sus modelos como patrones de diseño. También, se está trabajando en áreas paralelas como en el desarrollo de herramientas que ayuden a la gestión de un proyecto InterMod, y en la aplicación de InterMod en el desarrollo de aplicaciones de e-learning.
- **Apéndice A**: Este apéndice describe las herramientas desarrolladas para para facilitar la confección de algunos modelos propuestos en la metodología InterMod, y muestra su evolución.
- **Apéndice B**: Este apéndice explica el lenguaje de marcado UIML utilizado en las herramientas desarrolladas.
- **Apéndice C**:. Este apéndice muestra la aplicación de InterMod con TDD para el desarrollo de WebDiagram 3.0. Incluye la documentación y el análisis de los resultados en cantidad de tests y tiempos de desarrollo y evaluación.

1.4 Resumen

Este capítulo comienza con la motivación de una propuesta metodológica que integre Metodologías Ágiles (MA), Desarrollo Dirigido por Modelos (DDM) y Diseño Centrado en el Usuario (DCU). Los aspectos que más se destacan en esta motivación son: la tendencia actual hacia las metodologías ágiles, más flexibles y adaptables que las clásicas, el incremento de la productividad y calidad del producto software mediante la filosofía del Desarrollo Dirigido por Modelos, y la consideración de la usabilidad como factor de calidad de software.

A continuación, se citan los problemas de la integración desde las prespectivas del proceso ágil, de la IngUS y del DCU. Entre ellos, destaca la indefinición de unidad de progreso, evaluación y modelado, el tratamiento de los cambios en contextos variables y en resumen, el equilibrio entre la formalidad que exige el DCU y el DDM, y los principios ágiles.

Después se indican brevemente los tres objetivos de la metodología InterMod, propuesta en esta tesis: 1. Realizar un proceso de desarrollo de software ágil y robusto, que se adapte a los cambios y necesidades que puedan surgir en el mismo. 2. Formalizar el desarrollo mediante modelos que describan los requisitos de usuario, la interfaz de usuario y la funcionalidad, y 3. Considerar la usabilidad como una característica fundamental de la calidad del producto resultante. Además, se describen las aportaciones de InterMod que permiten afrontar los problemas de la integración: el concepto de Objetivo de Usuario como unidad de proceso, evaluación y modelado, el proceso ágil realizado mediante la realización de tres modelos y según una nueva perspectiva DCU, el modelo SE-HCI para la captación de los requisitos y la propuesta de integración de técnicas de evaluación de la IngUS en el proceso.

El siguiente capítulo repasa la evolución de los enfoques de desarrollo de software interactivo, destacando sus virtudes y carencias.

CAPÍTULO 2. Evolución en el Proceso de Desarrollo de Software Interactivo

2.1 Introducción

La Ingeniería de Software, tiene por objetivo aplicar disciplinas de ingeniería al desarrollo de productos software para mejorar su calidad. El concepto "*ingeniería*" proporciona un enfoque sistemático, disciplinado y cuantificable en el desarrollo, operación y mantenimiento de sistemas software ("IEEE SA - 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology", 1990).

Un *ciclo de vida software* define las fases, pasos, actividades, métodos, herramientas y entregables de un proyecto de desarrollo de software (Maciaszek and Liong, 2004). En las últimas décadas se han hecho numerosos intentos por encontrar el modelo óptimo en el desarrollo de software. La aplicación de las diferentes propuestas ha fracasado con frecuencia, provocando las llamadas "crisis del software" y así, la generación de nuevos enfoques.

Por otra parte, el software interactivo agrupa aquellas aplicaciones en las que el sistema reacciona a partir de la comunicación directa con el usuario final mediante la interfaz de usuario.

Este capítulo proporciona, en primer lugar, una visión de la evolución general de las propuestas de ciclo de vida en el desarrollo de software, desde que esta disciplina surgió en los años 60. A continuación, se centra en los aspectos de Human-Computer Interaction (HCI); comienza con una breve muestra de los enfoques de la Ingeniería de la usabilidad que se aplican a la realización de desarrollos interactivos, y después describe los esfuerzos realizados para intergrarla con la IS. Finalmente, describe el Desarrollo Dirigido por Modelos (DDM).

2.2 Hacia el desarrollo ágil de software interactivo

La construcción de sistemas de software largos y/o complejos, que deben ser construidos por uno o varios equipos de ingenieros, a veces en lugares diferentes, es el panorama habitual hoy en día. Este contexto ha condicionado la evolución de las nuevas metodologías de desarrollo de software (Ghezzi et al., 2002).

Las metodologías clásicas son "*heavyweight*"o "*plan-driven*"; esto es se dirigen por un plan que requiere en primer lugar una definición de los requisitos ("*big up-front design*"), una fuerte documentación y planes detallados de trabajo. Dos ejemplos típicos son los modelos en cascada y el modelo en espiral. Estas propuestas fracasaron dando lugar a nuevos enfoques iterativos e incrementales. Más recientemente, aparecieron las metodologías ágiles (MA) o "*lightweight*" que siguen los 12 principios ágiles descritos en el Manifiesto Ágil ("Manifesto for Agile Software Development," 2001) y que promocionan la recogida de requisitos en cualquier momento del proceso. A continuación se describen estos enfoques con más detalle.

2.2.1 Primeros modelos en Ingeniería de Software

Los primeros modelos que se generalizaron en IS fueron los modelos en cascada y después, sus variaciones. Los más destacables se describen en este apartado.

2.2.1.1 Modelo en cascada

El ciclo de vida clásico de la IS es el *modelo en cascada* en el que las actividades se enlazan secuencialmente y en donde cada fase finaliza completamente antes de que comience la siguiente (Royce W.W., 1970) (Figura 2.1).

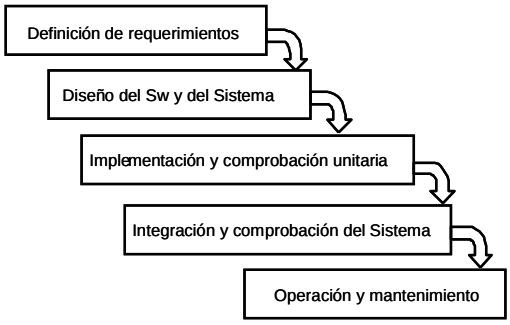


Figura 2.1 Esquema del modelo en cascada original

En detalle, las actividades principales del modelo son:

1. *Definición y análisis de los requisitos*. Se declaran los servicios, limitaciones y objetivos del sistema consultando a los usuarios del sistema.
2. *Diseño del software y del sistema*. Se establece la arquitectura general del sistema. Se identifican y describen las abstracciones fundamentales del sistema software y sus relaciones.
3. *Implementación y comprobación unitaria*. En este paso, se realiza la implementación y testeo de las unidades de programa. El testeo de las unidades implica verificar que cada unidad cumple su especificación.
4. *Integración y comprobación del sistema*. Las unidades de programa individuales se integran y testean como un sistema completo para asegurar que los requisitos de software se cumplen.
5. *Operación y mantenimiento*. El sistema se instala y se pone en uso. El mantenimiento implica: corregir los errores que no se habían percibido en las fases anteriores, mejorar la implementación de las unidades de programa y ampliar los servicios del sistema a medida que surgen nuevos requisitos. Normalmente, en este modelo, esta fase es la más larga.

En el momento en que surge este modelo los lenguajes de programación eran ineficientes, y el hardware consistía en grandes computadores de recursos limitados. En este contexto, los proyectos grandes exigían la inclusión de fases de desarrollo bien definidas y especificaciones de requisitos previas y extensas. Se realizaban contratos con los clientes en los que se exigía que los requisitos no debían cambiar. Esto es, un aspecto crucial para la comunicación de los desarrolladores, era la documentación de requisitos y de las distintas fases (Brooks,Jr., 1995).

Son numerosos los autores que hablan de las ventajas y las desventajas de este modelo. A continuación aparece una recopilación de lo más destacable.

Ventajas del modelo en cascada

- Es simple de describir y entender.
- Es sencillo de aplicar y gestionar. Se realiza la planificación al principio, no debe planificarse a cada paso.(McConnell, 1996)
- La documentación se produce en cada fase y permanece a lo largo del proyecto.
- El proceso es ordenado, con retos guiados por la documentación que se produce en cada etapa, como por ejemplo “Fase de requisitos completa” (Larman, 2003).

Desventajas del modelo en cascada

- Todos los requisitos se recogen al principio, lo que hace muy difícil responder a los cambios de requisitos que proponga el cliente posteriormente (McConnell, 1996). En el modelo en cascada, el coste de los cambios se incrementa con el tiempo (Stone et al., 2005).
- Es necesario disponer de técnicos especialistas para cada fase, y especialmente para la confección e interpretación de los requisitos.
- Es necesario realizar el diseño estructural completamente antes de comenzar la implementación. Bradac encontró que la naturaleza lineal del ciclo de vida clásico llega a “estados de bloqueo” en los que algunos miembros del equipo de proyecto deben esperar a otros a fin de terminar tareas interdependientes (Bradac et al., 1993). McConnell se

pregunta: ¿Porqué hay que retrasar la implementación de las áreas que son fáciles de diseñar hasta que se finalice el diseño de otras áreas complejas? (McConnell, 1996).

- El usuario actúa únicamente en la primera etapa de recogida de requisitos, y en las pruebas finales. Durante el proceso, el usuario es ajeno al desarrollo hasta que el producto ha finalizado y se prueba antes de lanzarlo.
- El proyecto se trata como un todo, no hay posibilidad de dividir la complejidad del sistema (Ghezzi et al., 2002).
- La planificación se realiza al principio y es difícil valorar los riesgos de un proyecto largo al principio del mismo. Por consiguiente, es difícil obtener una planificación correcta y una valoración ajustada (Ghezzi et al., 2002). La dificultad de planificar por adelantado el proceso de desarrollo de un sistema completo es debido al hecho de que muchos sistemas, especialmente los sistemas web, cambian sus requisitos frecuentemente y crecen en su funcionalidad y contenidos durante su ciclo de vida.
- La implementación comienza tarde lo cual ocasiona un grado de incertidumbre y riesgo grande (Larman, 2003).
- Este modelo genera pocos signos visibles de progreso hasta el final y esto crea la impresión de desarrollo lento. Los clientes desean ver resultados tangibles que les asegure que sus proyectos se distribuirán a tiempo (McConnell, 1996).

Áreas y situaciones adecuadas para su aplicación

Varios autores indican cuándo es recomendable utilizar el modelo en cascada y cuándo es mejor evitarlo**:**

- MacCormack comenta que este modelo es bueno para entornos en los que los requisitos de usuario y la tecnología necesaria se entienden bien. Sin embargo, indica que su aplicación en entornos más inciertos, como la IS para Internet, es problemática (MacCormack, 2001).
- McConnell señala que el modelo en cascada trabaja especialmente bien si se tiene una plantilla poco experta, porque proporciona una estructura al proyecto que ayuda a minimizar el esfuerzo (McConnell, 1996).

- Pressman incide en su utilidad como modelo de proceso en situaciones en las que los requisitos sean fijos y el trabajo avance en forma lineal hasta el final. Sin embargo, también dice que el modelo en cascada es inapropiado hoy en día, en que el trabajo de software es acelerado y está sujeto a una corriente sin fin de cambios (en las características, funciones y contenido de la información) (Pressman, 2010).
- Jiang y Eberlein indican que este modelo es deseable en el contexto tecnológico en el que surgió y para proyectos largos que requieren contratos previos entre desarrolladores y clientes. Sin embargo, también indican que después de que los contratos se hayan establecido, se supone que los requisitos no van a cambiar (Jiang and Eberlein, 2009).
- Eckstein, dice que estos modelos han tenido éxito en dominios con requisitos estables. En estos dominios, todo puede ser formalizado, y puede establecerse un plan detallado desde el principio. Es más, en estos casos, un proyecto se puede "blindar" a ese plan sin preocuparse de que pueda modificarse. Además de requisitos estables, estos proyectos suelen tener costo y tiempo ilimitado. Se considera más importante cubrir todos los requisitos que entregar un conjunto de ellos a tiempo y dentro del presupuesto (Eckstein, 2004).
- Boehm señala que no funciona bien con determinados proyectos, particularmente con aplicaciones interactivas (Boehm, 1988).
- Larman incide en que este modelo es inapropiado para proyectos complejos. No debería usarse para desarrollar software orientado a objetos, en proyectos complejos o impredecibles, o en aquéllos proyectos en los que los requisitos son desconocidos o están sujetos a cambios (Larman, 2003).
- Sommerville propone que el modelo en cascada sea usado únicamente cuando los requisitos se entiendan bien y difícilmente vayan a cambiar radicalmente durante el desarrollo del sistema. (Sommerville, 2006).

2.2.1.2 Variaciones del modelo en cascada

Stone prueba que en la práctica, las fases de desarrollo se solapan y se proporcionan información unas a otras. Durante el diseño, son identificados problemas en los requisitos;

durante la codificación, se detectan problemas de diseño, etc. (Stone et al., 2005). El proceso de software no es un modelo lineal simple, sino que requiere una secuencia de iteraciones en las actividades de desarrollo.

Evolución de Sommerville

Sommerville muestra una visión evolucionada del modelo en cascada original (Figura 2.) añadiendo iteraciones hacia delante y hacia atrás entre los distintos pasos (Sommerville, 2006).

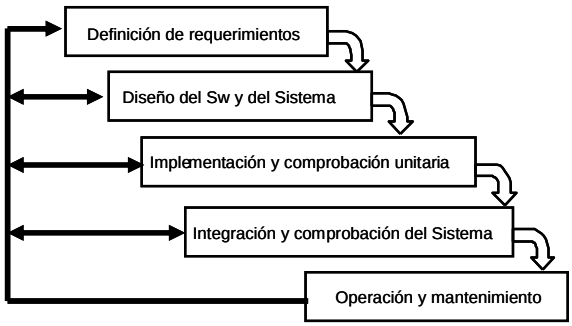


Figura 2.Esquema del modelo en cascada evolucionado

Sin embargo, este mismo autor indica que, debido al coste de producir y aprobar documentos, las iteraciones son costosas, ya que el proceso implica rehacer actividades. Por tanto, después de un número corto de iteraciones, es normal evitar partes del desarrollo como la especificación, y continuar con las fases posteriores. Los problemas se van dejando para después o son ignorados. Esta falta de atención a los requisitos puede significar que el sistema no haga lo que los usuarios quieren. También puede ocurrir que los sistemas estén mal estructurados o diseñados. Si los errores están en la recogida de requisitos o en el diseño, pueden no ser descubiertos hasta la fase de implementación y pruebas.

Otras variaciones del modelo en cascada intentan paliar este problema (McConnell, 1996):

- *Modelo Sashimi*: Modelo en cascada con fases solapadas. Este modelo sugiere estar diseñando antes de considerar el análisis de requisitos completado, por ejemplo. Las fronteras son más ambiguas por lo que el trabajo en paralelo resulta más complejo.

- *Modelo cascada con subproyectos*: El proyecto parte de un único análisis de requisitos y un diseño arquitectural, pero se divide después en subsistemas independientes lógicamente.
- *Modelo en cascada con reducción del riesgo*: Sitúa una espiral de reducción del riesgo al comienzo del proyecto, con prácticas de recogida de requisitos mediante prototipos de interfaz de usuario, storyboarding, entrevistas, etc.

Otras evoluciones del modelo en cascada dan lugar a opciones muy conocidas, como el modelo V-Shaped y el modelo en espiral, que se describen brevemente a continuación.

El modelo V-Shaped

En el modelo V-Shaped (Dwayne, 1997) destacan las planificaciones para evaluaciones tempranas. Al igual que el modelo en cascada, este ciclo de vida (Figura 2.2) propone unas etapas secuenciales; cada una de ellas debe completarse antes de empezar la siguiente. La diferencia principal es que hay planes de test en cada fase.

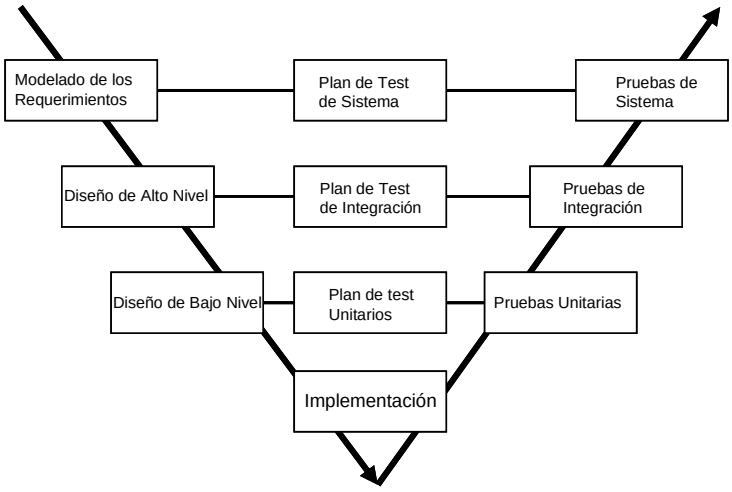


Figura 2.2 El ciclo de vida V-Shaped

El desarrollo comienza con la recogida de todos los requisitos sin embargo, antes de pasar a la siguiente fase, se efectúa un plan de pruebas. En esta fase, el plan de pruebas desarrolla pruebas para verificar la funcionalidad especificada en la definición de requisitos. En la siguiente fase, se diseña la arquitectura del sistema a nivel general. El plan de test a este nivel

comprobará la integración correcta de todas las unidades finalizadas. Las fases de diseño a bajo nivel se centran en el diseño de los componentes de software individuales y también, en los planes de test unitarios desarrollados en esta fase para probar los componentes individuales. La implementación se realiza al final (aparece en el vértice de la V). Una vez que la codificación finaliza, el desarrollo asciende por la parte derecha de la V, moviéndose por los planes desarrollados en las fases anteriores. Si surge un problema en las pruebas, el ciclo de vida comienza en esa fase.

El énfasis de este modelo en planificar test de prueba reduce los riesgos de mal funcionamiento. Sin embargo sus desventajas son las mismas que las del modelo en cascada, y este modelo no se recomienda para proyectos complejos.

El modelo en espiral

El modelo en espiral de Boehm (Boehm, 1988) incluye características del modelo en cascada junto con el prototipado y el análisis de riesgos.

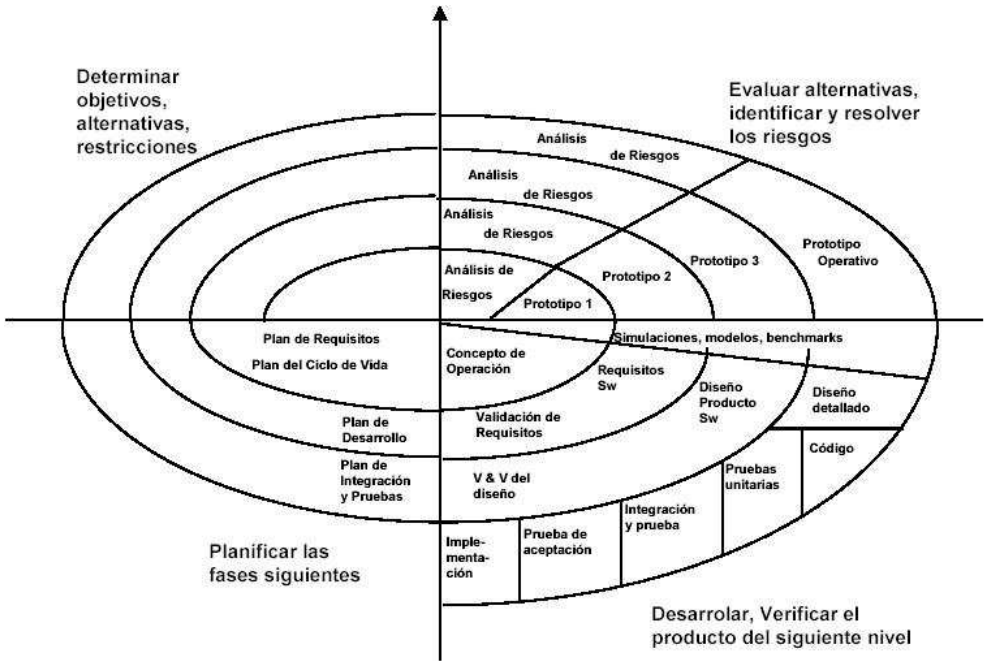


Figura 2.3 Modelo en espiral (http://m0del0espiral.blogspot.com.es/)

Un proyecto software pasa muchas veces por cuatro fases visualizado en una espiral que va ampliándose en número de iteraciones y coste. Las cuatro fases de cada bucle son: establecer

objetivos, análisis de riesgos, desarrollo y validación, y planificación. Cada vuelta incide en un paso del modelo en cascada. Así, la vuelta inicial trata de la factibilidad del sistema; la segunda, la especificación de los requisitos; la siguiente, el diseño del sistema, etc. El énfasis en la planificación repetida del proyecto y en la evaluación con el cliente le da a este modelo un carácter fuertemente iterativo.

Una de las ventajas de este modelo es que a medida que el coste se incrementa, el riesgo decrece. Cuanto más tiempo y dinero se invierta, menos riesgo habrá (McConnell, 1996). Existen puntos de validación al final de cada iteración y el análisis de riesgos proporciona indicaciones tempranas. Si el proyecto no puede hacerse, se descubrirá pronto y no aumentará el coste. El modelo espiral es un enfoque realista para el desarrollo de sistemas y de software a gran escala. Como el software evoluciona a medida que el proceso avanza, el desarrollador y el cliente comprenden y reaccionan mejor ante los riesgos en cada nivel de evolución (Pressman, 2010). La desventaja mayor es su complicación ya que exige una gestión exhaustiva que a veces no es necesaria.

La visión inicial de este modelo no difiere mucho del modelo en cascada y por lo tanto hereda muchos de sus problemas. Sin embargo, son posibles otras visualizaciones sobre este modelo (Maciaszek and Liong, 2004). Por ejemplo, cada bucle puede ser considerado como una iteración que concluya con un producto y los fragmentos sucesivos son incrementos; todos los bucles en la espiral podrían realizar análisis de requisitos, en cuyo caso los fragmentos de ingeniería estarían relacionados con esos requisitos en construcción y se efectuaría un prototipo que los cubriera.

2.2.1.3 El fracaso del modelo en cascada

Hasta los años 90 dominaba el modelo en cascada pese a la controversia que generaba (Hanna, 1995) (Cotterman et al., 1981) (McCracken and Jackson, 1982) (Dijkstra, 1979). Algunas discusiones sobre este modelo se resumen en (Parnas and Clements, 1986):

- Los usuarios de un sistema saben exactamente lo que quieren pero no pueden expresarlo.
- Incluso si pudiéramos establecer todos los requisitos, hay muchos detalles que solo descubriríamos durante la implementación.

- Incluso si supiéramos todos esos detalles, como humanos, solo podríamos dominar parte de tanta complejidad.
- Incluso si pudiéramos dominar tanta complejidad, fuerzas externas añadirían cambios en los requisitos, algunas de las cuales podrían invalidar decisiones anteriores.

Varios autores hablaban de índices de fracaso en proyectos realizados con esta enfoque ("The Standish Group International, Inc. The CHAOS Report", 1994.) (Jarzombek, 1999) (Johnson, 2002) (Taylor, 2000). En concreto, un estudio sobre proyectos del Departamento de Defensa de EEUU, concluye que el 75% de los proyectos realizados con el estándar DOD-STD-2167 que promociona el ciclo de vida en cascada, fracasaron o no fueron nunca usados (Jarzombek, 1999). El problema es que pese a cumplir las especificaciones previstas, no cubrían las necesidades reales de los usuarios. En consecuencia, en los 80, las grandes compañías fueron adoptando otros modelos, cuya motivación para evitar el ciclo de vida en cascada fue que los requisitos eran cambiantes durante el proceso de desarrollo (Madden and Rone, 1984).

2.2.2 El desarrollo iterativo

El desarrollo iterativo es un enfoque al desarrollo de software en la que el ciclo de vida está compuesto de varias iteraciones en secuencia.

El desarrollo de software es un proceso complejo, continuo, iterativo, y repetitivo. Los procesos que se basan en especificar completamente los requisitos, luego diseñar, implementar y finalmente probar el sistema, no son adecuados para generar desarrollo de software rápidamente (Wong, 1984). MacCormack dice que la especificación total no puede hacerse *upfront*, y que la entrega particionada de un producto ayuda a determinar las prioridades del trabajo a ser realizado en los pasos sucesivos (MacCormack, 2001). En definitiva, la esencia de los procesos iterativos es que la especificación se desarrolla en conjunción con el software (Sommerville, 2006). Esta idea supone el gran cambio sobre el modelo "en cascada" tradicional. Sin embargo, su aparición fue progresiva; Royce, en su visión del modelo en cascada en los 70, ya recomendaba hacerlo con dos iteraciones. Sugería que un proyecto de 30 meses debía tener un modelo piloto de 10 meses, y justificaba su necesidad cuando el proyecto contenía elementos novedosos y factores desconocidos. Hay,

por lo tanto, insinuaciones de desarrollo iterativo en el artículo de Royce. Un poco más tarde, Harlan Mills escribió "*Top-Down Programming in Large Systems*" (Mills, 1971), donde promovía el desarrollo iterativo. Mills sugería generar una secuencia de sistemas intermedios de tal manera que a cada paso pudieran ser verificados. Sin embargo, no evitaba el paso largo de la especificación up-front, ni indicaba el feedback ni la adaptación en cada iteración. En los años sucesivos, el desarrollo iterativo derivó en el modelo incremental, en el modelo evolutivo y en los modelos ágiles. En muchos casos, las fronteras de estos modelos no están claras, ya que además del aspecto iterativo, comparten otras características.

2.2.2.1 El modelo incremental

El modelo incremental es aquél que produce "incrementos" de software susceptibles de entregarse. En cada incremento se proporciona un subconjunto de la funcionalidad del sistema. Puede combinarse con los flujos del proceso lineal del modelo en cascada. En este caso, cada secuencia lineal produce un incremento de software entregable. Cuando se utiliza un modelo incremental, es frecuente que el primer incremento sea "el producto fundamental" que abarca los requisitos básicos. Después de su evaluación o su uso por el cliente, se establece el plan para producir el incremento siguiente, que puede incluir la mejora del producto fundamental. En los modelos más dinámicos, la especificación, el diseño, la implementación y las pruebas se entremezclan. Las características "iterativo" e "incremental" son la piedra angular de las metodologías ágiles. En los sistemas cambiantes actuales, los procesos de desarrollo que se enfocan a la entrega rápida son necesarios para que el cliente pueda verificar en uso el sistema o subsistema en un plazo corto.

La práctica moderna del Desarrollo Iterativo e Incremental (DII), con el refinamiento dirigido por el feedback, la implicación del cliente y las iteraciones claramente delineadas, se aplicó en IBM para proyectos del Departamento de defensa en 1972 (Larman and Basili, 2003). El equipo organizó el proyecto en 4 iteraciones cerradas de 6 meses cada una. Aunque se realizó un esfuerzo grande en la especificación up-front, se percataron de que el enfoque DII era una manera de gestionar la complejidad y los riesgos de los desarrollos largos.

En 1975, Vic Basili y Joe Turner publican "*Iterative Enhancement: A Practical Technique for Software Development*"(Basili and Turner, 1975), donde definen las características de los DII clásicos: "*La idea de las mejoras iterativas es desarrollar un sistema de software*

incrementalmente, permitiendo al desarrollador tener en cuenta lo aprendido en las anteriores versiones. Este aprendizaje viene del desarrollo, y también del uso del sistema. El proceso comienza con una implementación simple de un subconjunto de los requisitos software, e iterativamente se mejora la secuencia de versiones hasta que el sistema completo sea implementado. En cada iteración, las modificaciones de diseño se hacen a medida que se añaden nuevas capacidades funcionales". Es en este momento cuando se habla de los inconvenientes de los requisitos up-front y de la necesidad de la participación del usuario y la replanificación en los proyectos grandes (Mills, 1976). Así mismo, surge la idea de la integración de componentes software al final de cada iteración. Cada paso iterativo finaliza por algunos de estos criterios (en orden de prioridad): finaliza lo planificado, llega a un punto de equilibrio, se puede medir algún beneficio para el usuario o al menos puede haber algún feedback en el entorno de usuario y por lo tanto, aprendizaje. La gestión no exige que se estime la fecha de finalización y el coste de todo el proyecto.

Los métodos con características iterativa e incremental son la tendencia actual y son numerosos los autores que ven ventajas en su utilización. El departamento de defensa de EEUU cambió los estándares que promovían el modelo en cascada por nuevos estándares que impulsaban el uso de métodos DII. En 1994, el estándar Mil-Std-498 establecía las características de los procesos DII: "*Si un sistema se desarrolla en trozos, sus requisitos y su diseño no deben quedar totalmente definidos hasta el final*". Pese a ello, también se pueden leer opiniones contrarias. A continuación se listan algunas de las ventajas e inconvenientes, así como se proponen áreas para su aplicación.

Ventajas del desarrollo incremental

- Los clientes no tienen que esperar a la entrega del sistema completo. La primera entrega satisface los requisitos principales y pueden empezar a usar el software.
- Los clientes pueden usar las entregas tempranas como prototipos y entender así los requisitos para los incrementos posteriores.
- Menos riesgo de fallos de proyecto. Aunque se encuentren problemas en algunos incrementos, hay seguridad de que algo será entregado al cliente.

- Los servicios más importantes del sistema se verifican varias veces ya que debido a su prioridad se realizan primero y los incrementos posteriores se integran con ellos.

Inconvenientes del desarrollo incremental

- Puede ser dificil hacer coincidir los requisitos del cliente con los incrementos adecuados ya que éstos deben ser pequeños y entregar alguna funcionalidad del sistema. (Sommerville, 2006).
- Como los requisitos no se definen en detalle hasta que un incremento es implementado, puede ser difícil identificar especificaciones comunes necesarias para todos los incrementos (Sommerville, 2006).

Áreas y situaciones adecuadas para su aplicación

El proceso incremental es útil cuando no se dispone de personal para la implementación completa del proyecto, en el plazo establecido por el negocio (Pressman, 2010). Además, los incrementos se planifican para administrar riesgos técnicos, cuando no se dispone al principio de todos los recursos hardware, por ejemplo.

2.2.2.2 El modelo evolutivo

El modelo evolutivo se basa en desarrollar una implementación inicial, valorarla con los usuarios, y refinarla a través de muchas versiones hasta que el sistema adecuado se considere finalizado. El término "*evolución*" aparece por primera vez en 1976 en "*Software metrics*" (Gilb, 1977). La "evolución" es una técnica para producir la apariencia de estabilidad. Este modelo es un refinamiento del modelo incremental en el que se exige la captura de feedback del producto instalado, y se utiliza para guiar la siguiente entrega. Esto es, en la entrega incremental "pura" se define un plan de futuras entregas, mientras que en el método de entregas evolutivas, el feedback no está dirigido por un plan de entregas; cada entrega se crea dinámicamente por la información que va llegando (Larman, 2003).

Los modelos evolutivos son iterativos. Un sistema complejo se implementa en pasos pequeños, y en cada paso se establece una medida de logro o una posibilidad de retirarse hasta

el anterior paso aceptado. Se tiene la oportunidad de recibir feedback del mundo real antes del despliegue total, y se pueden corregir posibles errores de diseño.

El modelo evolutivo recomienda entregas frecuentes a los implicados, cada pocas semanas, de resultados útiles (Gilb, 1985). La idea es ser más simple, más seguro, y más rápido en el desarrollo de sistemas de software, construyendo una versión mínima, colocándola en uso, y luego añadirle funcionalidades (y otras cualidades) de acuerdo a prioridades que surgen de su uso.

Los primeros enfoques continuaban estando dirigidos por la documentación. El método Cleanroom, por ejemplo, es una evolución del DII que incorpora desarrollo evolutivo con métodos más formales de especificación y pruebas (Mills et al., 1987). En los 90, los métodos tendían a realizar menor trabajo de especificación al principio y un análisis evolutivo más fuerte. Gilb describe el método Evo (Gilb and Finzi, 1988), que se distingue por una entrega evolutiva frecuente y un énfasis en definir objetivos cuantificables, y luego medir los resultados de cada iteración corta y "*time-boxed*" (fijar el fin de la iteración).

Ventajas e Inconvenientes

Comparten las ventajas e inconvenientes de los métodos incrementales. Además, el hecho de que los requisitos, el plan, la estimación, y la solución evolucione o sea refinada a lo largo de las iteraciones, produce una mejor aceptación del cambio pero también mayor incertidumbre del plan total al principio del desarrollo.

Áreas y situaciones adecuadas para su aplicación

Pressman recomienda usar un modelo evolutivo cuando los requisitos iniciales estén razonablemente bien definidos, pero el alcance general del esfuerzo de desarrollo imposibilite un proceso lineal (Pressman, 2010). Si además existe la necesidad de entregar cierta funcionalidad a los usuarios que se aumentará en entregas posteriores de software, se elige un modelo de proceso evolutivo.

2.2.2.3 El modelo de Prototipado Rápido

Este modelo es una variante del método evolutivo en el que se usan los prototipos como una técnica que ayuda cuando los requisitos no están claros (Pressman, 2010). El prototipado rápido (McConnell, 1996) es un modelo de ciclo de vida que comienza desarrollando el concepto inicial del sistema, esto es, los aspectos más visibles. Tras la aceptación por el cliente, se continua refinando el prototipo hasta que el cliente y el desarrollador consideran que es "suficientemente bueno". Llegado a este punto, se completa el trabajo y se entrega el prototipo como el producto final. Pressman señala que aunque puede tener inconvenientes, hacer prototipos es un paradigma eficaz para la IS (Pressman, 2010). La clave, en su opinión, es que todos los participantes deben estar de acuerdo en que el prototipo sirva como mecanismo para definir los requisitos. Después se descartará y se desarrollará con la mirada puesta en la calidad.

En los años 80 se empieza a utilizar el prototipado evolutivo, un enfoque de proceso evolutivo que no incluye iteraciones "*time-boxed*" normalmente. En estos años se utilizó en sistemas de inteligencia artificial. Es frecuente que el cliente defina un conjunto de objetivos generales pero no pueda detallar los requisitos. El proceso comienza convocando a los implicados para definir los objetivos generales del software, identificar requisitos y detectar las áreas en las que es necesaria una mayor definición. Se realiza un modelado en forma de "diseño rápido", que se centra en la representación de los aspectos del software que serán visibles para los usuarios finales. El diseño rápido lleva a la construcción de un prototipo que es evaluado por los implicados, proporcionando retroalimentación para mejorar los requisitos y el propio prototipo.

El concepto del prototipado rápido ya se observaba en el modelo en espiral (Boehm, 1988) que, como hemos visto, sugiere la construcción de una serie de prototipos para ayudar a los desarrolladores a identificar y reducir los riesgos mayores asociados con un proyecto. A finales de los 80, algunas compañías empleaban un modelo de prototipado rápido que enfatizaba la construcción de prototipos tempranos para ayudar a establecer los requisitos del cliente (Connell and Shafer, 1989). También, en los entornos variables, que tienen muchos cambios y tecnología que cambia frecuentemente, como los sistemas web, se utilizan prototipos que son mostrados pronto a los clientes en el proceso de desarrollo (MacCormack, 2001).

En algunos casos, el diseño inicial que los clientes prueban son versiones en uso del producto. Sin embargo, otros autores utilizan prototipos "para tirar" para establecer una dirección en el trabajo de diseño inicial (MacCormack, 2001). Este es el caso, por ejemplo, de los prototipos en los modelos iniciales en el modelo en espiral.

Ventajas e Inconvenientes

Entre las ventajas más comentadas del prototipado evolutivo están la visualización temprana de signos de progreso (McConnell, 1996), la percepción del sistema real por parte de los usuarios, y que los desarrolladores logran construir algo de inmediato (Pressman, 2010). En cuanto a los inconvenientes se destacan los siguientes:

- Es imposible saber al comienzo del proyecto cuánto tiempo llevará la realización de un producto aceptable (McConnell, 1996).
- El prototipado evolutivo, en realidad, incluye implícitamente análisis de requisitos, diseño, codificación y pruebas, como en los enfoques tradicionales, pero en incrementos más pequeños (McConnell, 1996).
- Los clientes perciben lo que parece ser una versión funcional del software, sin darse cuenta de que no se consideraron aspectos generales de calidad y que esto cambiará después (Pressman, 2010).
- Es frecuente la elección rápida, poco meditada, de elementos de implementación (algoritmos, lenguajes, sistemas operativos) (Pressman, 2010).

Áreas de aplicación

Este modelo es muy útil cuando los requisitos cambian continuamente o no se entiende bien la aplicación. (McConnell, 1996)

2.2.3 Ingeniería de software basada en componentes

Este enfoque de desarrollo de software llamado *Component-Based Software Engineering* (CBSE), se basa en la reutilización de componentes software y en su integración. Un

componente software es una unidad de software que puede ser dispensado independientemente y estar sujeto a composición por terceras partes. Szyperski define un componente software como una unidad de composición con interfaces especificadas contractualmente, y únicamente con dependencias de contexto explícitas (Szyperski et al., 2002).

Las fases de esta enfoque son:

1. *Especificación de los requisitos*. Similar a la actividad de recogida de requisitos en otros modelos.
2. *Análisis de componentes*. A partir de la especificación de los requisitos, se realiza una búsqueda de los componentes que implementan esa especificación.
3. *Modificación de los requisitos*. Los requisitos son analizados utilizando la información sobre los componentes que han sido descubiertos. Son modificados para adaptarse a los componentes disponibles. Si la modificación es imposible, la actividad de análisis debe rehacerse para buscar soluciones alternativas.
4. *Diseño del Sistema con reutilización*. El framework del sistema se diseña, o bien se reutiliza un framework existente. Los diseñadores tienen en cuenta los componentes que han sido reutilizados y organizan el framework para recogerlos. Se diseña nuevo software si los componentes reutilizados no están disponibles.
5. *Desarrollo e integración*. Se desarrolla el software que no se dispone, y los componentes se integran en un nuevo sistema. La integración puede ser parte del proceso de desarrollo más que una actividad separada.

En la composición de componentes hay que considerar la funcionalidad que se busca del sistema, los requisitos no funcionales y la facilidad con que un componente puede ser reemplazado por otro cuando el sistema cambie. Hay distintos tipos de composiciones (Sommerville, 2006):

- Composición secuencial: los componentes se ejecutan en secuencia.
- Composición jerárquica: un componente llama directamente a los servicios que provee otro componente.

- Composición aditiva: las interfaces de dos o más componentes se suman para crear un nuevo componente.

La composición de componentes reusables, que no hayan sido escritos para la aplicación, normalmente necesitan adaptadores para reconciliar interfaces diferentes (nombres de métodos, incompatibilidad en los parámetros, etc.).

Esta enfoque fundamenta el desarrollo de sistemas, basado en la *integración de servicios web,* que considera un proveedor estándar de servicio como un componente. Cuando un sistema necesita algún servicio, llama a un proveedor para que le proporcione ese servicio sin preocuparse de lo que ese componente esté ejecutando o el lenguaje de programación utilizado para desarrollar el componente. Por ejemplo, un componente podría proporcionar un servicio de búsqueda, y otro componente podría proporcionar un servicio de conversión que convierta de un formato gráfico a otro.

Un componente visto como un proveedor de servicio tiene dos características:

- El componente es una entidad ejecutable. El código fuente no está disponible.
- Los servicios ofrecidos por un componente están disponibles a través de una interfaz, y todas las interacciones se realizan a través de dicha interfaz. Esta interfaz se expresa en términos de operaciones parametrizadas y su estado interno no se expone.

Ventajas del CBSE

- La reducción del software a desarrollar y por lo tanto la disminución de tiempo y coste (Pressman, 2010) (McConnell, 1996) (Szyperski et al., 2002).
- Reduce el riesgo del proceso ya que el costo del software existente es conocido (Sommerville, 2006).
- Permite disponer antes del producto (Sommerville, 2006).
- Mejora la confianza en el producto ya que los componentes reusables han sido ya probados y usados (Sommerville, 2006) (Szyperski et al., 2002).
- Los especialistas pueden usar su conocimiento, desarrollando software reusable (Sommerville, 2006)

- Algunos elementos estándar pueden ser implementados como un conjunto de componentes reusables estándar. Esto es interesante por ejemplo para desarrollar interfaces de usuario estándar (Sommerville, 2006).

Desventajas del CBSE

- Esfuerzo inútil si los componentes no se eligen con cuidado y no son los que se necesitan (McConnell, 1996).
- Incremento de los gastos de mantenimiento. Si el código fuente de un componente no está disponible, los costos de mantenimiento pueden crecer porque los elementos reusables pueden ser incompatibles con los cambios del sistema (Sommerville, 2006).
- Hay un desequilibrio inevitable entre reusabilidad y usabilidad en componentes (Sommerville, 2006). Hacer un componente reusable implica proveer de un conjunto de interfaces genéricas con operaciones que cubran todas las formas en las que el componente puede ser usado. Sin embargo, hacer un componente usable significa proveer una interfaz simple y mínima que sea fácil de entender.
- La validación de los componentes. Esta actividad implica la realización de pruebas para el componente que permitan confirmar que ese componente es el que se necesita. El problema está en que la especificación del componente, normalmente realizada de manera informal, puede no estar suficientemente detallada como para poder desarrollar un conjunto completo de pruebas para el componente.
- El control de la evolución hacia nuevas versiones se pierde ya que los componentes reutilizables no están bajo control de los que los utilizan.
- Puede ser imposible integrar herramientas CASE con la librería de componentes (Sommerville, 2006).

Áreas y situaciones adecuadas para su aplicación

Este modelo es adecuado en proyectos grandes y distribuídos. Igualmente, en proyectos que deban acortar su tiempo de desarrollo y aquéllos que quieran reutilizar módulos en las nuevas versiones para disminuir costes.

2.2.4 El desarrollo ágil

Los métodos ágiles aplican desarrollo evolutivo en iteraciones "timeboxed" y planificación adaptativa. Además, promocionan la entrega evolutiva, e incluyen otros valores y prácticas que enfatizan la *agilidad*; esto es, la respuesta rápida y flexible al cambio (Larman, 2003) .

2.2.4.1 Manifiesto Ágil

En 2001, un grupo de desarrolladores de software y consultores formaron la "Alianza Ágil", y firmaron el "Manifiesto por el desarrollo ágil de software". En él se establecía lo siguiente ("Manifesto for Agile Software Development", 2001):

- *Individuals and Interactions are more important than processes and tools*. El cliente y los desarrolladores se convierten en el foco del proceso. En este sentido, dos autores del manifiesto, Cockburn y Highsmith, dicen: "*El desarrollo ágil se centra en los talentos y habilidades de los individuos, y adapta el proceso a personas y equipos específicos*" (Cockburn and Highsmith, 2001).
- *Working software over comprehensive documentation*. El software que funciona es más importante que la documentación exhaustiva.
- *Customer collaboration over contract negotiation*. Flexibilidad al cambio en los requisitos. Esto es, el cliente puede provocar el cambio en la especificación de la aplicación en cualquier momento. La satisfacción del cliente es la medida válida para aceptar un producto o subproducto software.
- *Responding to change over following a plan*. Un equipo ágil responde siempre bien ante los cambios. No se atiene a un plan fijo y establecido al inicio.

2.2.4.2 Principios de los Procesos Ágiles

La Alianza Ágil define 12 principios de agilidad ("Agile Alliance: The Twelve Principles of Agile Software", 2001-13):

1. La prioridad más alta es satisfacer al cliente a través de la entrega pronta y continua de software valido.

2. Son bienvenidos los requisitos cambiantes, aún en una etapa avanzada del desarrollo
3. Entregar software que funcione a menudo, con frecuencia de dos semanas a un par de meses, preferentemente lo más pronto que se pueda.
4. Las personas del negocio y los desarrolladores deben trabajar juntos, a diario y durante todo el proyecto.
5. Hay que desarrollar los proyectos con individuos motivados. Debe darse a éstos el ambiente y el apoyo que necesiten, y confiar en que harán el trabajo.
6. El método más eficiente y eficaz para transmitir información a los integrantes de un equipo de desarrollo, y entre éstos, es la conversación cara a cara.
7. La medida principal de avance es el software que funciona.
8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.
10. La simplicidad es esencial: el arte de maximizar la cantidad de trabajo no realizado.
11. Equipos con organización propia. El equipo se organiza a sí mismo para hacer el trabajo. El equipo selecciona cuánto trabajo cree que puede realizar en cada iteración, y se compromete con la labor (Schwaber, 2001). La organización propia sirve para mejorar la colaboración y elevar la moral del equipo (Pressman, 2010).
12. El equipo reflexiona a intervalos regulares sobre cómo ser más eficaz, para después afinar y ajustar su comportamiento en consecuencia.

2.2.4.3 Motivación

Los sistemas actuales operan en entornos cambiantes. El software debe desarrollarse rápidamente en un mercado muy competitivo. Los procesos que se basan en especificar completamente los requisitos, luego diseñar, implementar y finalmente probar el sistema no son adecuados para generar desarrollo de software rápido (Sommerville, 2006).

El desarrollo ágil surgió para hacer frente a la necesidad imperante de realizar cambios frecuentes. Las metodologías ágiles (MA) son un enfoque alternativo al desarrollo de software

ante el fracaso de los enfoques tradicionales en numerosos proyectos. El fallo de un proyecto no significa únicamente que sea cancelado o distribuido muy tarde, sino que no cumpla las expectativas (Larman, 2003, p. 50).

El desarrollo ágil se caracteriza por equipos de desarrollo pequeños, ciclos de vida iterativos, y el énfasis en los entregables en lugar de en la documentación, como sucede con los desarrollos tradicionales. El empleo de métodos ágiles reduce el coste de los cambios cuando el proyecto está en un estado avanzado (Beck and Andres, 2004) (Ambler, 2002).

El proceso ágil es adaptable al cambio rápido del proyecto y además, es incremental. Promueve la entrega de incrementos de software en periodos cortos de tiempo. Craig Larman en "*Challenges of Scaling Scrum to Large Organization*s" ("InfoQ: Craig Larman on the Challenges of Scaling Scrum to Large Organizations," 2011) cambia el concepto de "arquitectura" (una fase tras otra hacia la construcción final) por el de "jardinería" (hacer un poco de todo, intercalando fases).

Cuando las MA se impusieron, el contexto tecnológico había evolucionado mucho (Jiang and Eberlein, 2009):

- PCs potentes de bajo coste, posibilidades de almacenamiento ilimitado
- Facilidades de conexión en red
- Lenguajes orientados a objetos como C++ o Java
- Internet y tecnologías Web disponibles ampliamente
- Tecnologías de programación visual y GUI fuertemente interactivos
- Entornos de programación avanzados que ayudan al diseño de interfaces de usuario interactivas y permiten la implementación rápida de prototipos.
- Herramientas de gestión de requisitos
- Soporte de herramientas para el testeo automático.
- Facilidades hardware para contactar e implicar a los clientes mediante el uso de móviles, videoconferencias y herramientas de Internet.
- Herramientas de gestión de proyectos

- Herramientas para gestionar las versiones de un proyecto

Estos avances incrementaron la ambición de la industria del software y permitió a los desarrolladores tratar con proyectos complejos. Por una parte, las aplicaciones modernas deben estar en el mercado rápidamente. A veces, es más importante servir a los clientes necesidades básicas, rápidamente, que cumplimentar todos los requisitos y entregar tarde (Eckstein, 2004). Por otra parte, la documentación no debe ser compleja ni exhaustiva, con fases a cumplimentar, si no que son apuntes, notas, borradores; es más ligera.

2.2.4.4 Evolución en el desarrollo ágil

Varios autores argumentan la relación existente entre las metodologías clásicas en IS y las MA. Así, Abbas dice que algunas prácticas ágiles tienen raíces en las prácticas antiguas (Abbas et al., 2008).

Scrum ("Scrum Guides | Scrum.org - The home of Scrum," n.d.) es un método de desarrollo ágil de software concebido por Jeff Sutherland a principios de la década de 1990. Schwaber y Beedle lo han refinado posteriormente (Schwaber and Beedle, 2001). Scrum utiliza patrones de proceso de software o sprint en los que se define un grupo de acciones de desarrollo. Un sprint es una unidad de trabajo necesaria para alcanzar un requisito. Durante el sprint no se introducen cambios, de tal manera que el sprint permite a los miembros del equipo trabajar en un ambiente de plazo corto, pero estable. Se establecen reuniones breves de 15 minutos a diario, dirigidas por un lider (Scrum master). Esto ayuda al equipo a descubrir problemas y a socializar el conocimiento.

El desarrollo rápido (RAD: Rapid Application Development) surge en 1994 y supone una discusión sobre el proceso iterativo previo. Esto deviene en el método DSDM (Dynamic Systems Development Method) (Stapleton and Consortium, 2003), un método iterativo e incremental que define una fase de preparación, un estudio de viabilidad y un estudio de negocio, secuencialmente para la adquisición de normas para el resto del desarrollo. Cada desarrollo realiza: iteración del modelo funcional, iteración del diseño y construcción, y finalmente implementación. Termina con una fase de conclusión en la que se deja la solución operativa. El mantenimiento es una parte más del ciclo de vida. DSDM se rige por la

involucración del cliente, el poder de toma de decisiones del equipo del proyecto, las entregas frecuentes, los cambios reversibles, las pruebas continuas y la fuerte comunicación.

A mediados de los 90 apareció el proceso unificado de Rational (RUP: Rational Unified Process) (Booch, 2006). Las actividades destacan por la creación y el mantenimiento de modelos más que por la documentación en papel. Este enfoque encaja especialmente bien con UML, y su desarrollo está centrado en la arquitectura. El proceso establece al principio una arquitectura software que guía el desarrollo del sistema y las actividades están dirigidas por los casos de uso. RUP impulsa un control de calidad y una gestión del riesgo continuos. Este proceso tiene cuatro fases: iniciación, elaboración, construcción y transición; las dos primeras incluyen actividades de diseño, y las dos últimas su producción. Una iteración representa un ciclo de desarrollo completo, desde la captura de requisitos hasta la implementación y pruebas, que produce como resultado la entrega al cliente de un producto ejecutable.

En la misma época, Kent Beck introdujo XP en su libro *Extreme Programming Explained* (Beck, 1999), como un método orientado al desarrollo con énfasis en la comunicación, simplicidad y evaluación. Su principal objetivo era la reducción de los costes de producción de software. XP es el método que más se practica en el ámbito académico y en la industria. XP se rige por los cinco valores y los doce principios ágiles, y sus actividades principales incluyen planificación, diseño, codificación y pruebas.

En 1999 apareció el proceso iterativo FDD: Feature-Driven Development en el libro *Java Modeling Color with UML* (Coad et al., 1999) . FDD describe cinco actividades básicas: desarrollar el modelo general, construir una lista de "*features*", planificación por feature, diseño por feature y construcción por feature (Palmer and Felsing, 2002). Al igual que los métodos ágiles, FDD se centra en iteraciones cortas que entregan incrementos software completamente funcionales. FDD utiliza un enfoque basado en modelos UML, que se desarrollan para ayudar a visualizar y poner en práctica cada incremento de software .

Posteriormente, se han sumado otros métodos ágiles con diferentes grados de éxito. Crystal Family, entre ellos Chrystal Clear (Cockburn, 2004), es una colección de métodos creados por Alistair Cockburn y Jim Highsmith que permite a los desarrolladores lograr "maniobrabilidad" cuando surja la necesidad. El movimiento del *Lean Software* (Larman and

Vodde, 2008), basado en principios económicos y matemáticos, describe un marco de trabajo con cinco elementos:

- El tejado: Los objetivos de producción
- El pilar 1: Respeto a las personas
- El pilar 2: Mejoras continuas
- La base: El soporte a la gestión
- Los contenidos: El flujo de desarrollo del producto

Estos principios han inspirado el desarrollo del método Kanban que define una manera de gestionar el trabajo de software en función de alguna "unidad de valor" (puede ser una user story, característica o requisito mínimo), realizado con restricciones WIP (work in progress) ("limitedwipsociety," 2013). Se basa en visualizar el flujo de trabajo, limitar el trabajo en curso, medir y gestionar el flujo, hacer expícitas las reglas del proceso y utilizar modelos para reconocer las oportunidades de mejora.

Ventajas e Inconvenientes

Las ventajas vienen dadas por sus características diferenciales de los métodos clásicos. Así, los clientes (usuarios y propietarios del sistema) trabajan con el equipo de desarrolladores a lo largo del ciclo de vida, no únicamente al principio y/o al final. El feedback constante del cliente evita la firma de un contrato formal para el producto total y la documentación extensa necesaria para la transferencia de conocimiento (Maciaszek and Liong, 2004). La continua integración y los ciclos cortos permiten aceptar los cambios de forma temprana.

Chow recoge, en una revisión exhaustiva, una lista de factores de fracaso en proyectos ágiles (Chow and Cao, 2008) que nos remiten a los puntos débiles de este método, que deben ser tenidos en cuenta para su aplicación.

Los factores de fracaso, se clasificaron en cuatro categorías:

- Organizativos: Falta de liderazgo ejecutivo o de compromiso en la gestión, tener una cultura organizativa demasiado tradicional, política o de un tamaño demasiado grande, o una falta de soluciones logísticas ágiles.

- Personas: Falta de habilidades necesarias, límites para el trabajo en equipo, resistencia en los grupos o individuos, o relaciones malas con el cliente.
- Proceso: Una mala definición del alcance del proyecto, de sus requisitos, de la planificación, falta de mecanismos para el seguimiento del progreso del proyecto, falta de presencia del cliente o mal definido su papel.
- Técnicos: Falta de prácticas ágiles correctas y, uso de tecnologías y herramientas inapropiadas.

Áreas y situaciones adecuadas para su aplicación

Las MA tienen éxito en proyectos específicos: complejos y con muchos cambios. Este enfoque obtiene sus mejores resultados en entornos centrados en la gente y organizaciones centradas en la cooperación.

Algunos ingenieros de software tienden a descartar el desarrollo ágil en proyectos grandes porque la mayoría de los procesos ágiles se promueven solo para equipos pequeños, donde la colaboración, la cooperación y la comunicación oral es más fácil. La mayoría de los proyectos que fallan son grandes. Esto se debe en gran medida a una falta de comunicación entre los equipos, gestores, clientes, etc (Eckstein, 2004). La comunicación es uno de los pilares de las MA, es un factor de éxito que se destaca con frecuencia (Chow and Cao, 2008).

2.3 Integración de la Ingeniería de Software y la Ingeniería de la Usabilidad

Según el estándar ISO 9242-11 (1998), el concepto de usabilidad puede definirse como el nivel con el que un producto se adapta a las necesidades del usuario y puede ser utilizado por el mismo para lograr unas metas con efectividad, eficiencia y satisfacción en un contexto específico de uso. Por efectividad se entiende que cualquier software debe tener unos objetivos claros y éstos deben ser alcanzables. La eficiencia se refiere a lograr esos objetivos con los mínimos recursos posibles, y depende de las habilidades del usuario y de las posibilidades del software.

La Ingeniería de la Usabilidad (IngUS) es una disciplina que provee métodos estructurados para conseguir usabilidad en el diseño de la interfaz de usuario (IU en adelante), durante el desarrollo de un producto (Mayhew, 1999). Más recientemente, los objetivos de usabilidad han sido extendidos a otras actividades del desarrollo de software, particularmente al análisis de los requisitos y a la visión del sistema (Rosson and Carroll, 2001). En los últimos años, la usabilidad ha sido un aspecto crucial en el desarrollo de software. Una revisión en 1992 destacaba que prácticamente el 50% del desarrollo de software se dedicaba a la interfaz de usuario (Myers and Rosson, 1992). Otro estudio interesante estimaba en 1992, que el 63% de los proyectos de software sobrepasaban sus previsiones de coste, especialmente por causas relacionadas con la IngUS (Lederer and Prasad, 1992): cambios frecuentes por los usuarios, tareas pasadas por alto, mala comprensión de los requisitos de los usuarios, e insuficiente comunicación entre los analistas y usuarios. De cara a los usuarios finales, la IU es "el producto". Como Mayhew expresa "*Just about their entire experience with the product is their experience with its user interface*" (Mayhew, 1999).

Nielsen define la disciplina de Ingeniería de Usabilidad como un conjunto de actividades que tienen lugar en el ciclo de vida del producto, preferentemente en las primeas etapas, antes incluso de que la IU haya sido diseñada (Nielsen, 1993). Uno de los aspectos importantes en ingeniería de la usabilidad es la inclusión de una especificación de la usabilidad.

Debido a las necesidades especiales de los sistemas interactivos, es preciso aumentar el ciclo de vida estándar a fin de abordar las cuestiones de HCI (Losada et al., 2009b). El resultado es que la construcción del sistema tendrá en cuenta las evaluaciones de usabilidad, y la interacción con los usuarios será observada y evaluada con el fin de determinar la forma de hacerlos más usables (Dix et al., 2003). El experto en usabilidad evalúa los sistemas mediante métodos de usabilidad adecuados que se pueden aplicar desde las primeras etapas del proceso de desarrollo (Mayhew, 1999). Nielsen y Mack proporcionan una buena revisión de estos métodos (Nielsen and Mack, 1994).

2.3.1 Evolución en los ciclos de vida de la Ingeniería de la Usabilidad

Hay tres premisas comunes en los ciclos de vida clásicos utilizados en IngUS (Preece et al., 1994):

- Son centrados en el usuario, en el sentido de que implican a los usuarios a lo largo del proceso.
- Integran el conocimiento de expertos en usabilidad.
- Son fuertemente iterativos, para comprobar que el diseño cumple con las necesidades de los usuarios.

En estos métodos, los requisitos suelen ser recogidos por adelantado (up-front), de forma iterativa pero previamente a la implementación de la lógica de negocio, y la evaluación es una parte esencial que debe comenzar tan pronto como sea posible en el proceso de desarrollo. Al igual que en la IS, ha habido una evolución constante en los enfoques de la IngUS desde los años 70. Una de las primeras referencias es la de Gould and Lewis en la que describen un enfoque basado en tres estrategias (Gould and Lewis, 1985):

- Centrado en los usuarios y sus tareas
- Medidas empíricas
- Diseño iterativo

Entre las visiones más importantes en IngUS están el Diseño Centrado en el Usuario, el Diseño Centrado en el Uso y el Diseño centrado en la Actividad, que se describen brevemente a continuación, y después se describe la primera en más profundidad por ser la mayoritaria.

Diseño Centrado en el Usuario

El Diseño Centrado en el Usuario (DCU) es el enfoque clásico en esta disciplina que involucra a los usuarios durante todo el proceso de diseño y desarrollo, intentando optimizar la usabilidad de un sistema interactivo. En el DCU se tiene en cuenta a los usuarios reales y el equipo de desarrollo trata de entender las necesidades de los usuarios antes de cualquier implementación.

Diseño Centrado en el Uso

Constantine introdujo una variante del DCU: el Diseño Centrado en el Uso (Constantine and Lockwood, 1999). Constantine sugiere concentrarse en las tareas en lugar de en los usuarios. Su diseño Centrado en el Uso se centra en modelos esenciales y entendibles como el *role model*, *task model* o *content model*. Con la aplicación de estos métodos, la HCI es más formal (y efectiva), y la simplicidad de su sintaxis permite la colaboración de los implicados. Los modelos del diseño centrado en el uso pueden aplicarse tanto por el personal de HCI como de la IS. Constantine recomienda usar prototipos abstractos o esenciales (low-fidelity prototyping).

Diseño Centrado en la Actividad

Donald Norman (Norman, 2006) propone el Diseño Centrado en la Actividad, que se centra, al igual que el Diseño Centrado en el Uso, en las tareas del usuario. Sin embargo, además del tiempo en completar una tarea, considera que hay otros factores cruciales como el uso gozoso de una aplicación. Norman apoya la integración del diseño emocional y la consideración fuerte de la satisfacción del usuario.

2.3.2 Diseño Centrado en el Usuario

En ISO 13407 p.7 encontramos una guía de principios del DCU, para conseguir sistemas usables (“ISO 13407:1999 - Human-centred design processes for interactive systems,” 1999):

- Participación activa de los usuarios
- Distribución apropiada de funciones entre el usuario y el sistema
- La iteración de las soluciones de diseño
- Equipos de diseño multidisciplinares

Según establece ISO 13407 p.10, las cuatro actividades esenciales del DCU serían (“ISO 13407:1999 - Human-centred design processes for interactive systems,” 1999):

- Entender y especificar el contexto de uso

- Especificar los requisitos de organización y de usuario
- Producir soluciones de diseño (prototipos)
- Evaluar diseños con usuarios y comparar con los requisitos

La diferencia principal entre el ciclo de vida clásico y el DCU es que en éste, el diseño se centra en el usuario, su trabajo y su entorno, y en cómo la tecnología puede ser desarrollada y diseñada para soportarlo (Preece et al., 1994). Además, se caracteriza por un proceso de diseño rápido e iterativo con evaluaciones continuas.

Los ciclos de vida clásicos en el DCU proporcionan métodos y herramientas para recoger toda la información requerida. La mayoría de ellos son "heavy-weighted" en el sentido de que analizan y documentan todo lo posible sobre usuarios, flujos de trabajo, contextos, etc, al principio del proyecto y en su globalidad. A continuación se exponen algunos métodos que cumplen las premisas del DCU.

Ciclo de vida en estrella

En el ciclo de vida en estrella (Hix and Hartson, 1993), mostrado en la Figura 2.4, la evaluación es el punto central de todas las etapas. El proceso es completamente iterativo, el desarrollo del sistema puede empezar en cualquier fase y a ésta puede seguirle cualquier otra, debido a la evolución constante de los requisitos, el diseño y el producto hasta una escritura definitiva.

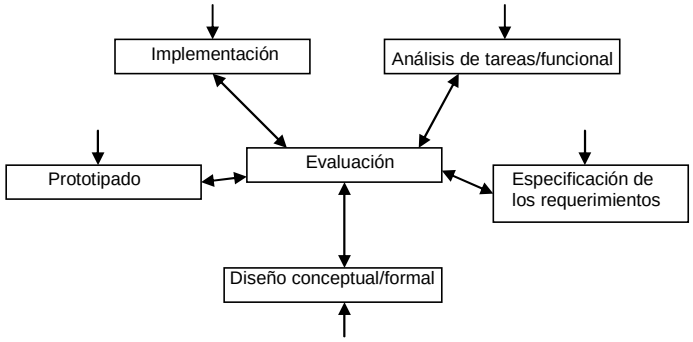


Figura 2.4 Proceso en estrella

Proceso de diseño y desarrollo de IU de Greenberg

Como muestra de la evolución en los ciclos de vida clásicos en el desarrollo de interfaces centrados en el usuario, Greenberg propone un proceso iterativo de diseño de interfaces y evaluación (Greenberg, 1996), que se visualiza en la Figura 2.5.

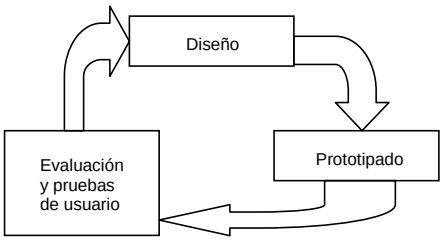


Figura 2.5 Proceso de diseño y desarrollo de IU de Greenberg

Propuesta de Nielsen

El modelo propuesto por Nielsen es un proceso de ingeniería de software, cercano al DCU, iterativo, que incluye consideraciones de usabilidad, incluso antes de que el diseño haya comenzado, en el proceso de desarrollo de software (Nielsen, 1993). El estudio del usuario, su participación y el prototipado son algunos puntos fuertes en esta propuesta.

El ciclo de vida de Ingeniería de la Usabilidad de Mayhew

Este modelo, propuesto en 1999 (Mayhew, 1999), proporciona una descripción detallada de cómo realizar las tareas de usabilidad y especifica cómo éstas pueden ser integradas en los ciclos de vida de IS tradicionales. Incluye una etapa de análisis de requisitos, otra de diseño/pruebas/desarrollo basada en la construcción de prototipos y en la evaluación iterativa, y finalmente una etapa de instalación. Además, incluye explícitamente una guía de estilo para capturar los objetivos de usabilidad del proyecto, que se verifica a cada paso. Es muy útil para aquellos equipos con poca experiencia en usabilidad (Rogers et al., 2011).

Ingeniería de la Usabilidad basada en escenarios

Es otro método DCU clásico (Rosson and Carroll, 2001), que basa las prácticas de evaluación de usabilidad en los escenarios de interacción del usuario. Los escenarios describen los

comportamientos y experiencias del usuario. Ayuda a los diseñadores y analistas a enfocar la atención en el usuario y sus tareas. Las representaciones de escenarios pueden ser elaboradas como prototipos, storyboards o vídeos. Esta enfoque tiene tres fases interconectadas: el análisis del problema en escenarios, el diseño de los escenarios, y el prototipado y evaluación. La primera fase se realiza mediante entrevistas con los implicados, estudios de campo y tormentas de ideas. La segunda traslada el problema hacia su visualización en escenarios de distinto tipo, que describen los detalles de la acción del usuario y su feedback. La tercera fase implementa prototipos que muestran una o más piezas de la solución propuesta en un escenario, y realiza las evaluaciones. Este método distingue entre evaluaciones formativas, realizadas para guiar el rediseño, y evaluaciones sumativas, que sirven para la verificación del sistema.

El proceso MPIu+a

El proceso MPIu+a (Granollers i Saltiveri et al., 2005), mostrado en la Figura 2.6 es una propuesta de la IngUS y de la accesibilidad, basada en tres pilares: la IngUS (columna de la izquierda), el prototipado (columna central) y la evaluación (columna de la derecha).

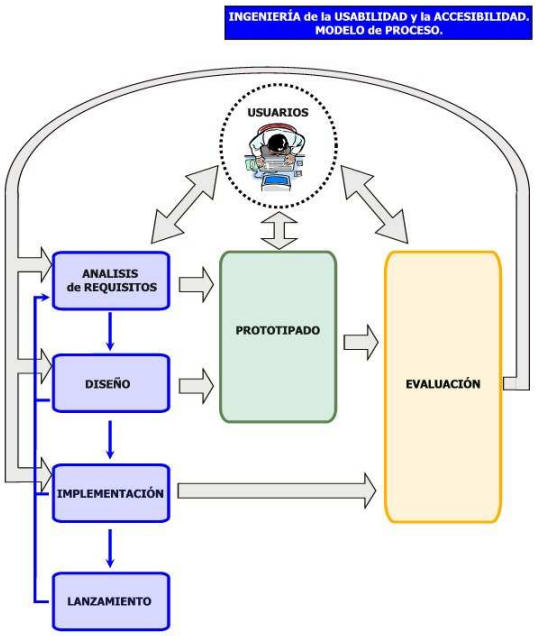


Figura 2.6 Modelo de proceso de la ingeniería de la usabilidad y de la accesibilidad

2.3.3 Técnicas de evaluación de la usabilidad

Existen varias clasificaciones de estas técnicas. Algunos autores las clasifican según el tipo de implicados en la evaluación: usuarios, expertos y propuestas híbridas (Dix et al., 2003)(Freiberg and Baumeister, 2008). Otros autores las clasifican según su momento de aplicación en el ciclo de vida (Nielsen, 1993) o según el tipo de evaluación en técnicas de inspección, indagación y test (Granollers i Saltiveri et al., 2005). Se hace un breve repaso de estas técnicas agrupadas por tipo de evaluación.

En las técnicas de inspección, unos expertos analizan el grado de usabilidad de un sistema a partir de la inspección o examen detallado de su interfaz (Granollers i Saltiveri et al., 2005). Estas técnicas implican la participación de expertos de usabilidad y/o usuarios finales en el proceso de evaluación de la interfaz de usuario. Muchas técnicas de inspección pueden ser utilizados en las primeras fases de software de ciclo de vida, sobre los primeros prototipos obtenidos. Sin embargo, algunas de ellas también pueden ser utilizados para tratar la facilidad de uso, de todo el sistema, sobre el prototipo final. Algunos ejemplos de técnicas de inspección son la evaluación heurística (Nielsen and Molich, 1990), el recorrido cognitivo (Wharton et al., 1994), el recorrido de la usabilidad plural (Bias, 1994) y la inspección de estándares (Granollers i Saltiveri et al., 2005), que sirven como fuente de información para mejorar los elementos de la interfaz de usuario.

Las técnicas de indagación pueden ser utilizadas para obtener información acerca de los deseos de los usuarios, sus necesidades, su comprensión del sistema, etc. Esta información es esencial en las etapas tempranas del desarrollo. La implementación de este tipo de técnicas se lleva a cabo hablando con los usuarios, observándolos o haciéndoles responder a las preguntas verbalmente o por escrito. Entre otras, las técnicas de indagación incluyen cuestionarios, entrevista estructurada y no estructurada, grupos de discusión, grabaciones de la navegación o la observación de campo (Nielsen and Molich, 1990).

Por último, las técnicas de evaluación basadas en test analizan la interfaz cuando los usuarios llevan a cabo sus tareas. Normalmente, algunos usuarios representativos realizan tareas específicas utilizando el sistema (o prototipo), y los evaluadores observan y recogen los datos de interacción. Las técnicas de test más representativas comprenden el protocolo de pensar en voz alta (Clayton Lewis, 1982), la medida de las prestaciones (Dumas and Redish,

1993), el método del conductor (Nielsen, 1993), el test retrospectivo y la ordenación de tarjetas.

2.3.4 Integración de la disciplina HCI en la Ingeniería de Software

La integración de la disciplina HCI en la IS se ha venido debatiendo con intensidad en los últimos años. Con la finalidad de lograr una buena integración es necesario que la disciplina HCI y la IS hablen el mismo lenguaje y acuerden un cierto grado de formalidad en la descripción de IUs. La HCI se centra en el logro de características del diseño de IU como la facilidad de uso, la facilidad de aprendizaje, la efectividad, la satisfacción o la estética. Por otra parte, la Ingeniería de Software se centra en cómo trasladar los requisitos funcionales a un sistema ejecutable. Es conveniente, por lo tanto, que el proceso de integración sea controlado por personas que tengan un conocimiento profundo de IS y HCI.

Los modelos formales de la IS y la documentación excesiva (como las guías de estilo) de HCI son "demasiado caros" para el diseño de productos interactivos, en el sentido de que conllevan demasiado tiempo y dinero. La mayoría de los enfoques modernos de la IngUS y la IS no se oponen a la colaboración; al contrario, subrayan las coincidencias (Ferre et al., 2004) (Losada et al., 2009b).

El uso de UML en IS está muy extendido por lo que se han realizado numersos esfuerzos para integrar en él las técnicas y modelos típicos de HCI. Así, algunos autores han propuesto extender UML para incluir modelos de tareas (Nunes and e Cunha, 2000). Otros buscan la equivalencia entre los diagramas de UML, como los casos de uso y los diagramas de actividad, y modelos de HCI, como los modelos de tareas de usuarios (Markopoulos and Marijnissen, 2000). También, ha sido habitual dotar de mayor poder a los diagramas de actividad para representar conceptos de HCI como el orden o tipo de las tareas (Pihneiro da Silva, 2002).

Los diagramas de casos de uso son los únicos diagramas UML que proporcionan una visión del efecto del sistema en respuesta a la interacción con el usuario, y por lo tanto es la herramienta que se utiliza habitualmente para la realización del análisis de tareas. El diseño basado en los casos de uso proporciona una forma de especificar, a través de modelos, los requisitos del sistema, el comportamiento del usuario y la funcionalidad del sistema. Sin

embargo, este modelo no proporciona suficiente información sobre el usuario ni la usabilidad como para ser capaz de diseñar sistemas interactivos usables (Nunes and e Cunha, 2001). Los casos de uso esenciales son una variante de los casos de uso utilizados en el diseño centrado en el uso, que permiten enlazar la interfaz y su uso, relacionando el diseño de la interfaz con un propósito esencial del sistema. Estos autores introducen también la noción de los casos de uso esenciales estructurados para ser usados en el diseño de la IU, integrados con el resto del desarrollo.

Otra propuesta de integración viene del diseño basado en escenarios, que utiliza descripciones de uso del sistema. Una ventaja de esto es la mayor comprensión por los usuarios y desarrolladores que colaboran juntos (Rosson and Carroll, 2001).

Actualmente, el foco de atención son las metodologías ágiles y su integración con las disciplinas de la IngUS y mayoritariamente con el DCU.

2.3.4.1 Integración de DCU en las metodologías ágiles

La popularidad de las metodologías ágiles es creciente en la industria por lo que su integración con el DCU es una discusión habitual. Se comentan a continuación las semejanzas, diferencias y puntos de encuentro de ambas posturas. Después, se relatan los puntos problemáticos de la integración y se resumen algunas soluciones de integración de ambas.

Puntos de encuentro

Las metodologías ágiles parten de la premisa de que la implicación del cliente contribuye al éxito del producto final en cuanto a calidad, alcance, tiempo y coste (Chow and Cao, 2008). De igual manera, las prácticas de usabilidad implican al usuario para asegurarse de que los diseños se ajustan a sus necesidades y deseos. Además, al igual que los métodos ágiles, la mayoría de las propuestas centradas en el usuario son procesos iterativos en los que el diseño se valida numerosas veces con diferentes técnicas de usabilidad. En resumen, los principios y las prácticas ágiles son comparables a los del DCU en cuanto al diseño iterativo, participación de clientes vs. usuarios, resultados parciales vs. prototipado, user stories vs. escenarios, test vs. evaluación.

Sin embargo, el enfoque de ambas prácticas es diferentes. Las MA se centran en el proceso, esto es, se enfocan en el desarrollo de la funcionalidad que puede ser reconducida y cambiada en cualquier momento si el cliente lo desea. Se trata de un proceso que comienza con un desarrollo mínimo y evoluciona de una manera adaptable y flexible. Sin embargo, las prácticas de usabilidad se centran en el usuario y sus objetivos. Asumen que el usuario no puede explicar claramente lo que quiere por lo que debe ser ayudado mediante las técnicas apropiadas (Cooper et al., 2007). Suponen que gracias a los resultados en el empleo de estas técnicas, el diseño de la interacción será adecuado para el usuario y no necesitará hacer muchos cambios imprevistos.

En los desarrollos clásicos que utilizan DCU se realizan estudios de usabilidad lo más pronto posible, ya que se trabaja en todas las características del producto a la vez. Los ciclos de vida ágiles se caracterizan por entregas continuas de pequeños incrementos o variaciones. Cada entrega o versión ejecutable comprende un conjunto de características del producto final, y tiene su análisis de requisitos, diseño, implementación y comprobaciones. Las versiones ejecutables se crean en una iteración o sprint. En los proyectos ágiles y usables, cada iteración se centra en el desarrollo de unas pocas características, con lo que los estudios de usabilidad se realizan poco a poco.

En cuanto a los métodos, los del DCU son más informales que los de la IS y por lo tanto, más abiertos a la colaboración con los implicados. Memmel dice en relación con esto:"*With regard to the usage of informal artefacts like storyboards, scenarios or paper prototypes, HCI is generally more informal and therefore more open to collaborative design with stakeholders*" (Memmel et al., 2007).

En cuanto a los puntos de encuentro de ambas disciplinas, los métodos ágiles promocionan el prototipado low-fidelity, los diagramas de actividades o los casos de uso en la recogida inicial de los requisitos. Como lenguaje para la recogida de los requisitos, algunas investigaciones (Memmel et al., 2007) sugieren los prototipos y los escenarios, conocidos como historias de usuario en el desarrollo ágil. Estas técnicas tienden puentes entre DCU e IS.

Los escenarios pueden ser usados como una visión del sistema, al explicar la funcionalidad y el comportamiento interactivo, así como para la descripción de usuarios y sus tareas. Sirven para la evaluación y facilitan la creación (Rosson and Carroll, 2001).

Las metodologías ágiles reconocen los prototipos como un tipo de entrega pequeña, que puede ser cambiada continuamente. El DCU los utiliza para el diseño y evaluación de la interfaz de usuario. Las especificaciones visuales como los prototipos de alta fidelidad pueden garantizar que el sistema final cumple las expectativas del usuario en cuanto a la IU, y su comportamiento. El prototipado minimiza el riesgo de tomar decisiones de diseño equivocadas y alcanzar el diseño óptimo.

Una revisión con desarrolladores ágiles que integran DCU (Hussain et al., 2009) muestran que las técnicas del DCU más utilizadas en integración con MA son: prototipado low-fidelity, diseños conceptuales, estudios de observación de usuarios, evaluaciones de usabilidad con expertos, estudios de campo, técnica "personas", test de evaluación iterativos rápidos, y test de evaluación de usabilidad en laboratorios. La misma revisión concluye que los resultados de la integración en las diferentes propuestas muestran que la mayoría perciben que han añadido valor a sus procesos y equipos, que ha mejorado la calidad de la usabilidad y la calidad del producto desarrollado y que ha aumentado la satisfacción de los usuarios finales (Hussain et al., 2009).

Problemas en la Integración Metodologías Ágiles y DCU

Algunos autores destacan algunos problemas de la integración. Entre ellos:

- Las inconsistencias provocadas por los cambios en la IU. Los cambios en la arquitectura de software normalmente no tienen impacto en lo que el usuario ve, en la interfaz de usuario. Sin embargo, esto no es así si cambia la IU. Los cambios continuos en la IU debido a un diseño iterativo rápido, puede ocasionar conflictos con las expectativas del usuario y su entendimiento. Esto causa inconsistencias y finalmente insatisfacción (Constantine, 2002).
- La granularidad de los estudios de usabilidad. Según Sy, debe haber cambios en la granularidad de los estudios de usabilidad (Sy, 2007). Los ciclos en las iteraciones de corta duración permiten completar diseños pequeños. Debemos movernos hacia un número menor de test de usabilidad en cada entrega.
- Relacionado con esto, hay que valorar el coste de las evaluaciones. Un proceso de evaluación de usabilidad completo tradicional al comienzo de cada iteración puede ser

demasiado largo. Además, recrutar a los participantes y evaluadores, diseñar las pruebas y analizar los resultados de evaluación puede ser excesivamente largo y pesado en cada iteración. Según Memmel, las iteraciones del DCU pueden llevar mucho tiempo hasta llegar a una solución, por lo que la integración uno a uno de los procesos y técnicas de HCI es en general inapropiado para un curso ágil (Memmel et al., 2007).

- El particionamiento del diseño. Trocear un diseño en piezas pequeñas puede ser un problema desde el punto de vista del DCU. La evaluación de pequeñas entregas con los implicados no asegura que el sistema total provea un modelo conceptual, navegacional o de contenido consistente. Sy indica que, previamente, hay que definir objetivos de diseño, que se obtienen por observación, mediante estudios del contexto (Sy, 2007).
- Los cambios. Tradicionalmente, el trabajo de diseño y la evaluación de usabilidad se ha realizado antes de la implementación. La codificación comienza cuando el proceso iterativo del diseño de IU concluye, y esto incluye el diseño del prototipo, evaluación, análisis y rediseño. Si un fallo de diseño es encontrado posteriormente, deberá ser corregido en la siguiente versión del producto. Esto está en contradicción con los principios de las metodologías ágiles que permiten los cambios en cualquier momento del desarrollo ("Manifesto for Agile Software Development", 2001). Además el modelo ágil impulsa las iteraciones cortas y el mínimo diseño up-front.
- Los entregables no funcionales. El proceso iterativo en el DCU produce normalmente resultados en forma de diseños no funcionales y esto puede estar en disonancia con los principios ágiles. Sin embargo, Larman indica que el resultado de una iteración ágil no es necesariamente un software funcional (Larman, 2003).
- El tiempo destinado al diseño y documentación al principio del proyecto. Las metodologías ágiles prescriben un tiempo mínimo de diseño y documentación al principio del proyecto. En su lugar, se realiza el diseño necesario en cada iteración. Esto causa un problema al emplear técnicas de usabilidad tradicionales, en las que todo el trabajo de diseño debería estar listo antes de cualquier implementación.

Propuestas de Integración

Una discusión entre Kent Beck y Alan Cooper sobre la integración de XP y DCU mostró las discrepancias en ambas disciplinas ("Extreme Programming vs. Interaction Design," 2003). Sin embargo, concluye con esperanzas de integración. Dice Beck: "*To me, the shining city on the hill is to create a process that uses XP engineering and the story writing out of interaction design. This could create something that's really far more effective than either of those two things in isolation."*

La discusión sobre los enfoques ágiles al diseño de IU ha dado lugar a un movimiento en la comunidad HCI que empieza a reconsiderar sus ciclos de vida heavy-weight (Constantine, 2002) (Michael Gellner, 2004). Esto ha dado lugar al desarrollo de enfoques light-weight, como por ejemplo, *eXtreme Usability* (Federoff et al., 2008) o *Agile Human-Centered Software Engineering* (Memmel et al., 2007).

Existen varias revisiones con las propuestas más interesantes de integración entre métodos ágiles y DCU (Hussain et al., 2009) (Dayton and Barnum, 2009) (Hellmann et al., 2010). Las soluciones para lograr una buena integración son diversas. Algunas (Constantine and Lockwood, 2002) proponen utilizar modelos como los descritos por Ambler en su Agile Modeling (Ambler, 2002-13a). Otras, proponen un modelo que tienda puentes entre los MA y el DCU (Blomkvist, 2005) (Losada et al., 2009c). La mayoría sugieren integrar varias técnicas del campo del DCU como los estudios de campo, la técnica personas, los test de usabilidad, los prototipos de papel, las evaluaciones de usabilidad con expertos, etc, en sus procesos ágiles. Miller (Miller, 2006) y Sy (Sy, 2007) sugieren que los diseñadores de la interacción que entiendan y aceptan los conceptos ágiles son adecuados para el papel de clientes.

El proceso siguiente suele ser habitual en las propuestas de integración: El primer paso para mejorar la usabilidad del sistema suele consistir en crear prototipos de IU basados en el análisis de las necesidades del usuario y sus tareas. Estos prototipos comienzan por una versión low-fidelity y van siendo refinados hacia una versión high-fidelity. Después, normalmente, se evalúa el diseño del prototipo utilizando una variedad de técnicas. Finalmente, los resultados de la evaluación son analizados y el diseño es mejorado para solucionar algún fallo de usabilidad encontrado. El proceso comienza de nuevo con el diseño

actualizado, en un círculo iterativo hasta que se logra un diseño satisfactorio. Después, comienza la implementación.

A continuación, se describen algunas de las propuestas.

Constantine (Constantine and Lockwood, 2002) propone un enfoque light-weight llamada *Usage-Centered Engineering* que comparte la filosofía del Agile Modeling de Ambler (Ambler, 2002-13a). Constantine describe una arquitectura de la navegación y un esquema del diseño de la interacción basado en un modelado de tareas rápido y comprensible.

El trabajo de Sy "*Adapting Usability Investigations for Agile User-centered Desig*n" (Sy, 2007) integra test de usabilidad, entrevistas, e investigación contextual dentro de un marco ágil. Al principio de cada iteración se determina la historia de usuario de la siguiente entrega y las características (features) a desarrollar. Se describe cada característica y el criterio de aceptación que determina cuando está completa. Se realiza una investigación contextual en una etapa previa al desarrollo del proyecto o en paralelo con el test de usabilidad de la primera entrega. Durante las etapas de diseño se utilizan prototipos cuya evaluación proporciona la información necesaria para su implementación.

Otra propuesta de integración, CRUISER (Memmel et al., 2007), reune caractarísticas de XP, MA y DCU. Utiliza escenarios y prototipos en el proceso de diseño en el que hay una fuerte implicación de usuarios. Comienza con una recogida up-front inicial de los requisitos. Adopta las propuestas de Constantine, y para analizar las necesidades de los usuarios utilizan modelos de rol y modelos de tareas inducidos por los casos de uso esenciales. Sugieren guías de estilo light-weight que contienen patrones de IU. Todo ello da lugar a prototipos iniciales, incluso antes de la recogida total de los requisitos. La segunda fase es la fase conceptual cuyo objetivo es la generación de prototipos mas detallados e interactivos que permita una discusión con los implicados. De acuerdo con las MA, los prototipos deben ser fáciles de trabajar y, sobre todo, fáciles de producir y mantener. Comienza entonces la fase de construcción de test y la codificación. A partir de aquí, CRUISE se asemeja al planteamiento incremental e iterativo de XP; por ejemplo, recomiendan la integración de personal de HCI en el desarrollo pair programming. Durante las primeras iteraciones, las entregas son prototipos horizontales con un incremento en la profundidad de la funcionalidad. El último paso en el ciclo de vida CRUISER es el despliegue y la fase de producción. Mientras los usuarios están

trabajando con el sistema, pueden solicitarse nuevas funcionalidades o soluciones de usabilidad o emocionales, por lo que el ciclo de vida permite retroceder desde cualquier fase.

2.4 Desarrollo Dirigido por Modelos

El Desarrollo de Software Dirigido por Modelos (DDM - Model-Driven Development) ha desplazado la atención de los lenguajes de tercera generación hacia los modelos, especialmente hacia los expresados en UML (Sendall and Kozaczynski, 2003). El propósito del DDM es incrementar la productividad y reducir el tiempo de desarrollo, aumentando el nivel de abstracción y utilizando conceptos más cercanos al dominio del problema, en lugar de las herramientas de los lenguajes de programación. Esto es, los métodos de desarrollo de software se enfocan más a la creación y explotación de modelos del dominio que a los conceptos algorítmicos que subyacen bajo ellos. Los modelos pueden ser utilizados de manera horizontal para describir diferentes aspectos del sistema y también, de manera vertical para refinar un nivel alto de abstracción a uno más bajo.

Las tecnologías de la Ingeniería Dirigida por Modelos (MDE - Model-Driven Engineering) combinan lo siguiente (Schmidt, 2006):

- Los lenguajes de modelado específicos del dominio (DSML- Domain-Specific Modeling Languages) se describen utilizando metamodelos, que definen las relaciones entre los conceptos en un dominio y especifican con precisión la clave semántica y las restricciones asociadas con los conceptos de ese dominio. Tener elementos gráficos, que se refieren a un dominio familiar, asegura que los sistemas software se ajustan a las necesidades de usuario (Schmidt, 2006). Además el chequeo del modelo puede detectar y prevenir muchos errores en un momento temprano del desarrollo.
- Los motores de transformación y los generadores analizan ciertos aspectos de los modelos y luego sintetizan varios tipos de artefactos, como código fuente, simuladores, descripciones XML, o representaciones del modelo alternativas. Estas transformaciones ayudan a asegurar la consistencia entre las implementaciones de la aplicación y la información de análisis asociada con los requisitos funcionales y de calidad capturados por los modelos. El reto clave de la modelización es la transformación de estos modelos

de alto nivel en modelos específicos de la plataforma que puedan ser usados para generar código.

De esta forma, la MDE busca incrementar la productividad, maximizando la compatibilidad entre sistemas, simplificando el proceso de diseño y promocionando la comunicación entre individuos y equipos (The Object Management Group, 2003).

La Arquitectura Dirigida por Modelos (MDA- Model-Driven Architecture) forma parte de un estándar del OMG-Object Management Group, y es un framework para el DDM que define tres pasos en el proceso de desarrollo, que van desde el diseño de alto nivel hasta la realización de software (Kleppe et al., 2003):

1. Realización de un modelo del sistema software independiente de la tecnología de implementación. Este modelo se denomina Modelo Independiente de la Plataforma (PIM-Platform Independent Model).
2. Transformación del PIM en uno o más modelo específicos de la plataforma (PSM-Platform Specific Models), utilizando una estrategia de mapeo particular. Un PSM especifica un sistema utilizando unos constructores de implementación disponibles en una tecnología de implementación específica, por ejemplo la plataforma .Net.
3. Transforma El PSM en código.

Aunque MDA sea únicamente una posibilidad dentro del DDM, es la más visible. Otras visiones para DDM son: Domain-Oriented Programming (Thomas and Barry, 2003), las Factorías de Software (Greenfield, 2004) y el desarrollo ágil dirigido por modelos, que se explica en la siguiente sección.

2.4.1 Modelado Ágil

En palabras de Ambler, el desarrollo dirigido por el modelado ágil (AMDD) "*is a chaordic, practice-based methodology for effective modeling and documentation of software-based Systems*" (Ambler, 2004). Su objetivo es reducir al mínimo el modelado y documentación (Abrahamsson et al., 2003). El modelado ágil busca desarrollar software de manera ligera con principios, prácticas y metodologías que ayuden a mejorar el proceso de desarrollo.

Mejorar la comunicación entre los implicados se considera crucial, por lo que promueven el modelado con herramientas simples que permite la participación activa. Otro aspecto que se destaca es el uso de pizarras o papel para crear modelos en un primer momento.

El desarrollo dirigido por modelos ágiles es una propuesta iterativa que incluye un ciclo inicial con la conceptualización inicial de los requisitos (días) y el visualizado de la arquitectura inicial (días), y varios ciclos de desarrollo con el debate del modelo (minutos) e implementación (idealmente con TDD, tarda horas). Las revisiones son opcionales en todos los ciclos y pueden tardar horas. La diferencia con MDA es que en lugar de crear modelos extensos antes de escribir el código, se crean modelos ágiles que sean únicamente "suficientemente buenos" y se vuelve hacia atrás cuando sea necesario (Ambler, 2004). El esfuerzo de diseño está dirigido hacia las actividades de modelado y codificación, mientras que la mayoría de las otros enfoques inciden en los esfuerzos de implementación.

Por otra parte, AMDD se diferencia de otras técnicas como feature-driven development (FDD) o los desarrollos dirigidos por los casos de uso, en que no especifica el tipo de modelo a crear. Esto es, mientras que las features o los casos de uso son la base que sustenta el diseño, AMDD solo sugiere que se modele el artefacto necesario sin especificar cuál será.

2.4.2 Herramientas de modelado

Trabajar con varios modelos interrelacionados que describen un sistema software, requiere un esfuerzo significativo para asegurar la consistencia general. Por lo tanto, la automatización del chequeo de la consistencia entre los modelos mejora la productividad de los desarrolladores y la calidad de los modelos. Otras actividades que pueden automatizarse son el refinamiento, los modelos de ingeniería inversa, la generación de nuevas vistas, la aplicación de patrones y la refactorización. Esto se hace cogiendo uno o más modelos fuente y produciendo uno o más modelos objetivo, siguiendo un conjunto de reglas de transformación. Este proceso se denomina transformación de modelos. Esto es, para que la visión del DDM sea real, las herramientas deben ser capaces de soportar la automatización de las transformaciones de modelos. Estas herramientas deben ofrecer la posibilidad de aplicar transformaciones de modelos predefinidas y también, ofrecer un lenguaje que permita a usuarios avanzados definir sus propias transformaciones de modelos y ejecutarlas.

El crecimiento de UML y su estandarización, han influído para que se hayan extendido las herramientas basadas en UML: Poseidon, TogetherSoft, I-Logix's Rhapsody, Rational Software Modeler, ArgoUML, etc. Sin embargo, cada vez surgen más herramientas para aumentar la capacidad de diseño hacia el Modelado específico del dominio (DSM), en donde nuevos lenguajes de modelado pueden ser definidos para dominios particulares. Algunas de estas herramientas son: Generic Modeling Environment-GME (Ledeczi et al., 2001), MetaEdit+ ("MetaCase - Domain-Specific Modeling with MetaEdit+," n.d.) y Eclipse Modeling Framework-EMF (Steinberg et al., 2009) .

2.4.3 Entornos de desarrollo de Interfaces de Usuario Basados en Modelos

Mientras que en Ingeniería de Software MDA es la tendencia actual, en la IngUS es frecuente la propuesta de entornos de desarrollo de IU basados en modelos (MB-UIDE), que siguen una propuesta similar para diseñar, desarrollar y generar automáticamente IUs, a través de la creación de diversos modelos (Vanderdonckt, 2005).

Szekely define un modelo de interfaz como una especificación declarativa de alto nivel de un único aspecto de una IU, como por ejemplo su apariencia, características de layout o comportamiento dinámico (Szekely et al., 1996). En los MB-UIDEs aparecen algunos modelos coincidentes en las distintas propuestas, para describir la interfaz.

- ***Modelo del dominio***: Describe los objetos que manipula el usuario: sus características y comportamiento. Es habitual utilizar diagramas de clases en UML, modelos entidad-relación o especificaciones algebraicas.
- ***Modelo de usuario***: Recoge las características del usuario final: sus habilidades y limitaciones, sus preferencias y su ámbito de actuación.
- ***Modelo de tareas***: Describe las tareas que los usuarios realizan en la aplicación, y las subtareas que les lleva a realizar sus objetivos, así como la relación entre ellas. La descomposición jerárquica de tareas es la propuesta habitual (Paterno, 2000) (Annett and Duncan, 1967), pero también se utilizan otras propuestas como UAN (Hartson et al., 1990) o Task Knowledge Structures (Johnson and Johnson, 1991).

- ***Modelo de diálogo***: Describe las posibles tareas que los usuarios pueden realizar en la aplicación, junto con la respuestas del sistema, en la consecución de los objetivos perseguidos.
- ***Modelo de presentación***: Describe los aspectos visuales de la interfaz. Además de las propuestas de extensión de UML (Pinheiro da Silva and Paton, 2003), HTML y, sobretodo XML, son las tendencias más fuertes. La falta de un estándar de un lenguaje común que sirva para describir todos los tipos de dispositivos y sus diferentes alcances, hace que hayan surgido distintas propuestas, como UIML (Abrams et al., 1999), XIML ("XIML.org," n.d.) o UsiXML (Limbourg et al., 2005), entre otros.

2.4.4 Ventajas e Inconvenientes del DDM

El DDM busca aumentar la calidad del software, reducir su complejidad, y mejorar la reutilización permitiendo a los desarrolladores trabajar en niveles altos de abstracción, ignorando los detalles innecesarios (Hailpern and Tarr, 2006). Sin embargo, tiene algunos inconvenientes:

- ***Redundancia***: Se dan múltiples representaciones de los artefactos de un proceso de desarrollo de software, que representan las diferentes vistas o niveles de abstracción del mismo concepto.
- ***Problemas de consistencia***: Cuando los cambios suceden a niveles bajos de abstracción, por ejemplo en el código, es difícil trasladar este cambio a los modelos. Esto es debido a que la inferencia de semántica desde las abstracciones de bajo nivel a las de alto nivel, es mucho más difícil que generar abstracciones de bajo nivel a partir de las de alto nivel. La relación entre los modelos también debe ser consistente. Esto es, si ocurre un cambio en un modelo (vista), este cambio se debería reflejar en otro modelo (vista) interrelacionado.
- ***Aumento de complejidad***. En algunos casos puede aumentar la complejidad en el proceso de desarrollo en lugar de reducirla. Algunos modelos, con mucho detalle, son difíciles de realizar y mantener (Bell, 2004).

- ***La tendencia MDA***. Según Ambler "*The MDA is a very sophisticated and complex approach to modeling, one that only a very small minority of elite developers will be able to adopt*" (Ambler, 2004).
- ***Requiere más expertos***. Cada tipo de modelo requiere un tipo particular de habilidades para crearlo y manterlo. Es necesario disponer también, de desarrolladores expertos para entender las relaciones entre los modelos y el impacto de los cambios en ellos. Además, dada la necesidad de tecnologías de transformación en DDM, los desarrolladores deben tener fuerte conocimiento en las notaciones de transformación, y éstas pueden ser complejas.
- ***Los lenguajes de DDM***. En particular, la estadarización de UML 2.0 tiene numerosas críticas. Primero, con el objetivo de atender a todas las necesidades se ha vuelto enorme y complejo (Thomas and Barry, 2003). Segundo, algunas construcciones en UML 2.0 no tienen una semántica clara, como por ejmplo los casos de uso. Esto complica la automatización de las herramientas de transformación.

Áreas y situaciones adecuadas para su aplicación

Es adecuado para el desarrollo de sofware industrial, en proyectos grandes y distribuídos (Hailpern and Tarr, 2006).

2.5 Resumen

Este capítulo hace una revisión de los ciclos de vida en Ingeniería de Software, desde los primeros modelos propuestos en cascada hasta el momento actual en el que imperan las metodologías ágiles. Esta evolución histórica se acompaña con las ventajas, inconvenientes y áreas de aplicación de cada enfoque que indican los autores del área.

A continuación, el capítulo se centra en la Ingeniería de la Usabilidad. Se hace un repaso a la evolución de los ciclos de vida, especialmente en el Diseño Centrado en el Usuario, y también se explican brevemente las técnicas habituales empleadas para la evaluación de la usabilidad. Después, se hace un repaso a las propuestas y dificultades de integración de la

disciplina HCI en la Ingeniería de Software y especialmente, del DCU en las metodologías ágiles, por ser un objetivo de esta tesis.

El Desarrollo Dirigido por Modelos es el tercer punto tratado en este capítulo. Se hace un resumen de sus características, tipos y herramientas, y también se tratan los entornos de desarrollo de IUs basados en modelos. Finalmente, se indican las ventajas, inconvenientes y áreas de aplicación que destacan los autores de este área.

En el siguiente capítulo, se describe en profundidad InterMod, la metodología que propone esta tesis y que aglutina las ventajas de las MA, la IngUS y los DDM, cubriendo las carencias de estas tres filosofía cuando se aplican solas.

CAPÍTULO 3. InterMod, una Metodología Ágil para el Desarrollo de Aplicaciones Interactivas

3.1 Introducción

El mercado software es cada vez más competitivo en Internet y dado que la calidad de una aplicación es un factor determinante en su aceptación o rechazo, las exigencias de calidad son crecientes. No es suficiente con que un producto sea técnicamente excelente, sino que también se tienen en cuenta otros criterios de calidad, como la facilidad de uso o el cumplimiento de las expectativas del usuario final.

InterMod es una metodología ágil para el desarrollo de software interactivo de calidad (Losada et al., 2009b) (Losada et al., 2011a) (Losada et al., 2013a). Este enfoque integra tres filosofías: los métodos ágiles (MA), el Diseño Centrado en el Usuario (DCU) y el Desarrollo Dirigido por Modelos (DDM), y aunque es adecuado para realizar aplicaciones web, su utilidad no está restringida a este área.

InterMod organiza y desarrolla un proyecto software mediante una serie de *iteraciones*. En cada iteración, el trabajo se distribuye de acuerdo a diferentes *actividades* para alcanzar los *Objetivos de Usuario*. Estas actividades pueden llevarse a cabo en paralelo por diferentes equipos. Cada actividad está dirigida por *modelos* que deben ser validados por un equipo multidisciplinar compuesto por gestores, clientes, desarrolladores y usuarios finales.

En las siguientes secciones se describen los aspectos característicos de esta metodología: los Objetivos de Usuario (User Objectives-UOs), las actividades que permitirán desarrollar la aplicación a través de los UOs, y el proceso ágil de desarrollo de InterMod. Además, se incluyen las reglas de planificación de actividades que ayudan al equipo gestor a la toma de decisiones según la perspectiva DCU de InterMod.

3.2 Objetivos de Usuario, la guía del proceso ágil en InterMod

En InterMod, un proyecto se realiza de forma iterativa, cohesionando desarrollos y efectuando modificaciones o ampliaciones sobre desarrollos realizados total o parcialmente.

Los desarrollos nuevos pueden ser solicitados directamente por el usuario o bien, inducidos por estrategias del proceso. Estos desarrollos pasan por procesos de evaluación y validación por parte de los implicados (incluído el usuario final). InterMod guía el proceso ágil en base a estos nuevos desarrollos que se recogen durante todo el proyecto y se materializan en lo que se denomina Objetivos de Usuario-User Objectives (UOs) (Losada et al., 2011b) (Losada et al., 2011c).

3.2.1 Definición de Objetivo de Usuario

Un Objetivo de Usuario (UO) expresa un deseo de usuario como, por ejemplo, "*comprar una camiseta*" o "*reservar una sala de reuniones en un lugar de trabajo*" .

Un UO puede coincidir con uno o más requisitos funcionales (casos de uso) y/o requisitos no funcionales. Los requisitos no funcionales normalmente se refieren a la eficiencia, la ergonomía, la seguridad o a limitaciones debidas a los tipos de usuario o dispositivo. Por

ejemplo, en una solicitud de compra y venta de apartamentos, el usuario no quiere únicamente "*buscar un apartamento*", sino que desea "*buscar rápidamente un apartamento que me guste, preguntar por el precio y comparar este precio con el precio de mi apartamento*" o "*preguntar el precio de un apartamento y el crédito posible usando mi teléfono móvil* ". Estos UOs especializan e interpretan agrupadamente varios casos de uso genéricos como: "*buscar un apartamento*", "*preguntar el precio*", "*valorar un apartamento*", "*consultar información de créditos*". Las características subrayadas: *rápidamente*, *que me guste* y *usando mi teléfono móvil,* son requisitos no funcionales asociados a un UO.

Formalmente, los UOs se etiquetan con números consecutivos <i>, un nombre y una breve descripción del deseo de usuario en primera persona según el siguiente formato:

{UOi-*nombre*: breve descripción }. Por ejemplo,

> UO15- *Comprar un producto en oferta desde casa*: Quiero entrar desde mi casa en la web de la tienda, distinguir los productos en oferta y elegir uno para su compra.

Los UOs continúan vivos a lo largo del proyecto hasta la entrega final, y pasan por procesos de desarrollo distintos: especificación, presentación e implementación, y por procesos de integración y ampliación. Las variaciones sobre un UO implican su reactivación, bien para corregir errores o para añadir pequeños cambios. Es decir, la identidad del UO es la misma a lo largo del proyecto, pero puede cambiar su especificación, presentación y/o implementación.

Durante el desarrollo del proyecto y una vez descubierto y formalizado un UO, éste se materializa con una especificación en términos de la interfaz mediante la descripción de los comportamientos del usuario (acciones a realizar en la interfaz) y del Sistema (reacciones de la aplicación en la interfaz). Estas descripciones son sólo una parte de los requisitos funcionales, ya que sólo indican las reacciones en la interfaz. Es decir, no incluyen las reacciones de control de la aplicación, como "registrar la compra" o "guardar la línea de pedido en la Base de Datos". Además, en la especificación se debe indicar el comportamiento de la aplicación sobre la interfaz (navegación) en las situaciones alternativas o erróneas. La especificación inicial se completa con algunos aspectos gráficos que ayudarán a configurar un prototipo simple. Esto sirve a los desarrolladores para tener una primera visión global del

aspecto y comportamiento del UO, pero sin concretar las características precisas de la interfaz y la lógica de negocio.

La especificación de un UO tiene ciertas similitudes con los Casos de Uso en cuanto que formalizan los requisitos funcionales. Los Casos de Uso esenciales, al igual que los Casos de Uso tradicionales o concretos, son técnicas habituales con el propósito de formalización. A continuación, se muestran las diferencias entre Casos de Uso Concretos, Casos de Uso Esenciales y UOs en un mismo ejemplo.

3.2.1.1 Comparativa Caso de Uso vs.Objetivo de Usuario

El ejemplo consiste en especificar la funcionalidad "Dispensar Bebida" de una aplicación interactiva en la que una máquina dispensadora de bebidas permite a un usuario comprar la bebida deseada.

Caso de Uso Concreto

Los casos de uso fueron introducidos por Ivar Jacobson en 1986 como una manera simple y útil de describir los requisitos funcionales (Jacobson, 1992).

Los casos de uso son descripciones exhaustivas de los requisitos funcionales del sistema, en forma transaccional y plasmadas en acciones de usuario de bajo nivel en conjunción con las reacciones del Sistema, y que habitualmente se representan de forma textual. En un caso de uso puede haber varios tipos de actores. Un actor es una entidad con comportamiento: una persona identificada por un rol (por ejemplo un cliente), un sistema informatizado (por ejemplo el sistema de contabilidad de una empresa) o una organización o entidad (por ejemplo un cajero).

En el caso de uso "Dispensar Bebida" (Tabla 3.1) se definen las acciones del Usuario y del Sistema (la máquina dispensadora) que producen el efecto deseado "El Sistema dispensa la bebida…" , y además se incluyen separadamente las acciones que producen los cursos alternativos. Se mencionan los elementos necesarios que participan en el caso de uso, tanto del dispositivo ("ranura", "teclado", etc) como de la interfaz concreta ("menú", "selecciona", etc). Las acciones del Sistema son acciones internas de la lógica de negocio ("verifica", "detecta", etc) y acciones externas ("solicita cantidad", "devuelve la tarjeta", etc).

Tabla 3.1 Caso de uso "Dispensar Bebida"

Acciones del Usuario	Reacciones del Sistema
1. El Cliente inserta la tarjeta de crédito en la ranura	2. El Sistema solicita el PIN
3. El Cliente teclea 4 dígitos usando el teclado	4. El Sistema verifica la identidad del usuario 5. El Sistema solicita la selección del producto de un menú de opciones
6. El Cliente selecciona el producto	7. El Sistema solicita cantidad utilizando un menú
8. El cliente selecciona la cantidad	9. El Sistema verifica disponibilidad producto 10. El Sistema verifica saldo 11. El Sistema devuelve la tarjeta a través de la ranura. 12. El Sistema dispensa la bebida por la zona de dispensa de productos.

CURSOS ALTERNATIVOS:

4.1. El Sistema detecta PIN erróneo. Paso 2
9.1. El Sistema verifica que no hay cantidad suficiente.
    9.1.1. Opción 1: Repetir selección. Paso 5.
    9.1.2. Opción 2: Cancelar. Fin.
10.1. El Sistema verifica que no hay saldo suficiente.
    10.1.1. Cancelar. Fin.

Caso de Uso Esencial

Los casos de uso esenciales forman parte del Diseño Centrado en el Uso (Constantine and Lockwood, 1999). Son narraciones estructuradas, expresadas en el lenguaje del dominio de la aplicación y de los usuarios, que describen una tarea de forma simplificada, generalizada, abstracta, libre de tecnología e independiente de la implementación, definida desde el punto de vista del usuario, dentro de un rol o roles en su relación con el Sistema, y que recoge el propósito de la interacción (Constantine, 1995). Un caso de uso esencial, a diferencia de los casos de uso concretos, no contiene suposiciones sobre el tipo de interfaz ni la tecnología que se usará. Se centra en lo que un usuario quiere hacer y en las responsabilidades del Sistema, más que en cómo lo logrará. En esta especificación (Tabla 3.2), similar a la anterior, las

acciones del Usuario y del Sistema (máquina dispensadora) se abstraen de la tecnología (tarjeta, ranura, etc) y de la interfaz concreta (menú, selección, etc).

Tabla 3.2 Caso de uso esencial "Dispensar Bebida"

Acciones del Usuario	**Reacciones del Sistema**
1. El Cliente se identifica	2. El Sistema valida la identidad del usuario 3. El Sistema solicita indicación de producto
4. El Cliente indica el producto	5. El Sistema valida producto
6. El Cliente indica la cantidad	7. El Sistema valida cantidad 8. El Sistema solicita Pago
9. El Cliente efectúa Pago	10. El Sistema valida Pago 11. El Sistema valida disponibilidad 12. El Sistema dispensa bebidas

CURSOS ALTERNATIVOS:

2.1. El Sistema deniega el paso. Paso 1.
5.1. El Sistema verifica que el nombre o código de producto no es correcto
    5.1.1. Paso 4.
7.1. El Sistema verifica que la cantidad no es una cifra correcta.
    7.1.1 Paso 6.
10.1. El Sistema verifica que el Pago es incorrecto
    10.1.1. Opción 1: Paso 9
    10.1.2. Opción 2: Cancelar. Fin.
11.1. El Sistema verifica que no hay producto.
    11.1.1. Opción 1: Paso 3.
    11.1.2. Opción 2: Cancelar. Fin.

Objetivo de Usuario

En un Objetivo de Usuario el foco de la especificación es el usuario final, y a él va dirigida la interacción. Los actores son siempre dos: el usuario y el Sistema como identidad abstracta que agrupa el comportamiento de la aplicación en la IU. Una vez detectado un UO, el primer paso en la vida de un UO es su formalización. Por ejemplo:

UO5-*Dispensar bebida*: El usuario quiere obtener una bebida en una máquina dispensadora de bebidas.

La especificación del UO (Tabla 3.3) describe únicamente las acciones del Usuario y del Sistema a nivel externo (comunicación directa Usuario-Sistema); esto es, describe los efectos de esas acciones en la interfaz de usuario: "El Cliente se identifica", "El Sistema responde", "El Sistema solicita"…

Tabla 3.3 Especificación de UO5 - Dispensar Bebida

Acciones del Usuario	**Reacciones del Sistema**
1. (*Nueva ventana*) El Cliente se identifica	2.a. El Sistema responde: "Identidad del usuario correcta". Paso 3. 2.b. El Sistema responde "Identidad Incorrecta". Paso 1. 3. (*Nueva ventana*)El Sistema solicita indicación de producto.
4. El Cliente indica el producto	5a. El Sistema responde: "Nombre de producto Correcto" . Paso 6. 5b. El Sistema responde: "Nombre de producto Incorrecto" . Paso 4. 6. El Sistema solicita indicación de Cantidad
7. El cliente indica la cantidad	8a. El Sistema responde: "Cantidad correcta y disponible". Paso 9. 8.b. El Sistema responde: "Cantidad no correcta". Paso 7. 8c. El Sistema responde:"Cantidad no disponible". Paso 4. 9. (Nueva ventana)El Sistema indica total a pagar y solicita Pago
10. El cliente efectúa Pago	11a. El Sistema responde "Pago correcto" y dispensa bebida. Fin. 11b. El Sistema responde "Pago insuficiente". Paso 10. 11c. El Sistema responde: "No se dispone de cambios". Devuelve Pago y Fin.

Por otra parte, al igual que los casos de uso esenciales, la especificación inicial de los UOs se abstraen de la tecnología y de la interfaz concreta (menús, colores, tipos de botones, etc). Sin embargo incluyen algunos elementos mínimos para la presentación en la interfaz de las

acciones (en el ejemplo "nueva ventana") que son necesarios para generar prototipos de bajo nivel que van a permitir evaluar mejor estas especificaciones con los usuarios.

Además, quedan integradas en la propia especificación las acciones alternativas del Sistema (2a-2b, por ejemplo) y las desviaciones ("Paso 3", por ejemplo), que representan la semántica de la aplicación (requisitos funcionales) a tener en cuenta posteriormente en la implementación.

Discusión sobre Especificaciones de UOs, Casos de Uso, Casos de Uso Esenciales y otros

En las tres formas expuestas: Casos de Uso, Casos de Uso Esenciales y Objetivos de Usuario, se observan diferencias en la descripción de las acciones del usuario y del sistema. Así, Larry Constantine y Lucy Lockwood observan excesos en los casos de uso concretos, en cuanto a las descripciones precisas sobre la forma en que se construirá la interfaz de usuario. Esta precisión obliga a tomar decisiones de diseño tempranas, e incluir estas decisiones en las especificaciones dificulta su modificación más tarde (Constantine, 1995). Alistair Cockburn razona que desde el punto de vista metodológico o de proceso, el diseño de la IU es posterior a la especificación concreta de la funcionalidad: "*The design of the UI is likely to change too often for such writings to be used as contractual requirements and as system requirements*" (Cockburn, A, 1997). Normalmente, un equipo de diseñadores lee la especificación y ofrece diferentes maneras de recoger y presentar la información. Por esta razón, otros autores prefieren trabajar a nivel de interacción semántica, describiendo sólo la información que interactúa, sin describir la secuencia o la naturaleza de la interacción (sin describir el diálogo preciso). Esta es la idea de Larry Constantine con sus "Essential Use Cases".

En cualquiera de los dos casos de uso, concretos y esenciales, el foco de la especificación está más en las acciones que debe realizar el Sistema que en la comunicación que debe tener con el Usuario, derive ésta o no de sus acciones internas. La especificación de los Objetivos de Usuario, sin embargo, permite centrar la atención en los aspectos de la interacción Usuario-Sistema a un nivel de usuario, de tal modo que la evaluación y verificación de dicha especificación muestre a los usuarios únicamente el comportamiento futuro de la aplicación en la interfaz. Los aspectos internos del comportamiento del Sistema no se especifican a este

nivel ya que no son relevantes para el usuario y dificultan su evaluación. Por ello, los Objetivos de Usuario que propone InterMod ayudan a organizar un proyecto definiendo las necesidades funcionales y restricciones debidas al Usuario y/o al Sistema, y además llevan implícitos los requisitos de usabilidad por su recogida y evaluación.

Existen también diferencias entre los UOs y los Goals, empleados en varios trabajos sobre análisis de requisitos. (Anton, 1996) (van Lamsweerde, 2004) Tanto los UOs como los Goals comparten algunas características comunes: pueden referirse a propiedades funcionales o no funcionales y permanecen activos durante toda la vida del proyecto como guía del mismo. Sin embargo, sus diferencias son notables en cuanto a su orientación y evolución en el desarrollo.

Los *Goals* son mecanismos lógicos que sirven para identificar y justificar los requisitos de software. El análisis necesario para identificar los *Goals,* implica la exploración de documentación seguida de un proceso de organización y clasificación. El análisis de captación de UOs, sin embargo, parte de los deseos directos de usuarios finales, y son evaluados con ellos de forma temprana para su mejor adquisición. Esto es, en palabras de Antón, "*Goals are high level objectives of the business, organization or system*" (Anton, 1996), y, de forma similar, para Lamsweerde "*A goal is an objective the system under consideration should achieve*" (van Lamsweerde, 2001); mientras que los UOs son objetivos abocados a cumplir deseos directos del usuario final.

Los UOs se formalizan mediante la descripción del comportamiento de los usuarios (acciones para completar) y los comportamientos del Sistema (reacción de la aplicación), en relación a la interfaz. Estas descripciones incluyen sólo una parte de los requisitos funcionales, ya que sólo se indican las reacciones del sistema en la interfaz. Es decir, no incluyen las reacciones de control de la aplicación, como "comprobar la compra" o "guardar la línea de pedido". Los UOs son útiles para organizar la interfaz y también para ayudar a descubrir y formalizar los casos de uso en etapas posteriores.

En cuanto a la evolución de los *Goals,* ésta se refiere a la forma en que éstos cambian desde el momento en que son identificados hasta el momento en que son operativos en la especificación del sistema. Esto encaja dentro de una perspectiva up-front en la que se contemplan todos los requisitos desde un principio. Desde la perspectiva ágil que comparte

InterMod, los requisitos, incluso después de su implementación y validación, pueden ser revisados y actualizados.

3.2.2 Tipos de Objetivos de Usuario

Los Objetivos de Usuario pueden ser:

- **UO Directo** o deseo directo del usuario
- **UO Indirecto** o necesidad interna del proceso de desarrollo, derivado de UOs Directos o Indirectos.
- **UO Incremento** o incremento de funcionalidad asociado a un UO Directo o Indirecto
- **UO Incrementado** o ampliación de un UO Directo o Indirecto con uno o más UO Incremento
- **UO Reutilizado** o UO realizado previamente en el mismo o en otro proyecto.

Se pueden observar dos tipos de clasificaciones de UOs; la primera, según la fuente que provoca su creación y la segunda, según la identidad o autonomía del UO. Según la primera, si la fuente es el usuario son UOs Directos, mientras que si lo es los desarrolladores, son UOs Indirectos. Según la segunda clasificación, hay UOs de identidad completa y ficticia. La identidad del UO es completa habitualmente, es decir, se trata de un deseo o necesidad autocontenida que podría ser un entregable en el sentido de las metodologías ágiles. Sin embargo, se consideran de identidad ficticia aquéllos que expresan de forma parcial y no autocontenida un incremento asociado a un UO existente, esto es, los llamados UOs Incremento. Estos UOs Incremento cuando se asocian a un UO de identidad completa dan lugar a un UO Incrementado, también con identidad completa. Finalmente, los UOs Reutilizados son aquéllos UOs de identidad completa, que han sido definidos con anterioridad en el proyecto actual o en otro proyecto.

A continuación, se describen con detalle los tipos de UOs, a los que se dota de una representación gráfica diferencial para facilitar su visualización y distinción en la documentación del proyecto. Para explicar los diferentes aspectos de los UOs y visualizar el

progreso del proyecto, se hace uso de fragmentos de un mismo proyecto denominado *WebButcher*, cuya evolución completa por UOs creados, en sus 8 primeras iteraciones, se muestra en la Figura 3.11. Este proyecto se centra en el desarrollo de una carnicería web que incluye además consejos y recetas, adaptada a las necesidades de los usuarios finales.

3.2.2.1 UO Directo

El UO Directo es un objetivo a desarrollar que surge directamente del usuario final. El usuario indica una necesidad que es valorada con técnicas de usabilidad y criterios de prioridad. Una vez aceptada por los implicados en el proyecto, se formaliza en un UO para su desarrollo y validación posterior. Los UOs Directos aparecen gráficamente con una circunferencia de trazo grueso y un número en su interior que representa la etiqueta del UO y da idea sobre su orden de aparición. Por ejemplo, la representación para UO1, queda representada como se muestra en la Figura 3.1.

Figura 3.1 Representación gráfica del UO Directo UO1

UO1- *Comprar filetes de vacuno cortados a mi gusto:* Quiero poder elegir el tipo de carne y el tipo de corte que quiero comprar. Incluso, quiero poder comprar unos finos y otros gruesos. Si no me gustan, quiero poder devolverlos. Quiero saber en todo momento cuándo vendrá mi pedido.

Con el fin de determinar los UOs Directos, es fundamental contar con la colaboración de un grupo de futuros usuarios de la aplicación. InterMod recomienda obtener los UOs Directos a partir de la información recogida por medio de métodos de indagación de la IngUS, siguiendo los criterios de calidad incluidos en los últimos estándares de usabilidad (“ISO/IEC 25010:2011,” , 2011).

Los métodos de indagación (Granollers i Saltiveri et al., 2005) engloban técnicas de aproximación contextual como estudios de campo o etnográficos, técnicas de aproximación por grupos como grupos de discusión dirigidos, y de aproximación individual como encuestas, cuestionarios y entrevistas. Cuanto más precisas sean recogidas las necesidades de la audiencia, más adaptado será el diseño y más satisfactoria la experiencia del usuario final.

En algunos casos la iniciativa de incorporar un UO Directo no surge en primera instancia del usuario final, sino de algún otro miembro implicado en el proyecto y es sugerido al usuario final (habitualmente en las entrevistas). Si finalmente es solicitado por el usuario, se considera que es un deseo del usuario y su desarrollo transcurre como un UO Directo.

3.2.2.2 UO Indirecto

Un UO Indirecto es un UO que surge por necesidades internas del desarrollo. Estas necesidades pueden provocar una división o una fusión de UOs. En caso de división daría lugar a varios UOs Indirectos división y en caso de fusión se obtendría un UO Indirecto fusión.

UO Indirecto división

Los implicados en el proyecto pueden decidir dividir un UO en varios UOs Indirectos división, que podrán ser realizados por un mismo equipo o por diferentes equipos de desarrollo, en una o en diferentes iteraciones. La división de UOs permite agilizar su desarrollo, bien por su complejidad ("*divide y vencerás"*) o bien por su urgencia de entrega (trabajo en paralelo). Los UOs Indirectos provocados por una división se dibujan con una circunferencia de trazo suave discontinuo Por ejemplo, en la Figura 3.2 se decide dividir el objetivo directo UO1 en los UOs Indirectos división {UO4, UO5, UO6, UO7}[1].

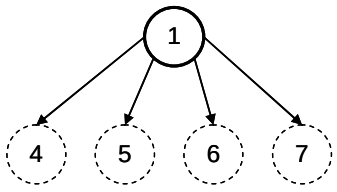


Figura 3.2 División de UOs

UO4- *Comprar filetes al corte:* Realizar la compra con opciones de tipo de carne (parte de la vaca, tipo de corte, indicaciones, cantidad).

UO5- *Realizar pago:…*

UO6*: Ver estado del pedido:…*

UO7*: Cambios y Devoluciones*

[1] La numeración de estos UOs se ha realizado después de haber formalizado otros UOs anteriormente: {UO2, UO3} (Ver visión completa en la Figura 3.11)

UO Indirecto fusión

Durante el ciclo de vida del proyecto, puede ocurrir que varios UOs se fusionen para desarrollarse en un único UO Indirecto fusión. Esto puede deberse a integraciones de funcionalidades abocadas a la consecución de la aplicación global, o bien a que se considera que las dependencias de los UOs son muy fuertes por lo que es es mejor su desarrollo integrado

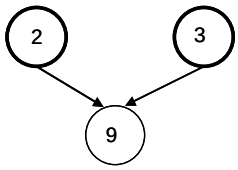


Figura 3.3 Fusión de UOs

UO2: *Recetas:* Quiero escribir recetas o comentarios sobre las recetas, de una manera fácil.

UO3: *Recomendaciones:* Quiero poder recomendaciones de recetas, de una manera fácil.

UO9: *Recetas o Recomendaciones*:. Escribir recetas o comentarios sobre las recetas, o recomendaciones de recetas, de una manera fácil

Los UOs Indirectos producidos por una fusión, se dibujan con una circunferencia de trazo suave. Así ocurre, por ejemplo, con la fusión de los UOs Directos 2 y 3, que se unen para su desarrollo conjunto en el UO Indirecto fusión UO9 (Figura 3.3).

Particularidad: UOs Directos tras sucesivas divisiones y/o fusiones

No todos los UOs producto de una división o una fusión, son necesariamente UOs Indirectos. En el transcurso de un proyecto los UOs evolucionan, se crean, se dividen y se fusionan. Algunos de estos UOs división o fusión pueden ser UOs Directos creados anteriormente (ya formalizados). Por ejemplo, un caso habitual es el de un UO que se divide, y tras varios procesos de división y fusión, se vuelven a fusionar en el UO inicial. Este es el caso visualizado en la Figura 3.4. En este ejemplo, UO1 se divide en UO4, UO5, UO6 y UO7. Tras la fusión de UO6 y UO7 se crea el UO8 Indirecto fusión. Después, se llega a UO1 (Directo) fusionando UO4, UO5 y UO8, y continúa así su desarrollo.

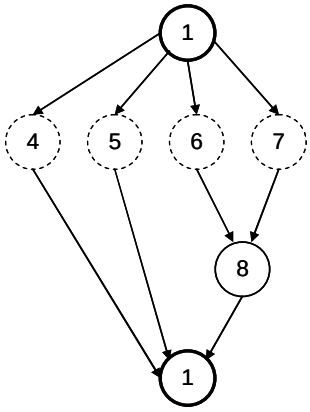


Figura 3.4 UO Directo surgido tras una fusión

UO8- *Gestión del pedido:* Ver el estado del pedido en cualquier momento y proceder a cambios y devoluciones

3.2.2.3 UO Incremento

Un UO Incremento se formaliza tras una petición de incremento de funcionalidad, que se efectuará sobre un UO de identidad completa. En este caso, el usuario o el equipo de implicados sugiere añadir nuevas funcionalidades o características no funcionales a un UO existente. Esto supondrá necesariamente realizar ampliaciones a partir del UO implicado. Como consecuencia, surge un nuevo UO llamado "UO Incremento" al que se le dota de identidad. Sin embargo, su desarrollo está ligado necesariamente al del UO que incrementa y, por lo tanto, su identidad se considera ficticia. La finalidad de su formalización en un UO es la de documentar explícitamente ese nuevo deseo. El UO incremento se representa con una semicircunferencia de trazo marcadamente fuerte (Figura 3.5).

Figura 3.5 UO Incremento

UO12- *Envasado al vacío*: Quiero solicitar que mi pedido se envase al vacío

3.2.2.4 UO Incrementado

Un UO Incrementado es un UO obtenido mediante la ampliación de un UO de identidad completa con uno o más UOs Incremento. Esto es, el UO incrementado tiene una identidad completa derivada del UO al que se le incorporan las nuevas necesidades no autónomas funcionales y/o no funcionales de los UOs Incremento. El UO Incrementado puede ser

Directo o Indirecto, si supone un incremento sobre un UO Directo o Indirecto, respectivamente.

UO Incrementado Directo

Un UO Directo incrementado con uno o varios UOs Incremento, da lugar a un UO Incrementado Directo. Por ejemplo, la Figura 3.6 muestra cómo el UO Directo UO1 se extiende con el UO de incremento UO12, y surge el UO Incrementado Directo UO13. El nuevo UO está representado con una circunferencia de trazo grueso recubierta parcialmente por una semicircunferencia.

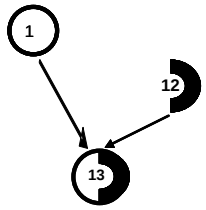


Figura 3.6 UO Incrementado Directo

UO1- *Comprar filetes de vacuno cortados a mi gusto*: Quiero…

UO12- *Envasado al vacío*:...

UO13- *Comprar filetes de vacuno cortados a mi gusto con la posibilidad de envasado al vacío*:…

UO Incrementado Indirecto

Un UO Indirecto incrementado con uno o varios UOs Incremento, da lugar a un UO Incrementado Indirecto. En la Figura 3.7 se observa otra posibilidad de obtención de UO13, en la que es el UO Incrementado de UO4 (Indirecto) con UO12 (UO Incremento). Se representa con una circunferencia de trazo suave, recubierta parcialmente por una semicircunferencia.

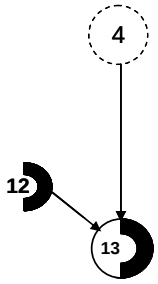


Figura 3.7 UO Incrementado Indirecto

UO13- Comprar filetes al corte con la posibilidad de envasado al vacío:…

3.2.2.5 UO Reutilizado

Un UO Reutilizado es un UO creado y evaluado, total o parcialmente, en otro proyecto o en el actual, que puede ser reutilizado. Los UOs Reutilizados pueden ser Directos o Indirectos. Se dibujan con dos circunferencias, siendo la interna de trazo grueso si es Directo y las dos de trazo fino si es Indirecto, tal y como se muestra en la Figura 3.8 y la Figura 3.9, respectivamente.

Figura 3.8 UO Reutilizado Directo

Figura 3.9 UO Reutilizado Indirecto

La Figura 3.10 muestra la incorporación del UO Reutilizado Directo UO10 en *WebButcher*.

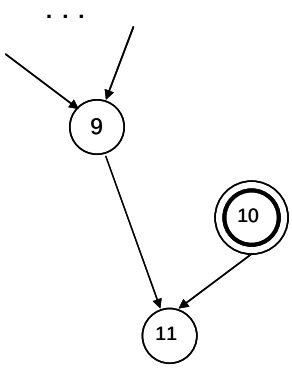


Figura 3.10 Incorporación de un UO Reutilizado

UO9- *Recetas y Recomendaciones*: Escribir…

UO10- *Imprimir a mi gusto:* Quiero imprimir sólo una receta o todas, o sólo las recomendaciones.

UO11- *Escribir, Leer e Imprimir recetas, recomendaciones:* Escribir, leer o imprimir a mi gusto recetas y recomendaciones de uno o varios productos.

La reutilización de UOs es una opción a valorar ya que agiliza el desarrollo de un proyecto. La formalización de estos UOs provenientes de otros proyectos o del mismo proyecto es un trabajo, actualmente en curso, del que se habla con más detalle en el capítulo 6.

3.2.3 Evolución del proyecto por UOs creados

La evolución de un proyecto puede verse gráficamente con facilidad a través del Diagrama de Creación de UOs. Este tipo de diagrama muestra los UOs creados, su tipo, las fusiones y divisiones, los incrementos y las reutilizaciones de UOs, en cada iteración.

El diagrama de la Figura 3.11, por ejemplo, documenta la evolución del proyecto *WebButcher* en las 8 primeras iteraciones.

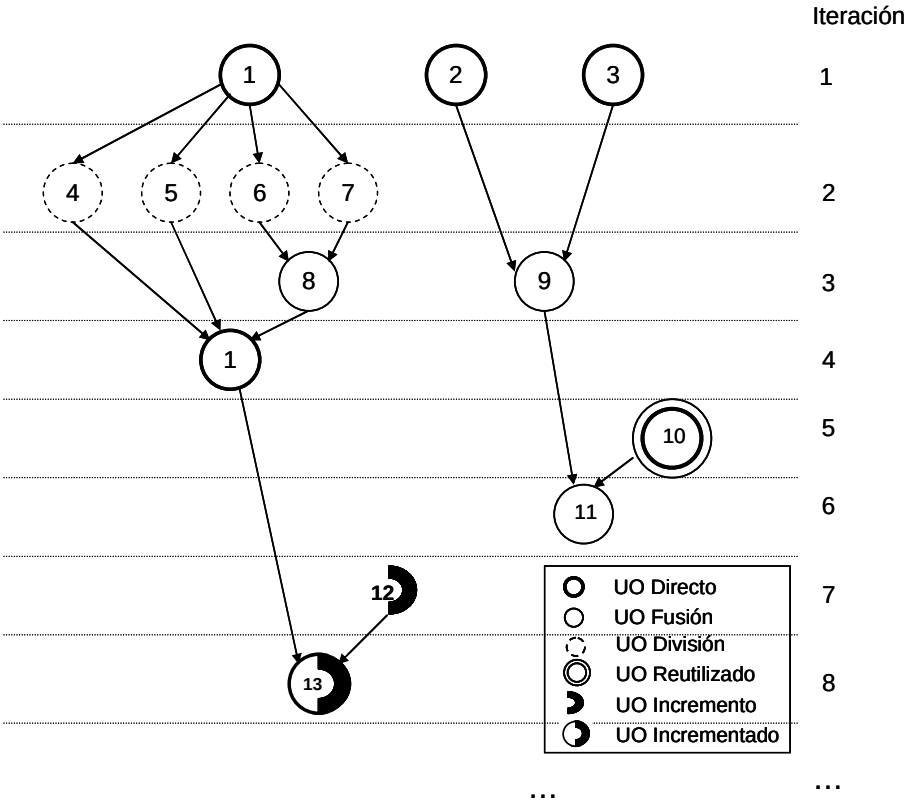


Figura 3.11 Diagrama de Creación de UOs del proyecto *WebButcher*

3.3 Actividades para un progreso incremental centrado en el usuario

La realización de un proyecto implica planificar, en cada iteración, las actividades asociadas al desarrollo de ciertos UOs. Cada actividad tiene por objetivo la realización y evaluación de uno de estos tres tipos de modelos del desarrollo: *Modelo de Requisitos*, *Modelo de Presentación* y *Modelo de Funcionalidad*. El paso de una iteración a la siguiente se establece tras los procesos de evaluación efectuados en dichas actividades. Primero se explican los tipos de actividades que pueden desarrollarse en un proyecto, mediante fragmentos del proyecto *WebButcher* y segundo, se muestran las actividades realizadas en *WebButcher*.

3.3.1 Tipos de Actividades

InterMod establece que cada actividad esta dirigida por modelos, siguiendo la propuesta del Object Management Group's Model-Driven Architecture ("MDA", n.d.). Además, toda actividad incluye la realización y evaluación del modelo correspondiente. Una actividad puede estar activa durante varias iteraciones hasta conseguir una evaluación positiva o su abandono. Dicho de otro modo, una actividad se considera finalizada cuando el modelo correspondiente ha sido creado y validado. Hay que tener en cuenta, sin embargo, que todos los modelos pueden ser revisados y modificados durante el proceso por lo que es posible reactivar actividades finalizadas.

3.3.1.1 Actividades de Desarrollo e Integración

InterMod establece dos tipos de Actividades principales que remiten al desarrollo de los UOs en cada iteración: Actividades de Desarrollo y Actividades de Integración.

Se pueden asociar tres tipos de Actividades de Desarrollo (DAs) con cada UO (Tabla 3.4). Expresaremos la realización de una actividad k determinada, sobre un objetivo UOi, como ***DA-k (i),*** *donde* $k \in (1..3) \wedge i \in$ *{Números UOs}*.

Tabla 3.4 Actividades de Desarrollo

	Nombre	**Representación**
DA-1	Análisis y Diseño de la Navegación	
DA-2	Construcción de la Interfaz	
DA-3	Codificación de la Lógica de Negocio	

Además de las DAs, para asegurar el progreso incremental adecuado del proyecto son necesarias las Actividades de Integración (IAs). Una actividad k de integración trabaja sobre un UOi fusión, e integra en un modelo k de dicho UO los modelos k creados y evaluados de los UOs fusionados. Estos procesos de integración van configurando la aplicación final y son la causa de numerosas revisiones en los modelos integrados. Hay tres tipos de actividades de integración (Tabla 3.5). Las actividades de integración realizadas sobre un objetivo fusión se expresan como ***IA-k(i),*** *donde* $k \in (1..3) \wedge i \in$ *{Números UOs fusión}.*

Tabla 3.5 Actividades de Integración

	Nombre	**Representación**
IA-1	Integración de los Modelos de Requisitos	
IA-2	Integración de la Interfaz	
IA-3	Integración de la codificación y Refactorización	

3.3.1.2 Actividades y Modelos. Perspectiva DCU

Existe un paralelismo entre las Actividades de Desarrollo y las Actividades de Integración en cuanto a su relación con los modelos creados (Figura 3.12). Las actividades **DA-1.** Análisis y Diseño de la Navegación e **IA-1.** Integración de los Modelos de Requisitos crean y evalúan el modelo M-1. *Modelo de Requisitos* para el objetivo implicado. La actividad **DA-2.** Construcción de la Interfaz crea y evalúa el modelo M-2. *Modelo de Presentación* para un UO previamente diseñado y evaluado, e **IA-2**. Integración de la Interfaz integra los M-2 de los UOs fusionados. Por último, la actividad **DA-3**. Codificación de la Lógica de Negocio e **IA-3**. Integración del Código y Refactorización crean y evalúan el modelo M-3. *Modelo de Funcionalidad* que guía la implementación en un lenguaje de programación concreto. Todos los aspectos relativos a los modelos: sus características y dependencias, se presentan con

detalle en el capítulo 4. Siguiendo con la nomenclatura empleada para las actividades, los modelos creados y evaluados para un determinado UO se expresan como ***M-k(i)*** *donde* $k \in (1..3) \wedge i \in$ *{Números UOs}*.

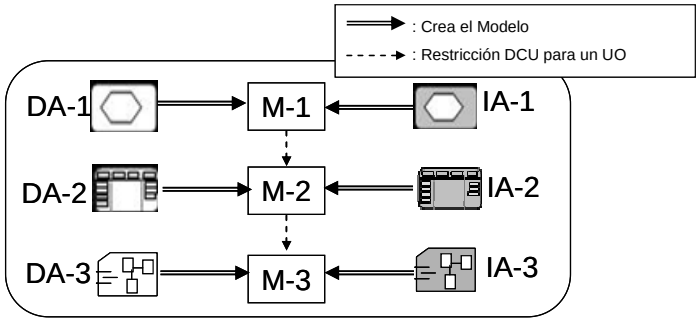


Figura 3.12 Relación Actividades y Modelos de un UO

Las actividades de los UOs en InterMod son muy parecidas a las desarrolladas en las fases de la Ingeniería de Software clásica, aunque su orden de realización se basa en restricciones diferentes. De acuerdo con las recomendaciones del DCU, una vez efectuados y evaluados los requisitos de un determinado UO (M-1), su interfaz debe ser generada y validada (M-2) antes de su codificación (M-3). Esto es, si un UO simple se considera como un proyecto pequeño, la realización de sus actividades deben asegurar que la validación de sus modelos se ordene como propone el DCU.

Por ejemplo, la Figura 3.13 muestra el caso de un UOi en el que se han finalizado las tres actividades de desarrollo de forma consecutiva y por consiguiente, la creación de modelos respeta la perspectiva DCU establecida en InterMod.

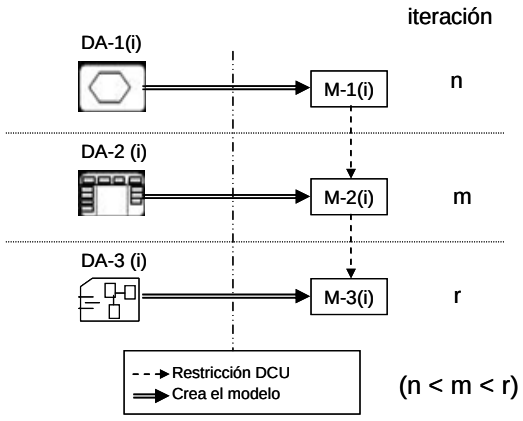


Figura 3.13 Desarrollo de los Modelos para un UOi

Sin embargo, considerando en su globalidad un proyecto complejo compuesto de diferentes UOs, la relación de orden entre las actividades de diferentes UOs está determinada únicamente por el interés ágil, derivado del progreso del proyecto o del interés del usuario. En

el ejemplo de la Figura 3.14, se realizan tres DAs distintas que producen tres tipos de modelos para distintos UOi en una relación de precedencia u orden de realización diferente del propuesto por DCU.

En este caso, la codificación de un UOi (M-3(i)) se realiza antes de efectuar el análisis de los requisitos de otro UOj (M-1(j)), y éste antes de la interfaz de otro UOk (M-2(k)).

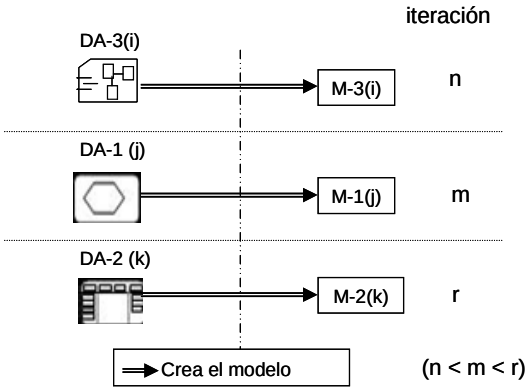


Figura 3.14 Realización de Desarrollo de los Modelos para varios UOs

Esto es, a diferencia de lo que postula el DCU, no es necesario que la interfaz de toda la aplicación (M-2) esté completamente desarrollada antes de pasar a codificarq la lógica de negocio (M-3), sino que este enfoque queda enmarcado exclusivamente al desarrollo de cada UO. Resumiendo, cada UO desarrolla ordenadamente sus modelos mediante la realización directa o indirecta de sus actividades de desarrollo e integración, pero en un momento dado pueden estar realizándose actividades diferentes para distintos UOs.

Ejemplo de Actividades y modelos

En el ejemplo de la Figura 3.15 se observa un proceso de desarrollo e integración de modelos con dos vistas: según el diagrama de actividades finalizadas (parte izquierda) y según el diagrama de creación de modelos (parte derecha). Inicialmente, se realizan las Actividades de Desarrollo DA-1(2) y DA-1(3), que crean los modelos M-1 para los UOs 2 y 3. Después, se efectúan las actividades DA-2 para esos mismos UOs, creándose sus modelos M-2. Finalmente, se produce una fusión de los UOs 2 y 3 en UO9. Esto implica el comienzo de un proceso de integración de modelos: la realización de la Actividad de Integración IA-1(9) y después IA-2(9), que integran los modelos correspondientes siguiendo el orden establecido por la perspectiva DCU. Tras ello, el desarrollo continúa en UO9 realizando la actividad de desarrollo DA-3(9) que efectúa el modelo M-3 para UO9. Este ejemplo muestra una

posibilidad de desarrollo de actividades para una parte de *WebButcher*, aunque no es el finalmente escogido (ver Figura 3.18).

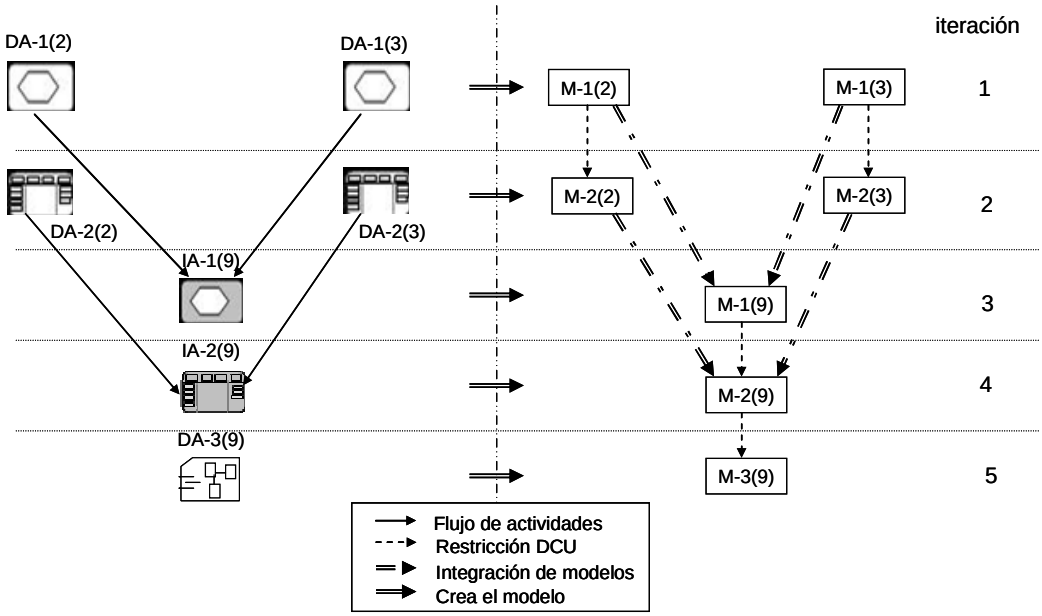


Figura 3.15 Diagrama de Actividades (DAs + IAs) y sus Modelos, para un desarrollo posible en *WebButcher*

3.3.1.3 Actividades de Integración Incremental

Las actividades de integración incremental son una variante de las actividades de integración que se efectúan cuando no existen todos los modelos a integrar de los UOs fusionados, y por lo tanto es necesario realizar a la vez integración y desarrollo de modelos. Ésta es una manera frecuente de trabajo en un proceso ágil. En los diagramas de actividades, las de integración incremental se representan con la imagen de la Actividad de Integración correspondiente y un doble recuadro, y se expresan como ***IA-k(i)*** (subrayado), *donde* $k \in (1..3) \wedge i \in$ *{Números UOs}*.

Este tipo de actividades pueden realizarse sobre un UO fusión o sobre un UO incrementado. Considerando un objetivo fusión, una actividad de Integración Incremental crea un modelo incremental a partir de algunos modelos (no todos) de los UOs fusionados. En la Figura 3.16 se efectúa la Actividad de Integración Incremental IA-2(9), donde UO9 es la fusión de los UOs 2 y 3. En este desarrollo, se produce M-2(9) extendiendo el modelo M-2(2) generado por DA-2(2). Este ejemplo muestra otra posibilidad de desarrollo para *WebButcher* que, como en el ejemplo de la Figura 3.15, no es la escogida finalmente.

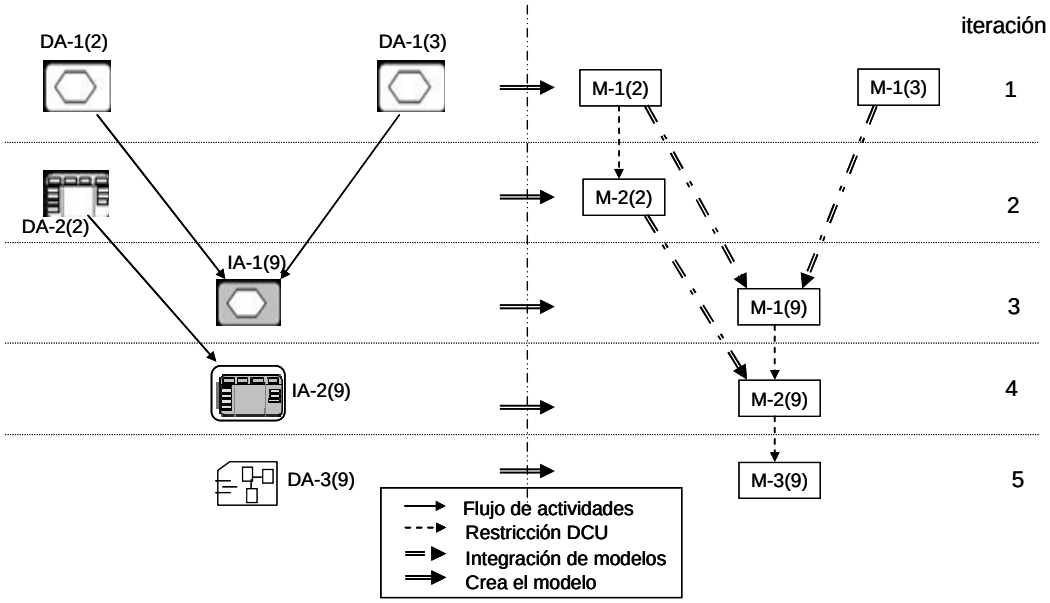


Figura 3.16 Diagrama de Actividades (DAs + IAs + IAs) y sus Modelos, para un desarrollo posible en *WebButcher*

Por otra parte, en los UOs Incrementados interviene siempre una Actividad de Integración Incremental, ya que los modelos de los UOs incremento implicados no existen individualmente. En la Figura 3.17 se muestra un proceso habitual de desarrollo en el que intervienen actividades de Integración Incremental y que es la opción escogida en *WebButcher*.

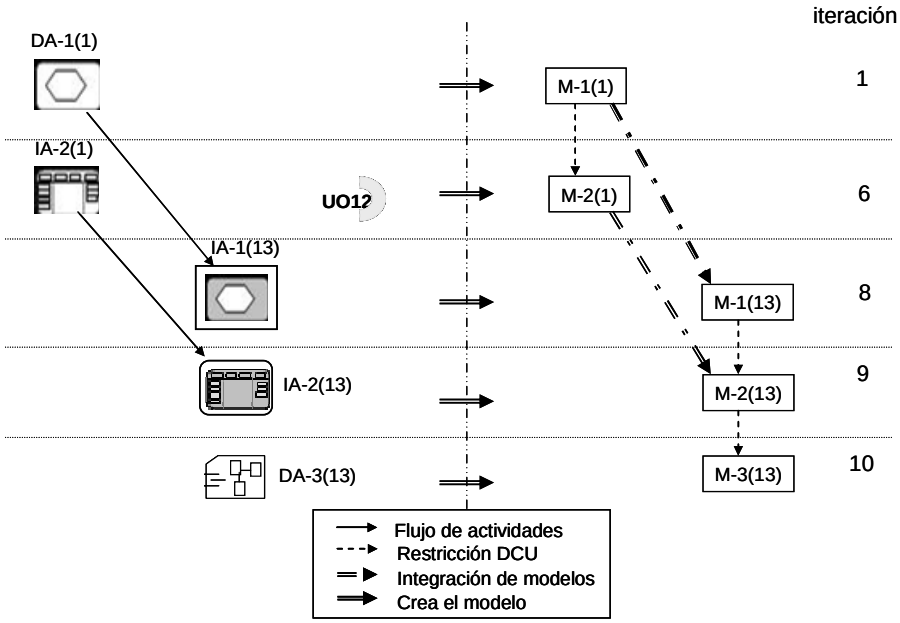


Figura 3.17 Diagrama de Actividades (DAs + IAs) y sus Modelos, para un desarrollo posible con UO Incremento en *WebButcher*

La actividad IA-1(13) es una actividad incremental para el UO incrementado UO13 y produce el modelo M-1(13), incrementando M-1 de UO1 con la información del UO Incremento UO12. La actividad IA-2(13) es también incremental ya que el modelo M-2(13) se obtiene incrementando el modelo M-2 de UO1 con las necesidades de UO12. En este ejemplo, a partir de este modelo, UO13 continúa su desarrollo individualmente con la actividad DA-3(13).

3.3.2 Evolución de un proyecto por actividades

Al igual que el diagrama de creación de UOs permite visualizar la evolución de un proyecto a través de los UOs creados en cada iteración (ver sección 3.3.3), los diagramas de actividades permiten visualizar el proyecto por actividades comenzadas, por actividades en desarrollo o por actividades finalizadas. El aspecto de los diagramas es semejante, salvo que la aparición de una actividad en una determinada iteración dependerá de si ha comenzado, está en desarrollo o ha finalizado en esa iteración. Si una actividad se planifica, se desarrolla y finaliza en la misma iteración, las tres visiones coinciden.

En el diagrama de la Figura 3.18 se observa la evolución por actividades finalizadas del proyecto *WebButcher*, visualizado según la creación de UOs en la Figura 3.11.

En cada iteración se indican las Actividades de Desarrollo e Integración (incluyendo las de Integración Incremental) que han finalizado para cada UO. Esto significa que únicamente aparecen, en cada iteración del diagrama, aquellas actividades que han creado un modelo y éste ha sido evaluado positivamente. Si una actividad no ha finalizado, ya sea porque está inconclusa o bien porque el modelo no ha sido evaluado positivamente, no aparece en este diagrama. Además, hay que tener en cuenta que una actividad puede darse por finalizada y sin embargo, volver a ser activada para una revisión o cambio mayor; esto significa que puede aparecer varias veces en este diagrama. Es interesante observar que en un momento dado (iteración), se pueden realizar actividades de distinto tipo para diferentes UOs. Por ejemplo, en la iteración 3 se realizan las actividades {DA-2, IA-2, IA-1}.

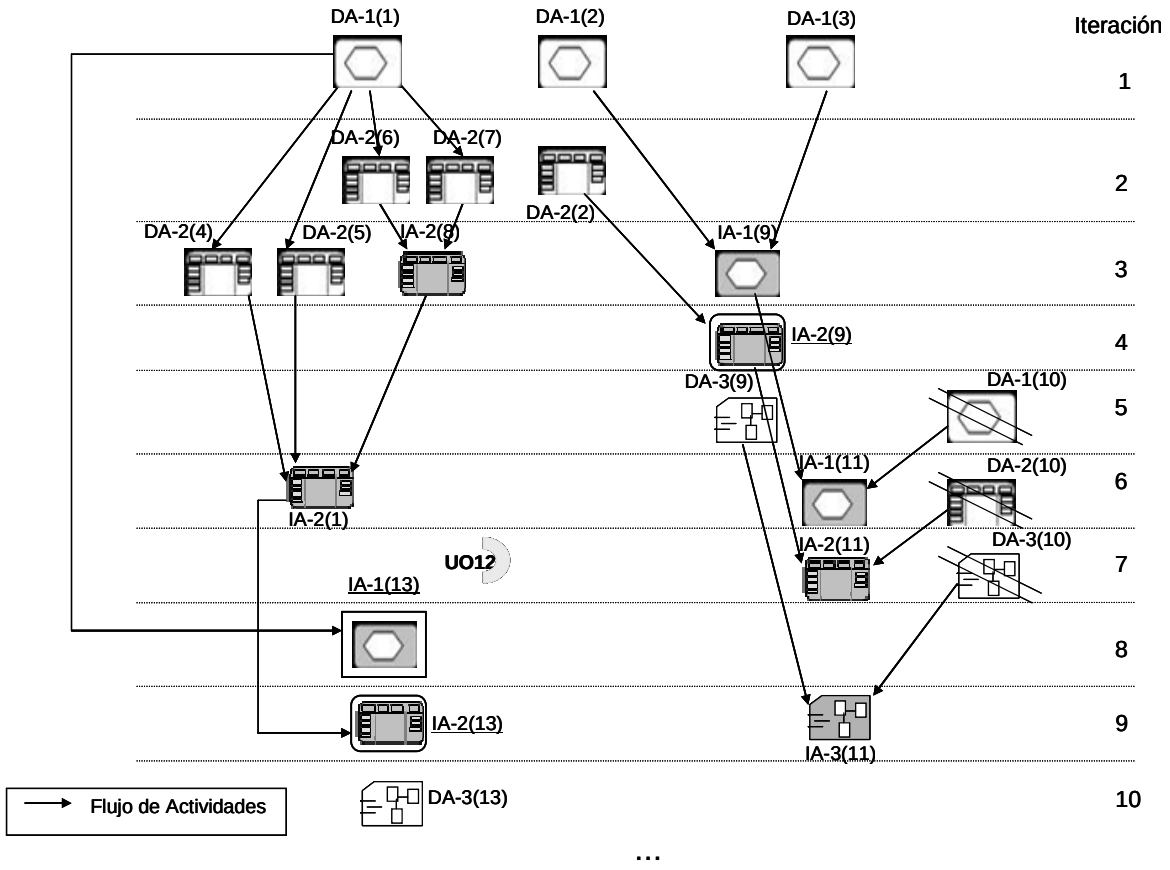


Figura 3.18 Diagrama de Actividades finalizadas del Proyecto *WebButcher*

Las actividades que aparecen tachadas: {DA-1(10), DA-2(10), DA-3(10)} representan, en el diagrama, la reutilización en ese punto del resultado de dichas actividades, realizadas con anterioridad

3.4 Proceso ágil para el desarrollo de aplicaciones interactivas

El proceso iterativo en InterMod (Figura 3.19), comienza con un análisis global (Paso 0), en el que se establece el marco de desarrollo común para el proyecto. Después, se continúa con una secuencia de iteraciones en las que se va perfilando ese marco común y obteniendo la aplicación concreta (Losada et al., 2013a) .

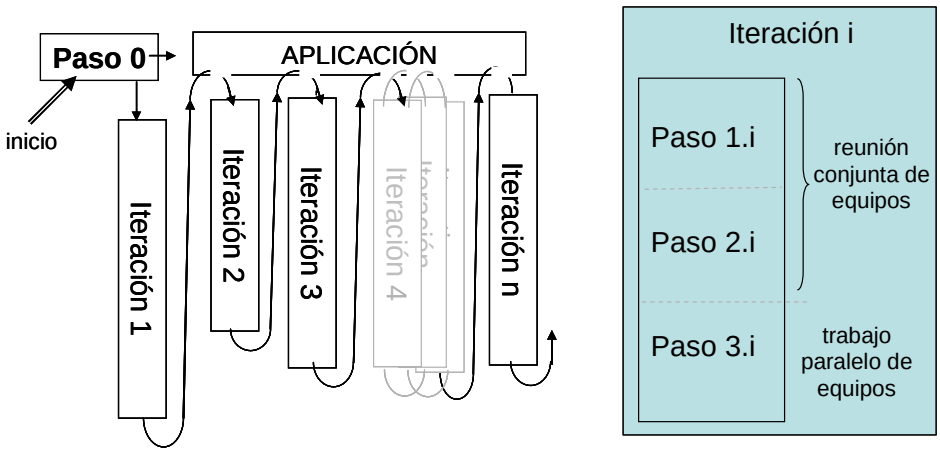


Figura 3.19 Pasos e Iteraciones del proceso InterMod

En cada iteración se toman, de modo ágil, las decisiones oportunas de desarrollo. Cada iteración comienza con una reunión de gestión en la que se actualiza la Lista de Objetivos de Usuario (Paso 1.i), y se planifica la iteración paralela para cada equipo teniendo en cuenta el progreso del proyecto y sus necesidades (Paso 2.i). Esta planificación consiste en seleccionar los UOs a desarrollar, las actividades a realizar entre las posibles, y los equipos que las realizarán. Finalmente, tomadas las decisiones de planificación, cada equipo progresa en la aplicación realizando las actividades de la iteración asignadas (Paso 3.i).

En la Figura 3.20 se resume el proceso de desarrollo de aplicaciones interactivas en InterMod: sus pasos, actividades y modelos.

Todas las iteraciones están guiadas por el mismo plan de actuación: *Organizar el trabajo de desarrollo ágil en base a los Objetivos de Usuario, que se van materializando con los modelos validados que se producen a partir de las actividades que realizan los equipos*. A continuación se explican los pasos del proceso con detalle.

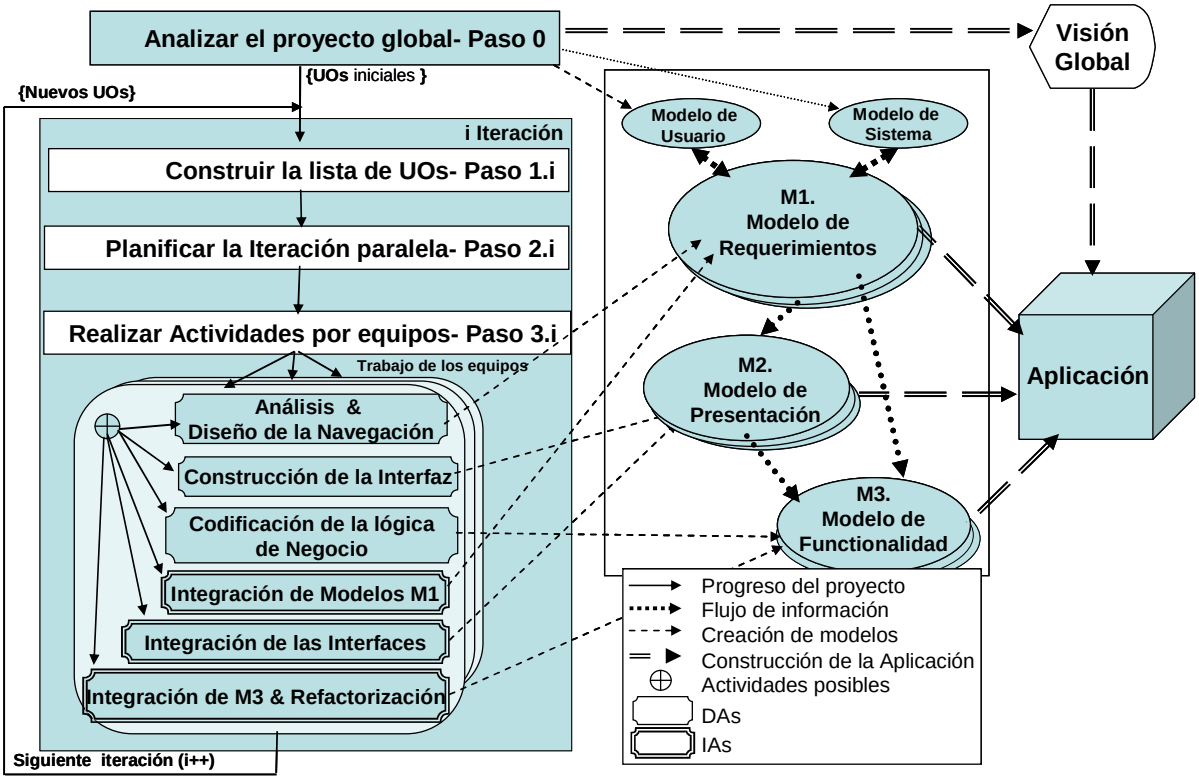


Figura 3.20 Pasos, Actividades y Modelos en la metodología InterMod

3.4.1 Análisis Global del Proyecto – Paso 0

Al comienzo del proyecto es necesario analizar el proyecto en su globalidad para determinar: (a) los UOs iniciales (UOs Directos iniciales), y (b) las decisiones de diseño globales. Los UOs iniciales (normalmente los más importantes o necesarios), junto con un menú general provisional que incorpora algunas funcionalidades, proporcionan una visión general primitiva de la aplicación. Esta visión general sirve de guía para situar los primeros UOs en la interfaz, así como para consensuar elementos gráficos generales como logos, atajos, metáforas, etc. Las decisiones de diseño globales se refieren a algunos requisitos generales de la aplicación, como tamaño de la letra, o a estrategias generales de codificación, como el lenguaje o el tipo de base de datos que se utilizará, o el nivel de accesibilidad o de seguridad requerido. Los implicados (usuarios, desarrolladores y gestores) toman parte en estas decisiones globales, mientras que la colaboración de un grupo de futuros usuarios de la aplicación es fundamental para determinar los UOs iniciales. Estos UOs se extraen a partir de la información obtenida

mediante métodos de indagación de HCI. Los criterios de decisión se basan en descubrir las funcionalidades más necesarias y prioritarias.

Todos los desarrollos del proyecto, guiados por los UOs, van a heredar las decisiones de diseño globales con el fin de garantizar, en toda la aplicación, su coherencia con el sistema y con las necesidades y preferencias de los usuarios. En cada iteración se pueden modificar o ampliar estas decisiones. Es decir, es posible que surjan cambios que vayan matizando y/o modificando la visión final del producto.

3.4.2 Construir la Lista de Objetivos de Usuario - Paso1.i

Los requisitos de la aplicación se recogen incrementalmente durante la construcción progresiva de la Lista de UOs. Un UO que se crea en una iteración permanece en la Lista de UOs durante el resto del proceso, ya que puede ser utilizado o actualizado en cualquier momento del proceso. La lista de UOs aumenta progresivamente a lo largo del proceso, de esta manera la Lista de UOs se modifica en el sentido de las metodologías ágiles (Larman, 2003), después de las evaluaciones realizadas por los implicados en el proyecto o bien por las opiniones surgidas en las reuniones continuas de los miembros del mismo equipo y entre diferentes equipos. Cada iteración actualiza la Lista de UOs con nuevos UOs, derivados de las necesidades nuevas que van surgiendo en el proyecto (UOs Directos), o generados a partir de la división o fusión de los UOs previos (UOs Indirectos).

El ejemplo de la Tabla 3.6 muestra la actualización de la Lista de UOs del proyecto *WebButcher* en las 10 primeras iteraciones (ver Figura 3.11).

Tabla 3.6 Actualización por iteración de la lista de Uos en el proyecto WebButcher

Iteración	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Lista_UOs	1	4	**8**	1	10	**11**	~~12~~	~~**13**~~		
	2	5	**9**							
	3	6								
		7								

n: UOn Directo | n: UOn división Indirecto | **n**: UOn Fusión Indirecto
n: UOn Reutilizado | ~~n~~: UOn Incremento | ~~**n**~~: UOn Incrementado

Para distinguir rápidamente los tipos de UOs en la tabla se utiliza la siguiente grafía: {normal para UOs Directos, subrayado para UOs divididos, **negrita** para UOs fusionados, enmarcado para UOs Reutilizados, ~~tachado~~ para UOs Incremento y ~~**tachado negrita**~~ para UOs Incrementados}.

En la primera iteración se incluyen tres UOs Directos {UO1, UO2, UO3}. La lista aumenta cuando se produce una división de UO1 en {UO4, UO5,UO6,UO7} en la iteración 2, y cuando se crean los UOs fusión UO8 y UO9 en la iteración 3, y UO11 en la iteración 6. También aumenta con la aparición de nuevos deseos de usuario, como el UO reutilizado UO10 en la iteración 5, y con los incrementos de funcionalidad a través de UO12 en la iteración 7 y UO13 Incrementado en la 8. Esto es, la lista creada en la iteración 1 es: Lista_UOs(1)= {UO1, UO2, UO3}; en la interación 2, se formará con todos los UOs de la iteración 1 más los nuevos UOs surgidos: Lista_UOs(2)= lista_UOs(1) ∪ {UO4, UO5, UO6, UO7}. En general, diremos que la Lista de UOs de toda iteración, salvo la primera, estará formada por todos los UOs de la iteración anterior y los nuevos UOs surgidos en esa iteración.

3.4.3. Planificar la Iteración Paralela - Paso2.i

El avance del proyecto se decide en cada iteración tras establecer tres cuestiones muy interrelacionadas (pero no necesariamente en este orden): qué UOs desarrollar, qué actividades realizar para los UOs seleccionados y cómo distribuir las diferentes actividades entre los equipos. Muchas veces, la respuesta a una cuestión viene condicionada por las otras; por ejemplo, se selecciona un UO por estar muy avanzado su desarrollo, o se escoge una actividad por la disponibilidad de un equipo. El estado del proyecto permite determinar qué actividades pueden abordarse considerando la perspectiva DCU y la relación entre los UOs. En función de esto, InterMod establece unas Reglas de ayuda a la planificación de Actividades (Sección 3.5).

¿Qué UOs desarrollar?

Considerando en cada iteración la Lista_UOs correspondiente (Tabla 3.6), el equipo gestor toma la decisión de qué UOs desarrollar, como muestra la Tabla 3.7.

Tabla 3.7 Selección de UOs a desarrollar en cada iteración, en el proyecto WebButcher

Iteración	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**…**
Selección_UOs	1	2	4	**9**	**9**	1	10	**11**	**11**	~~**13**~~	
	2	6	5		10	10	**11**	~~**13**~~	~~**13**~~		
	3	7	**8**			**11**					
			9								

n: UOn Directo | n: UOn división Indirecto | **n**: UOn Fusión Indirecto
n: UOn Reutilizado | ~~n~~: UOn Incremento | ~~**n**~~: UOn Incrementado

Esta decisión se basa en factores como la prioridad establecida para cada UO, la cercanía a la finalización de un UO (especialmente cuando es un UO Directo) o la disponibilidad de los equipos. Por ejemplo, en la iteración 5, los esfuerzos de desarrollo se enfocaron en la finalización del UO9 y además en la reutilización de UO10, por disponibilidad de otro equipo.

¿Qué actividades hacer para los UOs seleccionados?

Para determinar las actividades posibles a realizar, para cada UO se considera: 1) el estado de desarrollo de cada uno de los objetivos seleccionados y 2) las estrategias de desarrollo recomendadas por el DCU y asumidas por InterMod.

Hay que tener en cuenta que aunque los prerrequisitos del proceso determinan las DAs e IAs posibles en cada iteración, la elección dependerá también del número de actividades posibles a realizar y la experiencia de los grupos de trabajo. Igualmente, otras cuestiones de organización del trabajo como las prioridades de realización y de evaluación pueden influir en la toma de decisión en este punto. Esto es, el progreso del proyecto depende de las actividades finalizadas (y como consecuencia, de los modelos creados cuya evaluación ha sido satisfactoria), pero su evolución no es predecible. En la Figura 3.21 se muestran dos Diagramas de Actividades finalizadas para dos desarrollos diferentes de UO1, en *WebButcher*.

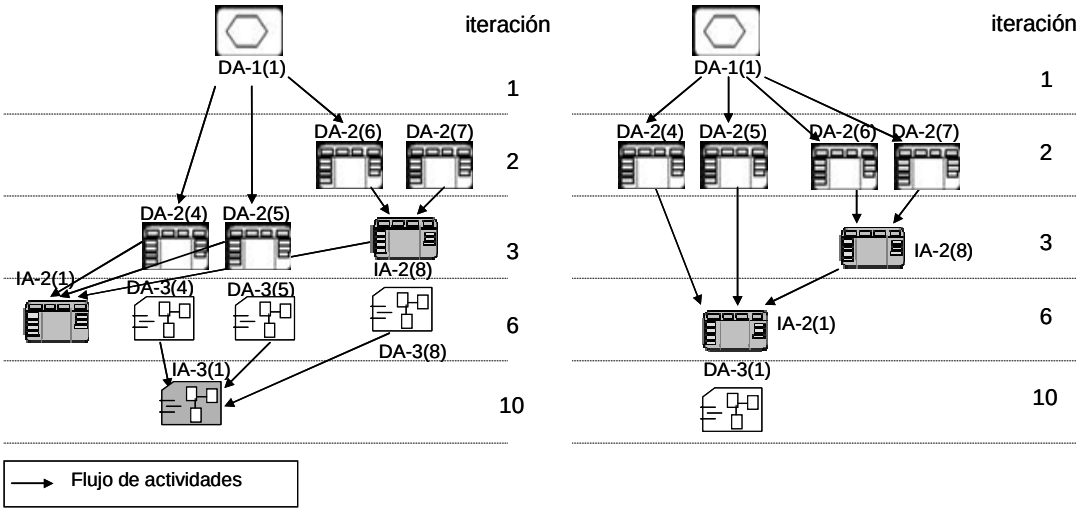


Figura 3.21 Paso 2.i –Selección posible de Actividades de UOs a Desarrollar en *WebButcher*

En el proceso de la parte izquierda se logra mediante integraciones de todos los modelos M-2 y M-3, mientras que en la evolución que aparece en la parte derecha se logra sólo por integración de los modelos M-2 y desarrollo posterior del modelo M-3.

¿Cómo distribuir las diferentes actividades entre los equipos de trabajo?

Las actividades seleccionadas se distribuyen entre los equipos del proyecto. La especialización en las actividades o en los UOs, así como el nivel de experiencia de los equipos influirá en la distribución de las actividades. La Tabla 3.8 resume la toma de decisión realizada en cada iteración para el proyecto *WebButcher*: los UOs seleccionados, las actividades a desarrollar y la distribución del trabajo en los tres equipos de desarrollo.

En este proyecto, mientras el primer equipo se especializa en el desarrollo de UO1 y el equipo 2 en el desarrollo de UO2 y UO3, el equipo 3 actúa de soporte de ambos equipos. La información incluida en la Tabla 3.8 recoge el plan tras la toma de decisiones de cada iteración y es parte de la Hoja de Gestión asociada al desarrollo del proyecto (sección 3.4.5).

Tabla 3.8 Distribución de las actividades por equipo, en cada iteración

PLAN	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**…**
1er. equipo	DA-1(1)	DA-2(4) DA-2(5) DA-2(6)	DA-2(4) DA-2(5)			IA-2(1)			IA-2(13)		
2º equipo	DA-1(2)	DA-2(2)	IA-1(9)	IA-2(9)	DA-3(9)	IA-1(11)	IA-2(11)	IA-3(11)	IA-3(11)		
3er. equipo	DA-1(3)	DA-2(7)	IA-2(8)		DA-1(10)	DA-2(10)	DA-3(10)	IA-1(13)		DA-3(13)	

DA-k(n): Actividad de Desarrollo donde k = {1,2,3} y n es el número de UO
IA-k(n): Actividad de Integración
IA-k(n): Actividad de Integración Incremental

3.4.4 Realizar las Actividades de la Iteración - Paso3.i

Cada equipo lleva a cabo las actividades establecidas en el plan. Realizar una actividad para un UO consiste en crear el modelo asociado y validarlo con los métodos adecuados (sección 4.2.3). Al finalizar cada iteración se debe determinar si una actividad ha finalizado, esto es, si el modelo correspondiente ha sido validado satisfactoriamente. En una tabla de "modelos creados y evaluados", se va recogiendo la información del momento (iteración) en el que los modelos se han considerado válidos y el tipo de actividad realizada para obtenerlos (Tabla 3.9). De hecho, ésta es otra manera de visualizar las actividades finalizadas del proyecto mostradas en el diagrama de actividades finalizadas de la Figura 3.18. En esta tabla se indica en cada celda, para un modelo M-k (1<=k<=3) y un UOj (1<=j<=13), el tipo de Actividad que ha creado y evaluado positivamente el modelo M-k(j), y el número de iteración i en que se ha obtenido. Esto es:

- Xi- Actividad de Desarrollo en la iteración i
- Ii- Actividad de Integración en la iteración i
- Ii- Actividad de Integración Incremental en la iteración i

Tabla 3.9 Momento de creación de modelos (hasta la iteración 10)

	UO1	**UO2**	**UO3**	**UO4**	**UO5**	**UO6**	**UO7**	**UO8**	**UO9**	**UO10**	**UO11**	**UO12**	**UO13**
M-1	X1	X1	X1						I3	X5	I6		I8
M-2	I6	X2		X3	X3	X2	X2	I3	I4	X6	I7		I9
M-3									X5	X7	I9		X10

Las celdas sombreadas indican que los modelos k correspondientes a un UOj no se han realizado por una actividada aplicada diresctamente a dicho UOj. Por ejemplo, la celda (M-1, UO9) contiene el valor I3 que indica que el modelo M-1(9) se ha creado y evaluado positivamente en la iteración 3 gracias a la actividad de integración I-A(9). Los modelos asociados a UOs en celdas sombreadas, no se obtienen por actividades realizadas directamente sobre los UOs, sino indirectamente mediante su creación a través de los modelos de otros UOs. Por ejemplo, {UO4, UO5, UO6, UO7, UO8}, todos conforman el UO1. Eso significa que al estar creado y evaluado M-1(1), los modelos {M-1(4), M-1(5), M-1(6), M-1 (7), M-1(8)} también están creados y evaluados por ser parte del modelo M-1(1). En el capítulo 4, se explican con detalle las circunstancias que determinan la creación de modelos de forma indirecta. La Tabla 3.9 forma parte del estado del proyecto, recogido también en la Hoja de Gestión (sección 3.4.5).

Habitualmente, también en este paso suceden las nuevas iniciativas de actualización de los marcos comunes de desarrollo de la aplicación con respecto a características del Sistema y del Usuario. Además, especialmente durante la realización de actividades de integración e integración incremental, pueden reactivarse modelos creados y evaluados anteriormente para el mismo UO u otros UOs. Es decir, en este punto es necesario poner en revisión los modelos para asegurar la coherencia y actualización de la aplicación.

3.4.5 El modelo del proceso en InterMod

Un proyecto que sigue la metodología InterMod sigue un proceso iterativo e incremental, dirigido por modelos y centrado en el usuario, cuya finalidad es la realización de los UOs que forman el proyecto. Además, esta metodología busca que el resultado se adecue a las necesidades de los usuarios, por lo que las evaluaciones con los implicados son importantes.

La Figura 3.22 formaliza el modelo de proceso de InterMod. Un proyecto se compone de iteraciones y equipos de trabajo. Al principio del proyecto se configura el *Modelo de Usuario* y el *Modelo del Sistema*, los cuales son evaluados y pueden ser revisados a lo largo del proceso. Cada iteración actualiza la *Lista de UOs* y genera un *Plan* de trabajo que selecciona los UOs-*Lista UOs Actual* y las actividades posibles (de esos UOs)-*DAs* e *IAs* a realizar (Las actividades IAs están incluidas en las IAs). Los *equipos* llevan a cabo las actividades de desarrollo y de integración (incluyendo las de integración incremental) que les asigne el plan de trabajo. Cada actividad efectúa su *modelo* correspondiente asociado al UOi en desarrollo, e indirectamente puede efectuar otros modelos de otros UOs relacionados con el UOi; además, un grupo de *evaluadores* compuesto por *usuarios*, *diseñadores* y *desarrolladores* realiza procesos de evaluación sobre esos modelos.

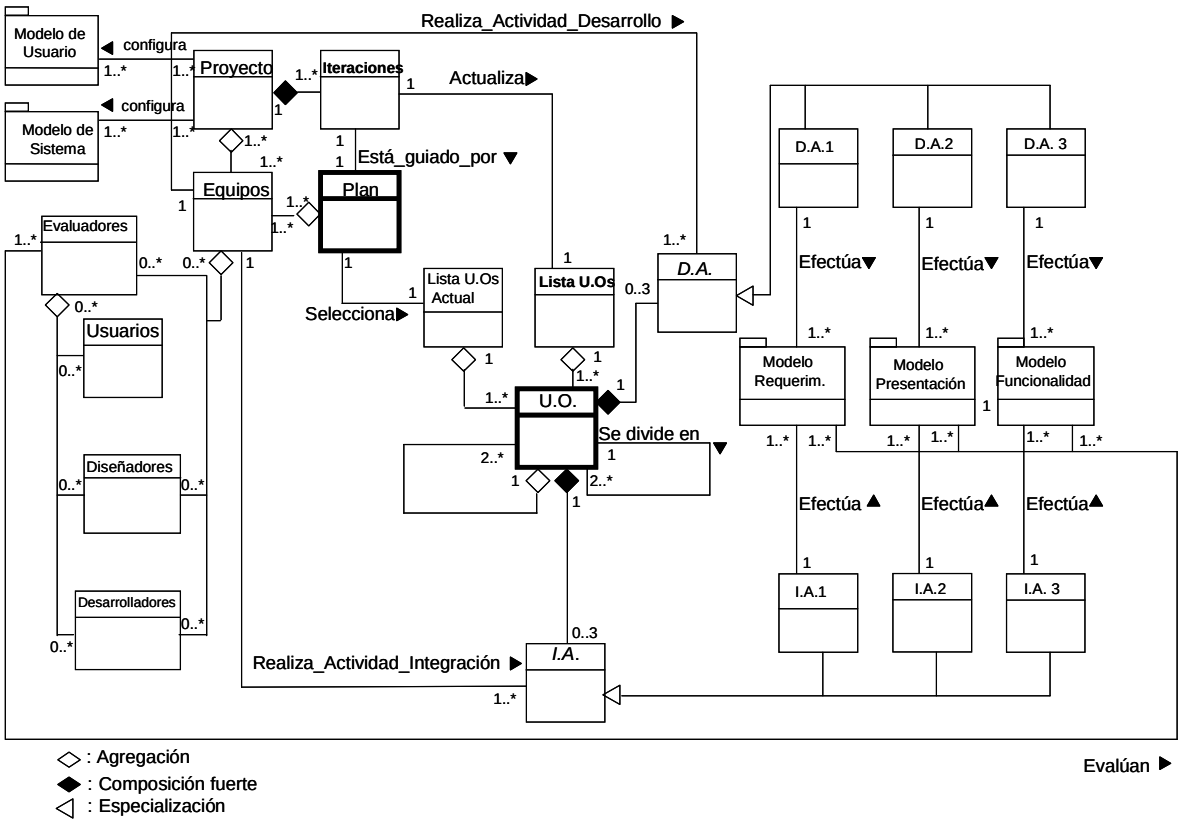


Figura 3.22 Modelo del proceso InterMod

Para todos los Objetivos de Usuario del proyecto se debe asegurar un desarrollo correcto y de calidad. La división del proyecto en UOs otorga al proyecto suficiente flexibilidad como

para plantearlo desde el punto de vista de las metodologías ágiles; sin embargo, no hay que perder de vista la calidad y rigor en el proceso de desarrollo obtenido gracias a la propuesta Model-Driven y User-Centered en InterMod.

La gestión del proyecto se lleva a cabo de forma iterativa, planificando en cada iteración las actividades a desarrollar para los UOs seleccionados y creando los modelos pertinentes con la validación adecuada correspondiente al tipo de modelo. Durante el proceso del desarrollo del proyecto se registran, iteración a iteración, todos los acontecimientos relevantes del mismo en una Hoja de Gestión: UOs creados, Lista de UOs, actividades a realizar, etc. La Hoja de Gestión del proyecto ayuda a visualizar su progreso y a tomar decisiones en cada iteración.

Las Tabla 3.10 y 3.11 muestran las dos partes de la Hoja de Gestión del proyecto *WebButcher* en sus 10 primeras iteraciones, mostradas parcialmente a lo largo de este capítulo.

Una parte de la Hoja de Gestión, la Tabla 3.10, incluye información sobre el estado del proyecto asociada a los UOs que se van creando: sus características de fusión, división e incrementado, y el momento y manera de creación de sus modelos (Xi, Ii, Ii, realización indirecta). El Diagrama de actividades finalizadas es la visión gráfica de la información sobre los modelos creados. Esta información es actualizada por el gestor del proyecto, durante el paso 1.i para los UOs, y al final del paso 3.i para los modelos.

Tabla 3.10 Hoja de Gestión del proyecto *WebButcher*: Estado del proyecto por UOs.

UOs	**UO1**	**UO2**	**UO3**	**UO4**	**UO5**	**UO6**	**UO7**	**UO8**	**UO9**	**UO10**	**UO11**	**UO12**	**UO13**
Fusión		**9**	**9**	**1**	**1**	**8**	**8**	**1**	**11**	**11**			
División				**1**	**1**	**1**	**1**						
Increm.	**13**											**13**	
M-1	X1	X1	X1						I3	X5	I6		I8
M-2	I6	X2		X3	X3	X2	X2	I3	I4	X6	I7		I9
M-3									X5	X7	I9		X10

Xi: Realizado por una actividad de Desarrollo en la iteración i
Ii: Realizado por una actividad de Integración
Ii: Realizado por una actividad de Integración incremental
Sombreado: Realizado indirectamente

La segunda parte de la Hoja de Gestión, la Tabla 3.11, incluye la planificación de cada iteración: la incorporación de nuevos UOs a la Lista de UOs y la asignación de actividades a los equipos . El Diagrama de Actividades Planificadas es la visión gráfica de esta información en la Tabla 3.11. Esta parte es actualizada por el gestor tras los pasos 1.i y 2.i, respectivamente.

Tabla 3.11 Hoja de Gestión del proyecto *WebButcher*: Planificación de las iteraciones

Iteraciones		1	2	3	4	5	6	7	8	9	10
Lista de UOs		1 2 3	4 5 6 7	**8** **9**	1	10	**11**	~~12~~	~~**13**~~		
PLAN	1er. equipo	DA-1(1)	DA-2(4) DA-2(5) DA-2(6)	DA-2(4) DA-2(5)			IA-2(1)			IA-2(13)	
	2º equipo	DA-1(2)	DA-2(2)	IA-1(9)	IA-2(9)	DA-3(9)	IA-1(11)	IA-2(11)	IA-3(11)	IA-3(11)	
	3er. equipo	DA-1(3)	DA-2(7)	IA-2(8)		DA-1(10)	DA-2(10)	DA-3(10)	IA-1(13)		DA-3(13)

n: UOn Directo — n: UOn división Indirecto — **n**: UOn Fusión Indirecto
n: UOn Reutilizado — ~~n~~: UOn Incremento — ~~**n**~~: UOn Incrementado
DA-k(n): Actividad de Desarrollo — donde k = {1,2,3} y n es el número de UO
IA-k(n): Actividad de Integración
IA-k(n): Actividad de Integración Incremental

El diagrama de creación de UOs se obtiene de la información de fusión, división e incremento de la parte del Estado del proyecto (Tabla 3.10) y de la Lista de UOs de la parte de Planificación (Tabla 3.11). Por ejemplo en un caso de división: {UO4, UO5, UO6, UO7} son UOs Indirecto división de UO1 (en la Tabla 3.10 se observa el valor 1 en las celdas [División, UO4-5-6-7]), y se han creado en la iteración 2 (en la Tabla 3.11 aparecen los valores 4,5,6,7 en la celda [Lista de UOs, 2]). Otro ejemplo en un caso de fusión: UO8 es un UO Indirecto fusión de {UO6, UO7} (en la Tabla 3.10 se observa el valor 8 en las celdas

[Fusión, UO6-7]), y se ha creado en la iteración 3 (en la Tabla 3.11 aparece el valor 8 en la celda [Lista de UOs, 3]).

3.5 Planificación de Actividades

La planificación de actividades implica: primero, aplicar las reglas de planificación que determinan las actividades posibles a realizar según la perspectiva DCU de InterMod y segundo, establecer un plan de realización y distribución de actividades entre los equipos. Para su mejor comprensión, se explica el significado de algunos predicados utilizados en las reglas de planificación y distribución.

Predicados de Modelos:

- *creadoEvaluado*(M-k(i)): el modelo M-k(i) ha sido creado y validado positivamente.
- *posible*(M-k(i)): es posible crear el modelo M-k(i) porque se cumple la perspectiva DCU.
- *enDesarrollo(M-k(i)):* el modelo M-k(i) está en desarrollo en la iteración actual.

Predicados de UOs:

- *fusionDe*(UOi, {UOl,UOm,…}): UOi es un UO fusión, resultado de la fusión de {UOl, UOm,...}
- *incrementadoDe*(UOi, UOj, {UOl, UOm, …}): UOi es un UO Incrementado a partir de UOj de identidad completa y los UO Incremento {UOl, UOm, …}.

Predicados de Actividades:

- *ListaPosiblesA(Actividad k(i))*: La actividad k de UOi forma parte de la lista de posibles actividades a planificar.
- *Plan (Actividad-k(i)):* La actividad k de UOi se incluye en el plan porque se ha asociado a un equipo en una iteración dada.

3.5.1 Reglas de Planificación

Las reglas de planificación de actividades establecen en cada iteración la lista de actividades posibles a realizar, y están determinadas por los estados de los modelos y sus fusiones.

Las reglas de planificación consideran los estados de los modelos en el proceso para construir, en cada iteración, la lista de actividades posibles a realizar *ListaPosiblesA*, como ayuda a la toma de decisiones. Las reglas de planificación de actividades se exponen a continuación.

3.5.1.1 Regla de Planificación de DAs

Cuando un modelo M-k(i) para un UOi está en estado *posible* se puede planificar DA-k(i). Esto es, DA-k(i) se incluirá en la lista de posibles actividades a planificar.

RPA-1: $\forall i \exists k$ [((1<=k<=3) $\wedge$ *posible*(M-k(i))) $\rightarrow$ ListaPosiblesA(DA-k(i))]

donde i es el nº del UO y k es el tipo de modelo o actividad

3.5.1.2 Regla de Planificación de IAs

Es posible realizar una actividad de integración, IA-k, k $\in$ (1..3), de varios UOs fusionados cuando se dan estas tres condiciones:

1. Varios UOs se han fusionado en un UOi Fusión
2. Los modelos M-k de los UOs fusionados están creados y evaluados
3. El modelo tipo (k-1) de un UOi Fusión está creado y evaluado, y en consecuencia, M-k(i) es *posible*.

RPA-2:$\forall i \forall r \exists k$ [((1<=k<=3) $\wedge$ *fusionDe*(UOi, {UOl,UOm,…}) $\wedge$ r$\in$ {l, m, …}

$\wedge$ *creadoEvaluado*(M-k(r)) $\wedge$ *posible* (M-k(i))) $\rightarrow$ ListaPosiblesA(IA-k(i))]

donde i,l,m,r son nº de UOs y k es el tipo de modelo o actividad.

El ejemplo de la Figura 3.23 muestra cómo se actualiza ListaPosiblesA con IA-1(9).

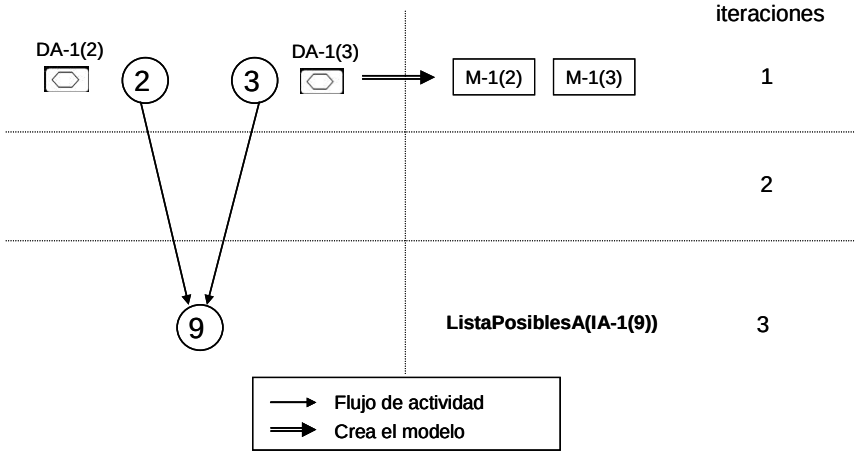


Figura 3.23 Planificación de Actividades de integración

En la iteración 3 es posible planificar IA-1(9) porque:

- UO9 es un objetivo Fusión de UO2 y UO3
- Los modelos M-1(2) y M-1(3) están en estado *creadoEvaluado*. En el ejemplo, han sido efectuados por las actividades DA-1(2) y DA-1(3) en la iteración 1.
- El modelo M-1(9) se encuentra en estado *posible*.

3.5.1.3 Reglas de Planificación de IAs Incrementales

Es posible realizar una actividad de integración Incremental, IA-k, k ∈ (1..3), de varios UOs cuando se dan estas dos condiciones:

- Varios UOs se han fusionado en un UOi Fusión o en un UO Incrementado.
- Algunos modelos M-k de los UOs fusionados están creados y evaluados, y otros no lo están.
- El modelo tipo (k-1) del UOi Fusión está creado y evaluado y, en consecuencia, M-k(i) es *posible*.

Esto se refleja en dos reglas de planificación: para UOs fusión y para UOs Incrementados.

RPA-3 trata el caso de un UOi Fusión, cuando no todos los modelos a fusionar están creados y evaluados:

RPA-3: ∀i∃r∃t∃k [((1<=k<=3) ∧ *fusionDe*(UOi, {UOl,UOm,…}) ∧ *posible* (M-k(i))
∧ r∈ {l, m, …} ∧ *creadoEvaluado*(M-k(r)) ∧ t∈ {l, m, …} ∧
not *creadoEvaluado*(M-k(t))) → ListaPosiblesA (IA-k(i))]

donde i,j,l,m,r,t son nº de UOs y k es el tipo de modelo o actividad.

RPA-4 trata el caso de la planificación de una actividad de integración incremental para un UOi Incrementado:

RPA-4: ∀i∀j ∃k [((1<=k<=3) ∧ *incrementadoDe*(UOi, UOj, {UOl, UOm, …}) ∧
posible (M-k(i)) ∧ *creadoEvaluado* (M-k(j))) → ListaPosiblesA (IA-k(i))]

donde i,j,l,m,r son nº de UOs y k es el tipo de modelo o actividad.

En el ejemplo de la Figura 3.24 se observa un desarrollo posible para UO9, diferente del mostrado en la Figura 3.23.

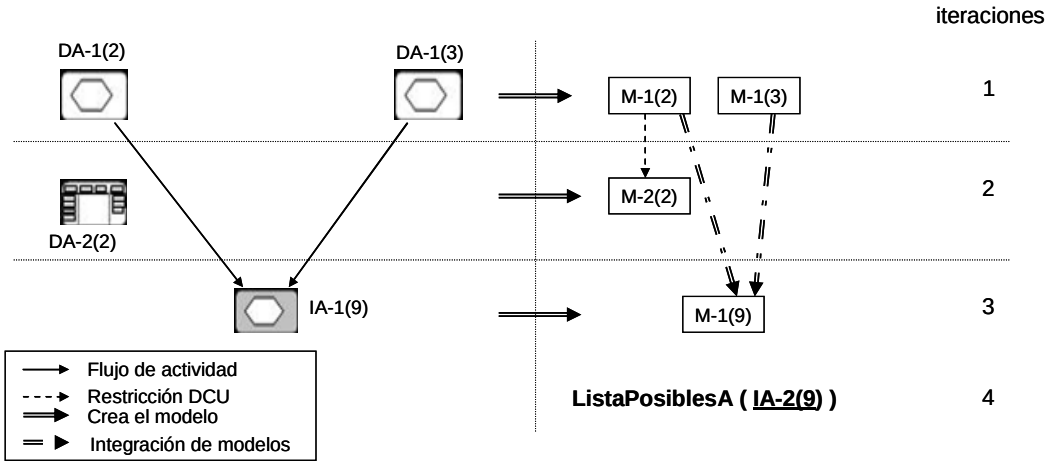


Figura 3.24 Planificación de Actividades de Integración Incremental

En este caso, en la iteración 4 se incluye IA-2(9) en ListaPosiblesA porque:

- *fusionDe*(UO9,{UO2, UO3})

- *creadoEvaluado* (M-2(2)): realizado por la actividad DA-2(2) en la iteración 2. Sin embargo, M-2(3) no se ha efectuado.
- El modelo M-2(9) se encuentra en estado *posible* al estar el modelo M-1(9) en estado *creadoEvaluado* (realizado por la actividad IA-1(9) en la iteración 3).

3.5.2 Plan de actividades: Distribución por equipos

Tras aplicar las reglas de planificación que indican las actividades posibles a realizar, el gestor debe decidir qué actividades de las *posibles* va a realizar cada equipo. Esta decisión permite construir manualmente, el plan de Actividades en la iteración. Esto se expresa mediante la siguiente regla manual de distribución de actividades (RMD):

RMD. $\exists i \exists k$ [((1<=k<=3) $\wedge$ ListaPosiblesA(Actividad-k(i))) $\rightarrow$ plan (Actividad-k(i))]

donde i es el nº del UO y k es el tipo de modelo o actividad

La aplicación manual de esta regla se refleja en la parte de la Hoja de Gestión asociada al Plan (ver Tabla 3.11).

En muchos casos es posible que varias reglas se apliquen sobre un mismo UO. Si existe la posibilidad de realizar DAs e IAs sobre un UO, las estrategias de gestión en InterMod, , aconsejan que se realicen actividades de integración IAs. Por ejemplo, supongamos el caso de la Figura 3.24. Tras la situación en la iteración 3, se aplican dos reglas sobre UO9:

1. La regla RPA-1 y ListaPosiblesA incluye a DA-2(9)
2. La regla RPA-3 y ListaPosiblesA añade IA-2(9)

Por lo tanto, en este momento para UO9:

- ListaPosiblesA (DA-2(9), IA-2(9)).

Las dos acciones tendrían como resultado el modelo M-2(9) al final de su realización y evaluación. Sin embargo, como M-2(9) ya está parcialmente hecho a través de su submodelo, M-2(2), parece razonable aprovechar el trabajo de desarrollo y evaluación realizados. Por tanto, se planificará IA-2(9) , tal y como se muestra en el desarrollo de la Figura 3.25.

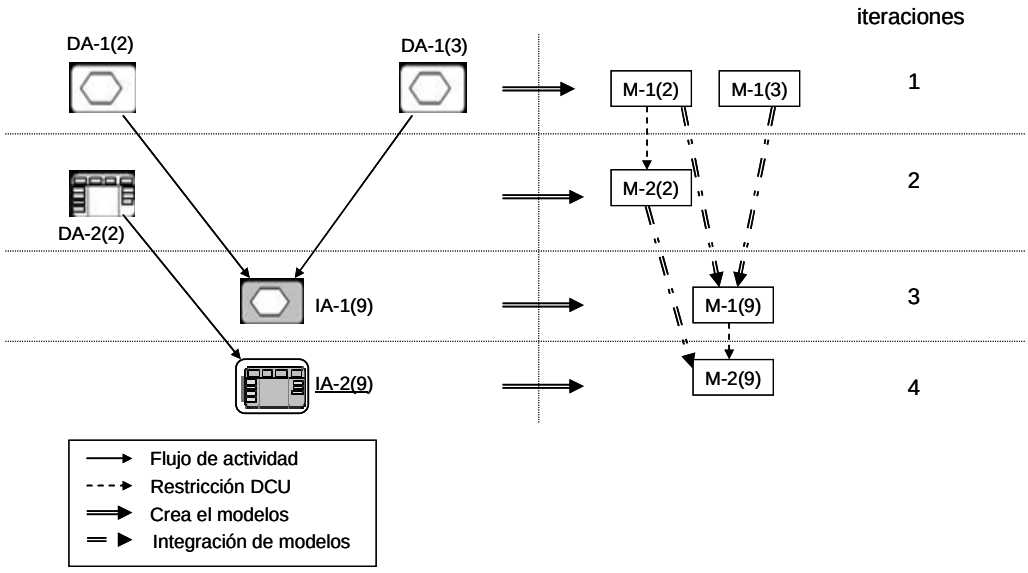


Figura 3.25 Ejemplo de precedencia de actividades de integración en un proceso de planificación

3.6 Resumen

En este capítulo se describen con detalle algunos aspectos característicos de la metodología InterMod: los Objetivos de Usuario, las Actividades, los Diagramas de Gestión y del Proceso, la Hoja de Gestión, las Reglas de Planificación, etc.

Un Objetivo de Usuario es o deriva de un deseo de usuario y es la medida del proceso ágil. A lo largo del mismo pueden producirse fusiones, divisiones y ampliaciones de UOs, y esto ha dado lugar a una tipología diversa de UOs en un proyecto. En la Tabla 3.12 se recogen los tipos de UOs, su representación, su origen (Directo-D, Indirecto-I, Ambos-A), su autonomía (Completa-C, Incompleta-I), y su uso como UO Reutilizado (Reutilizado-R, No reutilizado-N).

El desarrollo de UOs se realiza mediante dos tipos de actividades: Actividades de Desarrollo e Integración. Cada tipo, a su vez, tiene tres subtipos de actividades cuyo fin es la realización y evaluación de los modelos para el desarrollo de un UO: M-1.*Modelo de Requisitos*, M-2.*Modelo de Presentación* y M.3.*Modelo de Funcionalidad*. La perspectiva DCU de InterMod exige que cada UO se desarrolle mediante la realización y evaluación de estos tres modelos en un orden determinado. Esto es, una vez obtenidos y evaluados los requisitos de un determinado UO (M-1), su interfaz debe ser generada y validada (M-2) antes

de su codificación (M-3). Sin embargo, considerando en su globalidad un proyecto complejo compuesto de diferentes UOs, la relación de orden entre las actividades de diferentes UOs está determinada únicamente por el interés ágil, derivado del progreso del proyecto o del interés del usuario.

Tabla 3.12 Tipos de UOs y su representación

UO	REPRESENTACIÓN	Origen	Autonomía	Reutilización
Directo		D	C	N
Indirecto división		I	C	N
Indirecto fusión		I	C	N
Incremento		A	I	N
Incrementado Directo		D	C	N
Incrementado Indirecto		I	C	N
Reutilizado Directo		D	C	R
Reutilizado Indirecto		I	C	R

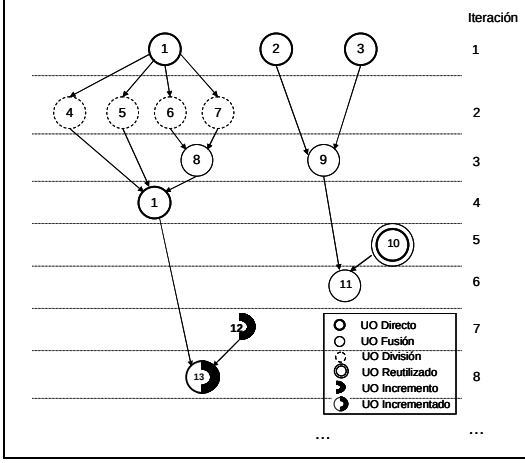
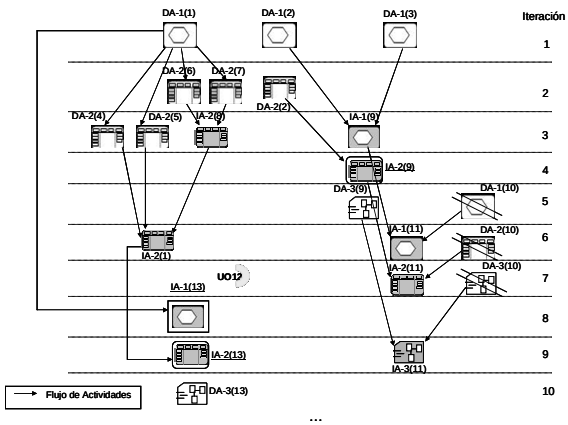
Este capítulo explica con detalle el proceso ágil InterMod. Un proyecto comienza con el Paso 0. *Análisis del proyecto global,* y continúa con un conjunto de iteraciones. Cada una tiene tres pasos: Paso 1.i-*Construir la Lista de UOs*, Paso 2.i-Planificar la iteración paralela y Paso 3.i-*Realizar Actividades por equipos*. Para planificar las actividades se han diseñado un conjunto de reglas que determinan las actividades posibles a realizar. Esta información ayuda a establecer un plan de realización y distribución de actividades entre los equipos, siguiendo los criterios de calidad adquiridos por la perspectiva DCU de InterMod. Estas reglas de planificación de Actividades se resumen en la Tabla 3.13.

Tabla 3.13 Tabla resumen de las Reglas de Planificación de Actividades

REGLAS de PLANIFICACIÓN de ACTIVIDADES	
$\forall i \exists k$ [((1<=k<=3) $\wedge$ *posible*(M-k(i))) $\rightarrow$ ListaPosiblesA(DA-k(i))]	**(RPA-1)**
$\forall i \forall r \exists k$ [((1<=k<=3) $\wedge$ *fusionDe*(UOi, {UOl,UOm,…}) $\wedge$ r$\in$ {l, m, …} $\wedge$ *creadoEvaluado*(M-k(r)) $\wedge$ *posible* (M-k(i))) $\rightarrow$ ListaPosiblesA(IA-k(i))]	**(RPA-2)**
$\forall i \exists r \exists t \exists k$ [((1<=k<=3) $\wedge$ *fusionDe*(UOi, {UOl,UOm,…}) $\wedge$ *posible* (M-k(i)) $\wedge$ r$\in$ {l, m, …} $\wedge$ *creadoEvaluado*(M-k(r)) $\wedge$ t$\in$ {l, m, …} $\wedge$ not *creadoEvaluado*(M-k(t))) $\rightarrow$ ListaPosiblesA (IA-k(i))]	**(RPA-3)**
$\forall i \forall j \exists k$ [((1<=k<=3) $\wedge$ *incrementadoDe*(UOi, UOj, {UOl, UOm, …}) *posible* (M-k(i)) $\wedge$ *creadoEvaluado* (M-k(j))) $\rightarrow$ ListaPosiblesA (IA-k(i))]	**(RPA-4)**

Siendo: i,j,l,m,r,t es el nº del UO - k el tipo de modelo o actividad

Como ayuda para el seguimiento del proyecto, InterMod propone varios diagramas que permiten visualizar su evolución por UOs creados, por actividades comenzadas, en desarrollo o finalizadas. La Hoja de Gestión se va rellenando iteración a iteración, e incluye información sobre el Estado del Proyecto: los proceso de fusión, división e incremento de los UOs y la tabla de modelos creados, y la Planificación de Actividades por equipos.

Diagrama de Creación de UOs	*Diagrama de Actividades*
	

Hoja de Gestión: Estado del Proyecto- Información del proceso de UOs

UOs	**UO1**	**UO2**	**UO3**	**UO4**	**UO5**	**UO6**	**UO7**	**UO8**	**UO9**	**UO10**	**UO11**	**UO12**	**UO13**
Fusión		**9**	**9**	**1**	**1**	**8**	**8**	**1**	**11**	**11**			
División				**1**	**1**	**1**	**1**						
Increm.	**13**											**13**	

Hoja de Gestión: Estado del Proyecto-Tabla de Creación de Modelos

UOs	**UO1**	**UO2**	**UO3**	**UO4**	**UO5**	**UO6**	**UO7**	**UO8**	**UO9**	**UO10**	**UO11**	**UO12**	**UO13**
M-1	X1	X1	X1						I3	X5	I6		I8
M-2	I6	X2		X3	X3	X2	X2	I3	I4	X6	I7		I9
M-3									X5	X7	I9		X10

Hoja de Gestión: Planificación de Actividades

Iteraciones		1	2	3	4	5	6	7	8	9	10
Lista de UOs		1 2 3	4 5 6 7	**8** **9**	1	10	**11**	~~12~~	~~**13**~~		
PLAN	1er. equipo	DA-1(1)	DA-2(4) DA-2(5) DA-2(6)	DA-2(4) DA-2(5)			IA-2(1)			IA-2(13)	
	2º equipo	DA-1(2)	DA-2(2)	IA-1(9)	IA-2(9)	DA-3(9)	IA-1(11)	IA-2(11)	IA-3(11)	IA-3(11)	
	3er. equipo	DA-1(3)	DA-2(7)	IA-2(8)		DA-1(10)	DA-2(10)	DA-3(10)	IA-1(13)		DA-3(13)

En el siguiente capítulo se describen los modelos implicados en el desarrollo de un proyecto InterMod, y aspectos relacionados con ellos como las propuestas de evaluación para cada tipo de modelo y la propuesta de gestión de modelos.

CAPÍTULO 4. Desarrollo y Gestión de Modelos en InterMod

4.1 Introducción

Las tecnologías de programación clásicas como J2EE, .NET y CORBA contienen cientos de clases y métodos con múltiples dependencias y efectos colaterales que hacen que la programación requiera un considerable esfuerzo. Además, su cambio y evolución continua precisan labores de mantenimiento y recodificación manual de software para adaptar el código de las aplicaciones existentes (Schmidt, 2006). Debido a estas dificultades, ligadas al proceso de codificación, el Desarrollo de software Dirigido por Modelos (DDM) resalta la importancia del modelado, o construcción de modelos abstractos, frente a la programación.

DDM tiene el potencial de incrementar la productividad del desarrollo y su calidad, describiendo los aspectos importantes de una solución mediante abstracciones más usables que las utilizadas en la codificación de un lenguaje de programación. Para que este incremento de productividad sea posible, las herramientas de desarrollo de software deben facilitar las tareas de construcción y transformación de modelos. InterMod propone generar y evaluar modelos tal y como indica el Object Management Group's Model-Driven

Architecture (MDA, n.d.). La información de unos modelos alimenta a otros hasta llegar a la configuración del producto final en una plataforma concreta. InterMod se centra en una descripción específica y detallada del modelo de requisitos M-1, y muestra el resto de modelos con una descripción general en el contexto de su relación con él (Losada et al., 2009a) (Losada et al., 2009c).

Este capítulo aborda, en primer lugar y de manera global, la descripción y validación de los modelos en InterMod, así como su relación e interdependencias. A continuación, describe con detalle el modelo abstracto de requisitos, así como una implementación concreta del mismo. Además, se incluye un conjunto de reglas para la Gestión de modelos. La actualización de los estados de los modelos mediante estas reglas de gestión, junto con las reglas de planificación (ver capítulo 3), aseguran un progreso del proyecto con criterios de calidad, formalidad y adaptadas al usuario final. Finalmente, se discuten algunos trabajos relacionados.

4.2 Necesidades de modelado en un proyecto

InterMod propone dos modelos generales que aseguran la consistencia de la aplicación y tres grupos de modelos, fuertemente relacionados entre sí, para el desarrollo de cada UO.

Estos modelos, junto con las Actividades que los crean y evalúan, se muestran en la Figura 4.1. Los modelos generales son el *Modelo del Usuario* y el *Modelo del Sistema*. Los modelos implicados directamente en el desarrollo de cada UO son el *Modelo de Requisitos* (M-1), el *Modelo de Presentación* (M-2) y el *Modelo de Funcionalidad* (M-3), que aparecían en el capítulo anterior en el contexto del modelo de proceso propuesto.

Los Modelos del Usuario y del Sistema contienen la información, común a toda la aplicación, que condiciona el proyecto. Estos modelos se establecen especialmente al principio del proyecto y se van refinando a lo largo del proceso de desarrollo. Los modelos implicados en el desarrollo de cada UO asumen las restricciones y características de los modelos generales del proyecto. El Modelo SE-HCI (*Semantically Enriched Human-Computer Interaction*) recoge para cada UOi su correspondiente modelo de requisitos, influenciado parcialmente por el modelo del Usuario y Sistema ya que únicamente se tienen en cuenta en este modelo las características independientes de la plataforma. Cada UOi tiene

un Modelo de Presentación y un Modelo de Funcionalidad, interrelacionados y alimentados por su correspondiente Modelo SE-HCI, y en general, por el modelo del Usuario y del Sistema en sus características concretas asociadas a un usuario y plataforma concreta.

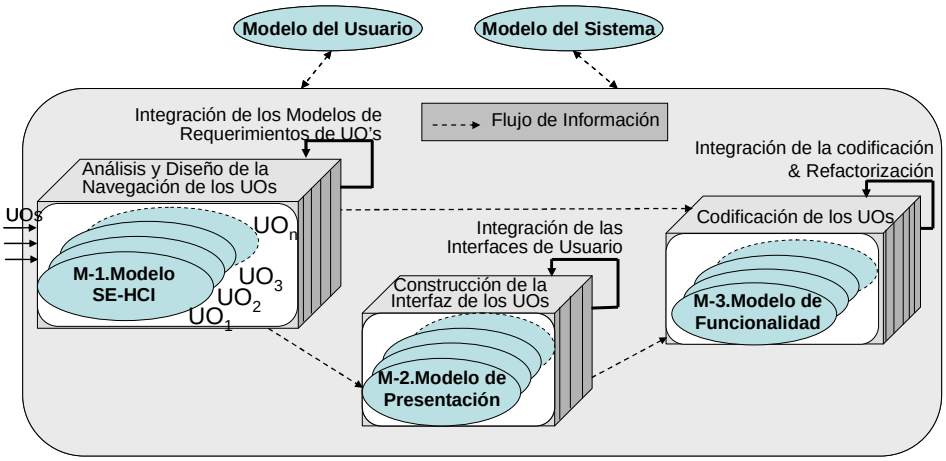


Figura 4.1 Modelos Generales y de Desarrollo en InterMod

4.2.1 Modelos generales: Modelos de Sistema y Usuario

El *Modelo de Sistema* y el *Modelo de Usuario* proporcionan la coherencia en el proyecto. Las decisiones globales de diseño unifican el proceso incremental de creación de la aplicación. Por ello, estas características se recogen para adaptar la aplicación a las características del usuario y del sistema:

- El ***Modelo de Sistema*** recoge las características que definen el tipo de plataforma (por ejemplo, el tipo de dispositivo, el tamaño de la ventana, el color, etc.). Además, este modelo incluye elementos de la aplicación en general, como la seguridad, el nivel de accesibilidad, el logo, o incluso descripciones de servicios web, en el sentido de la propuesta de Paternò (Paternò et al., 2011).
- El ***Modelo de Usuario*** recoge las características que definen el tipo de usuarios (por ejemplo, las preferencias de color, fuente, tamaño, posibles limitaciones en la interacción con la aplicación como el daltonismo, sordera, visión limitada, etc.).

Puede ocurrir que existan situaciones de conflicto en características comunes definidas en el *Modelo del Usuario* y en el *Modelo del Sistema*. Por ejemplo, supongamos el caso de un sistema que limite a un tamaño determinado los componentes del diseño gráfico (ventanas, botones, etc.) como el caso de dispositivos móviles de reducido tamaño, y el modelo de usuario exija un determinado tamaño en dichos elementos. En este caso existe un conflicto determinado por el hardware que condiciona la solución de diseño final. Todos los UOs recogen las características establecidas en estos modelos. Sin embargo, a lo largo del proyecto se pueden modificar algunos aspectos de los modelos de usuario y sistema que refinan el resultado final de la aplicación.

4.2.2 Modelos de Desarrollo de un UO: Requisitos, Presentación y Funcionalidad

Los modelos directamente implicados en el desarrollo de cada UO son los siguientes:

1. El ***Modelo de Requisitos***: Es un conjunto de modelos que recoge las necesidades de un UO, y que en InterMod se agrupan en el modelo ***M-1. Modelo SE-HCI*** (*Semantically Enriched Human Computer Interaction*). La especificación del modelo SE-HCI cubre algunos de los objetivos primordiales de InterMod: (1) Mejorar la captura de requisitos acercando su validación a los usuarios finales, y (2) Revisar aspectos de interfaz y lógica de negocio en etapas tempranas con diseñadores y desarrolladores. En este modelo se trabaja a nivel abstracto, sin implicación total de plataforma (dispositivo) ni lenguaje de implementación concretos. Sin embargo, la evaluación temprana con usuarios tendrá en cuenta algunas características mínimas asociadas con la plataforma destino, como el número de enlaces por pantalla, secciones, etc. En el apartado 4.3 se profundiza en las características de este modelo.
2. El ***Modelo de Presentación***: El ***M-2. Modelo de Presentación*** de un determinado UO establece los elementos gráficos concretos, recogidos parcialmente del M-1.*Modelo de Requisitos* previamente evaluados. Hay varios lenguajes de modelado de interfaces de usuario, ampliamente utilizados y probados, para la descripción a nivel abstracto y concreto, como XML Teresa (Berti et al., 2004a), XIML (XIML.org, n.d.) o UIML (Abrams et al., 1999) que pueden ser usados para reflejar este modelo. En este modelo se

asume una descripción concreta en una plataforma específica, por ejemplo, la interfaz gráfica de un móvil. Además, se indica una técnica de interacción concreta que soporte la interacción abstracta descrita; por ejemplo, que la selección se hace con un botón tipo radio.

3. El ***Modelo de Funcionalidad***: Guía la implementación en un lenguaje de programación concreto. El ***M-3. Modelo de Funcionalidad*** hereda las características de comportamiento del M-1. *Modelo de Requisitos* y completa la funcionalidad del M-2. *Modelo de Presentación* del UO, previamente evaluados. UML (Object Management Group - UML, n.d.) o SysML (SysML Open Source Specification Project | SysML.org, n.d.) son lenguajes alternativos que se usan habitualmente para representar este modelo.

4.2.3 Evaluación de modelos

InterMod promueve el Diseño Centrado en el Usuario, por lo que la evaluación de la usabilidad será primordial. Además, como en toda metodología ágil, los desarrollos deben evaluarse de forma incremental; en este caso, los resultados de los procesos de integración, validados, van conformando la aplicación final.

Los desarrollos asociados a los UOs Directos deben tenerse en cuenta con mucha atención en las evaluaciones, especialmente en las concernientes a la usabilidad, ya que esos UOs representan las tareas que realizarán los usuarios en la interfaz. Los modelos de los UOs Directos deben evaluarse siempre con usuarios finales, mientras que los modelos asociados a UOs Indirectos pueden evaluarse con expertos. De esta forma, los usuarios participarán únicamente en los momentos cruciales, y los desarrollos considerados internos (UOs Indirectos) serán evaluados por miembros del equipo de desarrollo, expertos en el tipo de evaluación implicada.

4.2.3.1 Evaluación de los modelos de Usuario y Sistema

Los modelos de Usuario y Sistema se configuran en su mayoría al inicio del proyecto (Paso 0 en la Figura 3.20, sección 3.4). Los métodos que propone InterMod para su recogida y validación son tomados de las técnicas de indagación empleadas en el ámbito de HCI. Se realizan hablando con usuarios, observándolos o dejándoles responder a preguntas,

verbalmente o por escrito. Los cuestionarios, las entrevistas estructuradas o abiertas, los focus group o la observación de campo (Nielsen and Mack, 1994) son técnicas utilizadas típicamente para ello. Los cuestionarios de perfil de usuario, como los propuestos por Mayhew, son útiles para determinar las características del usuario (Mayhew, 1999). Además, las reuniones entre expertos y desarrolladores ayudan a completar la configuración de los modelos de Usuario y Sistema iniciales.

Las evaluaciones de los modelos de desarrollo de los UOs evalúan también de manera indirecta los modelos de Usuario y Sistema, y como consecuencia se producen ajustes en estos modelos a lo largo del proceso.

4.2.3.2 Evaluación del M-1. Modelo de Requisitos o SE-HCI

Los requisitos se obtienen incrementalmente y se evalúan con métodos de HCI y equipos multidisciplinares. Esta evaluación es un punto clave en la metodología InterMod, ya que los modelos SE-HCI han sido enriquecidos con características de presentación y de funcionalidad que permiten descubrir, de forma anticipada, posibles errores de etapas posteriores. InterMod propone realizar estas evaluaciones con prototipos generados con la información de este modelo, bien en formato electrónico o en papel (Snyder, 2003). Esto favorece la intervención del usuario final, especialmente para los UOs Directos.

La inclusión de equipos multidisciplinares en las evaluaciones, en las que intervienen el usuario final, el diseñador y el desarrollador, asegura que los desarrollos se ajustan por un lado, a las necesidades del usuario final y por otro lado, a los requisitos de la aplicación. InterMod propone métodos de test combinados con el recorrido cognitivo para evaluar este modelo. Típicamente, unos usuarios representativos realizan tareas específicas en el sistema (o prototipo), y los evaluadores observan y recogen la información de la interacción. El recorrido cognitivo (Lewis et al., 1990) se enfoca a la evaluación de la facilidad de aprendizaje y uso de un prototipo de un sistema, por exploración de escenarios. Los principales métodos de evaluación por test aplicables aquí son la medida de las prestaciones (Dumas and Redish, 1993), pensar en voz alta (Lewis, 1982) y el método del conductor (Nielsen, 1993). La medida de las prestaciones se basa en recoger información medible acerca del rendimiento o algún aspecto subjetivo que afecte a la usabilidad del sistema. El tiempo para completar una tarea, el número de opciones de menú erróneas, las observaciones de

frustración o reflexiones sobre la facilidad de uso del producto son ejemplos típicos de estas medidas. Según Dumas y Redish (Dumas and Redish, 1993) los escenarios de tareas pueden ser útiles para las medidas de prestaciones. El *thinking aloud protocol* o pensar en voz alta requiere que el usuario interactúe con el sistema y verbalice al mismo tiempo sus pensamientos, sentimientos y opiniones, cuando efectúa un escenario de tareas. Esto ayuda a los desarrolladores a entender la opinión de los usuarios sobre la aplicación y a identificar fácilmente los problemas mayores de la interacción con los usuarios. Finalmente, el método del conductor se centra en el usuario inexperto mediante una interacción explícita entre él y el evaluador (o conductor). Su objetivo es descubrir las necesidades de información de los usuarios, que lleve a un posible rediseño de la interfaz.

Un escenario, según la definición de Rosson, es "*simply a story about people carrying out an activity*" (Rosson and Carroll, 2001). InterMod adapta esta idea a escenarios de UOs en evaluaciones de tipo test. Definimos un escenario de UO como una historia hipotética, diseñada por el evaluador para cada UO, a realizar por el usuario final a través de la interfaz.

Otras evaluaciones a tener en cuenta en este punto son las comprobaciones de que el modelo SE-HCI se ajusta a las restricciones impuestas en los *Modelos de Usuario y de Sistema*, y que la navegación para alcanzar el Objetivo de Usuario es eficaz y eficiente. El análisis de logs en prototipos software es un método interesante para la evaluación de estas últimas comprobaciones, tanto para el modelo M-1 como para el modelo M-3. Tiene la ventaja de ser cómodo para el usuario ya que puede analizarse de forma remota (Paternò et al., 2007).

4.2.3.3 Evaluación del M-2. Modelo de Presentación

El modelo de Presentación se evalúa para comprobar si las características gráficas del UO se ajustan a las necesidades generales del Usuario y del Sistema. Además, es necesario asegurar que el modelo cumple los criterios de usabilidad. Para esto, InterMod propone emplear métodos de inspección que implican a expertos en usabilidad en el proceso de evaluación. Muchas de las técnicas de inspección se pueden aplicar en prototipos iniciales, mientras que algunas otras también se pueden utilizar en prototipos finales. Sus resultados son una buena fuente de información que sirve para mejorar elementos específicos de la interfaz de usuario. La inspección de estándares y las evaluaciones heurísticas son ejemplos de estos métodos

(Nielsen and Mack, 1994). Un estándar es un requisito, regla o recomendación basado en principios probados y en la práctica, acordado por un grupo de profesionales oficialmente autorizados en un ámbito local, nacional o internacional (Smith, 1996). En la inspección de estándares, el evaluador realiza una inspección minuciosa de la interfaz para comprobar el cumplimiento de todos los puntos definidos en el estándar. Por su parte, la evaluación heurística proporciona un conjunto de heurísticos que utilizan uno o más evaluadores en el chequeo de la interfaz. Los criterios de usabilidad que se utilizan proceden, en su mayoría, de los originales propuestos por Nielsen (Nielsen and Molich, 1990). Estos heurísticos ayudan a identificar un porcentaje alto de problemas de usabilidad. En estos modelos, es interesante también, tener en cuenta las evaluaciones de accesibilidad.

4.2.3.4 Evaluación del M-3. Modelo de Funcionalidad

Estos modelos deben ser evaluados de acuerdo a criterios de calidad de software, tales como la funcionalidad, fiabilidad, eficiencia, mantenibilidad, portabilidad y usabilidad. Esto es, se debería verificar que cumplen las condiciones de un modelo de calidad como el propuesto en ISO/IEC 25000:2005 ("ISO/IEC 25000:2005 - Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE," 2005). Esto se realiza, principalmente, a través de pruebas ejecutables (como es habitual en Ingeniería de Software), con el fin de verificar los resultados de salida esperados a partir de los datos de entrada. Además, tanto los métodos basados en test como los métodos de inspección son adecuados para la evaluación de la usabilidad en este punto. El registro de uso (logging) y la medida de las prestaciones, en combinación con el recorrido cognitivo y los escenarios, son útiles para estimar la eficiencia y eficacia (Tullis and Albert, 2008). Además, la comprobación de la satisfacción del usuario final del producto conseguido, especialmente para los UOs Directos o UOs fusionados que implican a uno o varios UOs Directos, es un aspecto importante que puede hacerse con cuestionarios. La escala *System Usability Scale* (SUS) (Brooke, 1996) ha sido adoptada como una medida de usabilidad estándar de la experiencia de usuario, y consiste en diez sentencias que los usuarios valoran en un rango de cinco valores. Otras propuestas basadas en SUS son el Computer Usability Questionnaire (CSUQ) (Lewis, 2001) y el Post-Study System Usability Questionnaire (PSSUQ) (Lewis, 2002). Ambos son cuestionarios muy similares para evaluar un sistema al final de un estudio de usabilidad;

CSUQ se administra vía online y PSSUQ en persona. Más recientemente, UMUX (Usability Metric for User Experience) (Finstad, 2010) se presenta como una alternativa más compacta, organizada en base a la definición de usabilidad ISO 9241-11.

4.3 Modelo SE-HCI, el soporte estratégico de los requisitos del proyecto

El *modelo SE-HCI (Semantically Enriched Human-Computer Interaction)* describe el modelo de requisitos en InterMod, cuyo objetivo es adelantar y validar con los implicados del proyecto y en etapas tempranas: los requisitos, aspectos parciales de presentación y el comportamiento en la interfaz de la aplicación interactiva.

4.3.1 Formalización del Modelo SE-HCI

El modelo SE-HCI es el núcleo de la metodología propuesta y está influenciado, como el resto de los modelos de desarrollo, por el modelo del Usuario y Sistema. Para cada UO, el equipo de diseñadores formaliza su *Modelo SE-HCI*, que comprende el *Modelo de Tareas de Usuario*, el *Modelo del Prototipo*, el *Modelo de Comunicación* y el *Modelo de Comportamiento*. En la Figura 4.2 se observa este modelo con sus componentes para cada UOi.

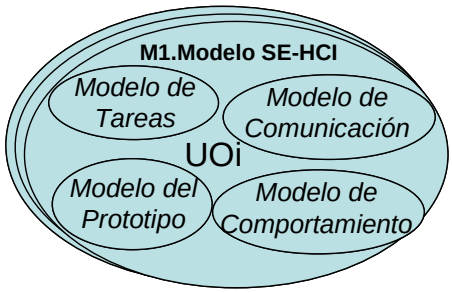


Figura 4.2 Modelo SE-HCI

El ***Modelo De Tareas de Usuario*** es un modelo clásico en el desarrollo de interfaces de usuario basadas en modelos (Puerta, 1997) (Paterno, 2000) (Limbourg, 2004), y describe la actuación correcta del usuario en uno o varios escenarios de la aplicación. En el modelo

propuesto por InterMod, las actuaciones descritas en un modelo de tareas se asocian a escenarios de un UO.

El ***Modelo del Prototipo*** recoge algunas características gráficas básicas, que afectarán al modelo de Presentación, necesarias para obtener prototipos de forma automática.

El ***Modelo de Comunicación*** recoge la comunicación directa Sistema-Usuario. Se incluye aquí la descripción de las acciones que el sistema lleva a cabo a nivel de interfaz de usuario, durante una sesión interactiva, así como sus relaciones temporales. Es decir, el *Modelo de Comunicación* incluye aquellas comunicaciones en las que el sistema se comunica directamente con el usuario mostrando una ventana de error o un mensaje simple. Esto significa que las operaciones del sistema sobre otros elementos en el entorno de la aplicación, por ejemplo una base de datos, no se expresa en el modelo ya que no está implicado en la comunicación directa con el usuario.

El ***Modelo del Comportamiento*** incluye el efecto en la navegación de las interacciones correctas e incorrectas. Ambos tipos de interacciones expresan las diferentes ejecuciones del UO. Es decir, esta información configura el *Modelo Comportamiento* que, junto con la descripción de las acciones del usuario (*Modelo de Tareas de Usuario*) y la descripción de las acciones del Sistema (*Modelo de Comunicación*), representa la semántica de la aplicación a través de interfaz de navegación. Gracias a los modelos que incluye, el Modelo SE-HCI adelanta a etapas tempranas la detección de errores en el modelado de la interfaz y de la lógica de negocio. Esto es posible porque sus características permiten agilizar su evaluación con los implicados en el desarrollo del proyecto, incluido el usuario final. Esto permitirá observar y valorar en un prototipo la semántica y un aspecto primitivo de la aplicación, y posteriormente guiar al *Modelo de Funcionalidad* de la aplicación.

La Figura 4.3 muestra el metamodelo SE-HCI, de una manera formal, en el que se observa las dependencias del modelo SE-HCI con los modelos que engloba. Los *Modelos de Prototipo y de Comportamiento* proporcionan al Modelo SE-HCI (y también al modelo de Tareas de Usuario) información suficiente como para generar prototipos de forma automática. La formalización de los UOs mediante el modelo SE-HCI, permite centrar el foco de atención en los aspectos de la interacción Usuario-Sistema a nivel de interfaz de usuario. Esto es útil para

la visualización y la evaluación temprana del comportamiento final del UO con los implicados.

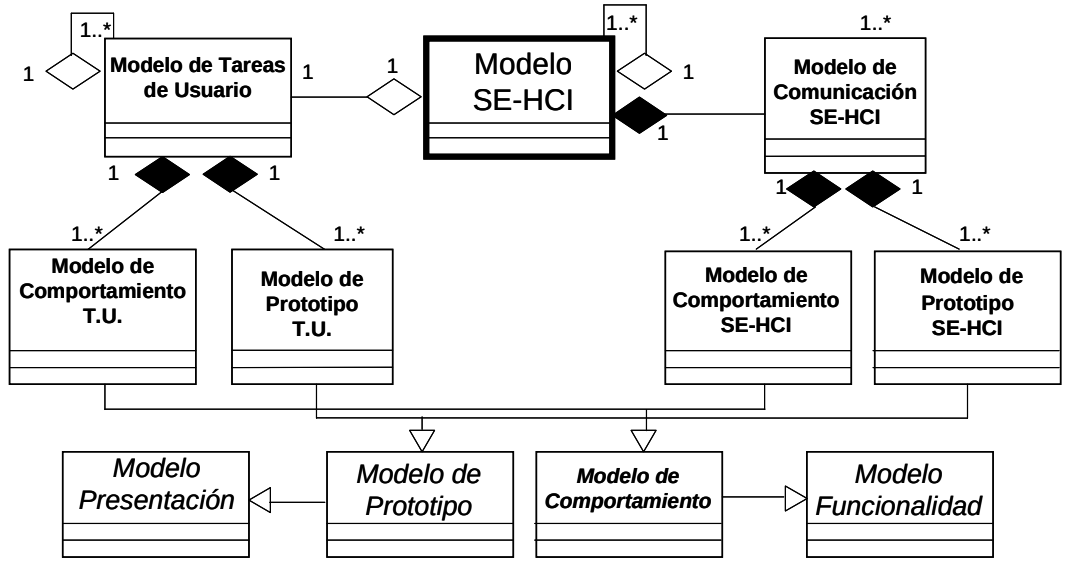


Figura 4.3 Metamodelo SE-HCI

En cuanto al medio adecuado para la evaluación, ya que la información almacenada en los modelos SE-HCI facilita la generación de prototipos, éstos son la propuesta de InterMod (Figura 4.4).

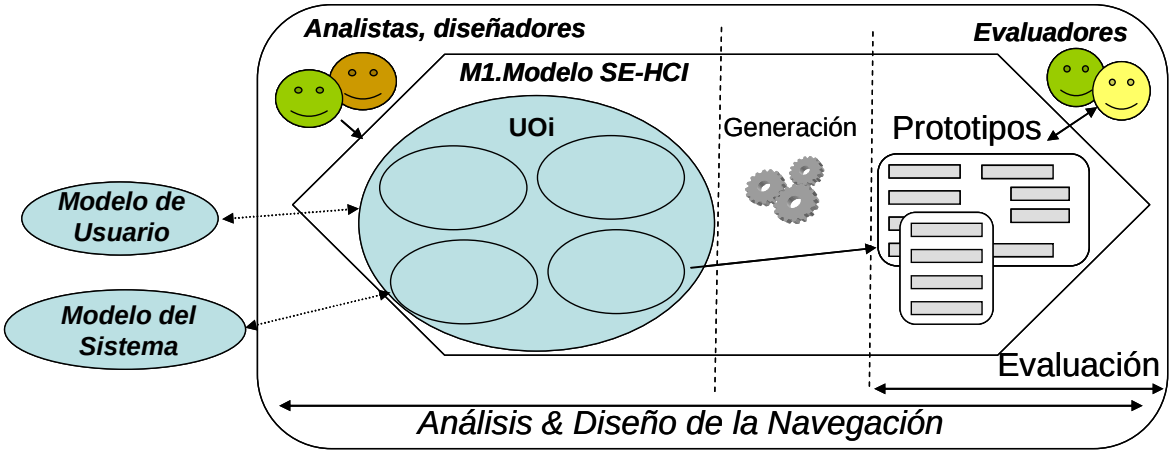


Figura 4.4 Generación de Prototipos a partir del modelo SE-HCI

Al igual que Wiegers pienso que es difícil visualizar exactamente cómo se comportará el software mediante la lectura textual de los requisitos o estudiando modelos de análisis. Los usuarios están más dispuestos a probar un prototipo que a leer un documento (Wiegers, 2003). El proceso sería el siguiente: los analistas y diseñadores gráficos completan el modelo SE-

HCI para cada UO teniendo en cuenta la información de usuario y sistema recogida previamente. Este modelo se valida mediante prototipos y evaluadores multidisciplinares donde la opinión del usuario final es primordial, sobre todo en la evaluación de los UOs Directos. La presencia en las evaluaciones de los diseñadores y desarrolladores de la aplicación permite validar en este punto, aspectos gráficos y funcionales.

4.3.2 WebDiagram: Una implementación del Modelo SE-HCI

WebDiagram es una herramienta interactiva creada para ayudar a los analistas y diseñadores a registrar y evaluar el modelo SE-HCI de aplicaciones en desarrollo. Su evolución y el lenguaje de descripción utilizado se describen en los Apéndices A y B.

4.3.2.1 Justificación de la notación utilizada en WebDiagram

En contraste con la mayoría de las conversaciones humanas, el diálogo Hombre-Máquina es más exigente, con una estructura más rígida y restringida que hace necesario la utilización de una notación formal para su modelado. Las justificaciones para el uso del modelo de diálogo (Dix et al., 2003) son: (1) facilitar el análisis del diálogo, (2) separar los elementos de la interfaz y de la lógica de negocio, (3) permitir al diseñador analizar la estructura propuesta, e incluso tal vez, permitirle usar una herramienta de prototipado para ejecutar el diálogo, y (4) Favorecer la comunicación entre los miembros de un equipo, ya que les permite debatir sobre aspectos de diseño y, llegados a un consenso, trasmitir el modelo de diálogo.

Las notaciones formales usadas para la descripción del diálogo Hombre-Máquina pueden agruparse en:

- Esquemáticas, mediante diagramas que permiten al diseñador observar de un vistazo la estructura del diálogo (Dix et al., 2003).
- Textuales, mediante descripciones textuales que detallan los eventos sistema-usuario.

En la elección de la notación utilizada en WebDiagram, se han valorado siete características que permiten establecer el potencial de las notaciones, su legibilidad y su independencia sobre

una filosofía concreta de codificación de la funcionalidad (Losada et al., 2010). Así, se valora lo siguiente:

1. La posibilidad de describir diálogos concurrentes
2. La posibilidad de describir accesos generales, como Home o Help
3. La legibilidad de la notación
4. Las posibilidades de agrupación semántica de los subdiálogos
5. La descripción del orden y semántica de los eventos
6. La independencia de la notación de una filosofía de programación
7. La posibilidad de utilizar directamente la notación como comunicación entre las partes implicadas en el desarrollo del producto

En la Tabla 4.1 se presenta una comparativa de varias notaciones esquemáticas, y se destacan con una ✓ las características que poseen, y con una ✗ aquéllas que no se dan. En sí mismas, ninguna de las propuestas es una buena herramienta de comunicación (Puerta, 1997), aunque en su mayoría por su simplicidad, tienen una legibilidad adecuada (Larman, 2003).

Tabla 4.1 Comparativa de notaciones esquemáticas

	STN	STN de Harel	Grafos	Petri Nets	Flujos Tradicionales	Árboles	Diagramas UML
Concurrencia	✗	✓	✓	✓	✗	✗	✓
Accesos Generales	✗	✗	✗	✓	✗	✗	✓
Legibilidad	✓	✓	✓	✗	✓	✓	✗
Subgrupos	✗	✗	✓	✗	✗	✓	✗
Orden, semántica	✓	✓	✓	✓	✓	✓	✓
Independencia de la codificación	✓	✓	✓	✓	✗	✓	✗
Herramienta de Comunicación	✗	✗	✗	✗	✗	✗	✗

Dentro de las notaciones textuales distinguimos dos grupos. Por una parte, las que derivan de una estructura formal (gramáticas, reglas de producción o las álgebras de proceso) y, por otra parte, las de aspectos narrativos (casos de uso y escenarios). El primer grupo destaca por permitir la descripción formal y precisa de los acontecimientos. Sin embargo, estos aspectos van en detrimento de la legibilidad y por lo tanto, dificultan su uso como herramienta de comunicación. El segundo grupo, por el contrario, contiene herramientas de comunicación útiles y directas (Rosson and Carroll, 2001) pero son poco formales y pueden contener descripciones ambiguas.

El modelo SE-HCI no sólo debe recoger los requisitos funcionales de la aplicación, sino que, además, debe favorecer el desarrollo de otros modelos en etapas tempranas. Para conseguir alcanzar los objetivos marcados es necesario paliar los inconvenientes del formalismo excesivo. Para ello se propone una notación de modelado del diálogo esquemático con aspectos textuales que faciliten la comunicación. El formalismo escogido es el de los grafos visualizados como árboles pero permitiendo accesos generales y descripciones de concurrencia. Además con el objetivo de agilizar los desarrollos, se incluye en la fase temprana del modelado del diálogo, elementos que simulen la semántica del programa y el aspecto final del producto. Se propone obtener prototipos cercanos al resultado final y convertirlos en herramientas para verificar los requisitos, funcionales y no funcionales, entre el usuario y los desarrolladores.

4.3.2.2 Implementación concreta del modelo SE-HCI en WebDiagram

La Figura 4.5 muestra la implementación concreta de la especificación expuesta anteriormente del modelo SE-HCI, realizada para la herramienta WebDiagram. Aunque se han hecho algunas pruebas para recoger la información del *Modelo de Usuario* y del *Modelo del Sistema* acerca de algunas características y limitaciones (número, tipo y tamaño de los botones, el número y tamaño de las ventanas, colores, etc) en función de los usuarios y dispositivos, no se ha profundizado en ese tema, y la versión en curso únicamente parametriza estas características.

En cuanto al *Modelo de Tareas de Usuario*, el concepto base es la *Tarea del Usuario*, que permite capturar las acciones del usuario, mediante una estructura en árbol formada por tareas hijas y hermanas. Cada tarea complementa su información con el *Orden* de las tareas respecto

a sus hermanas (secuencial, indiferente, elección), y con el *Tipo* de tarea (Unitario, opcional, repetitivo) que establece si es obligatorio llevar a cabo la tarea y el número de actuaciones necesarias. Este modelo soporta las tareas concurrentes que se realizarán en paralelo con otras tareas de usuario. Las tareas concurrentes no tienen orden ni tipo ya que pueden efectuarse en cualquier momento y las veces que se desee. Además, permite desglosar un modelo en varios submodelos.

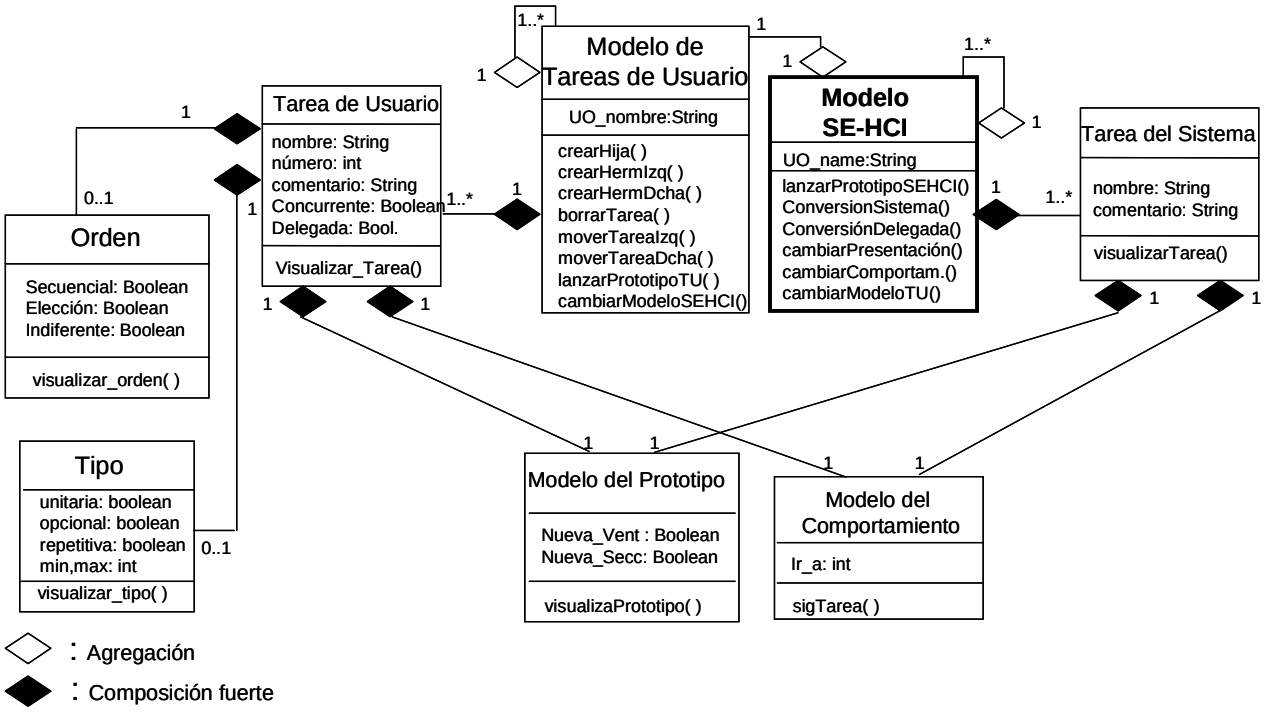


Figura 4.5 Metamodelo SE-HCI de WebDiagram

El *Modelo de Comunicación* incorpora al *Modelo SE-HCI* las *Tareas del Sistema* que expresan la comunicación directa del sistema en respuesta a la tarea del usuario, a nivel de interfaz. Por su parte, el *Modelo de Comportamiento* indica cuál es la siguiente tarea a realizar para cada tarea. Para ello, se realiza un recorrido en pre-orden de la estructura en árbol, pero además, se considera el tipo de la tarea y su orden, y también la desviación en el caso de interacción incorrecta. El *Modelo de Prototipo,* muy simple*,* permite visualizar las subtareas en una ventana nueva o en una nueva sección.

La herramienta WebDiagram (Figura 4.6) dispone de los recursos necesarios para la implementación de este metamodelo. La parte central de la Figura 4.6 muestra el modelo SE-

HCI, desarrollado para UO4 de la aplicación *WebButcher*, con WebDiagram 3.0. (UO4-Comprar filetes al corte:…).

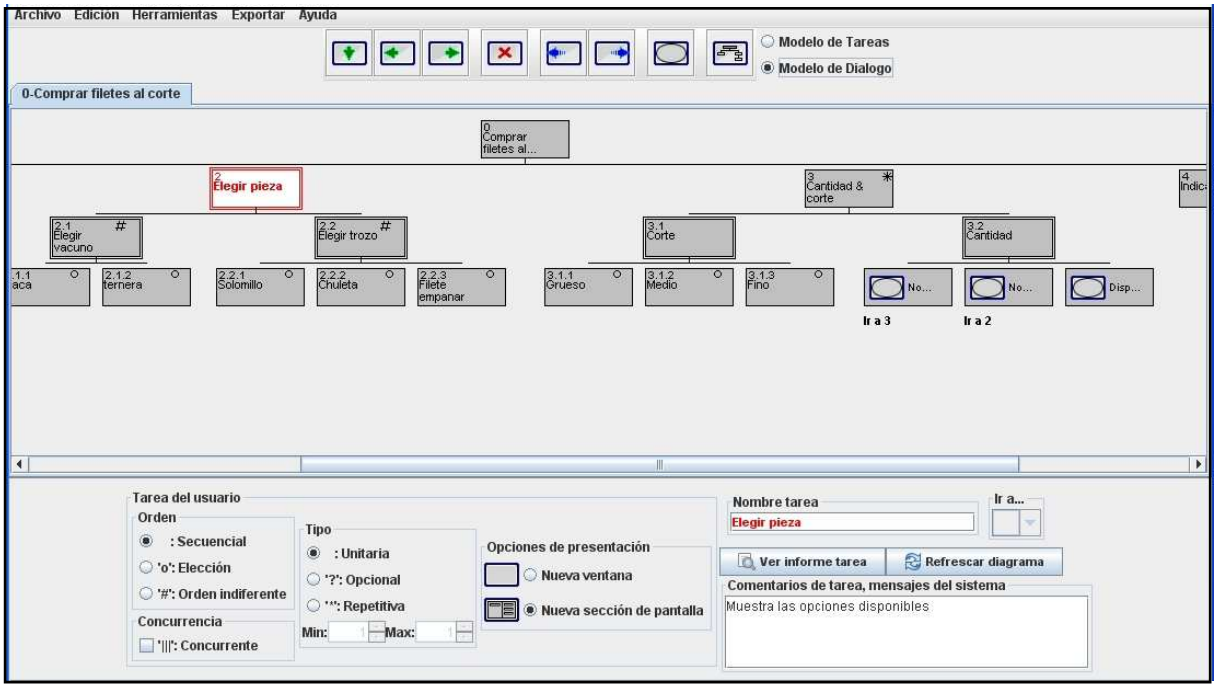


Figura 4.6 Modelo SE-HCI en WebDiagram 3.0 para M-1(4) de *WebButcher*

Para representar fácilmente el diagrama del modelo SE-HCI, WebDiagram usa una representación jerárquica simple, en concreto HTA (Annett and Duncan, 1967) con una notación JSD ampliada (Jackson, 1975).

La notación gráfica que se utiliza para recoger las características de los modelos que engloba se visualizan en la Figura 4.7. Cada tarea se representa mediante una rectángulo que incluye diferentes símbolos. Las tareas del usuario y las del sistema se diferencian en que estas últimas incluyen una imagen de un círculo inscrito en otro renctángulo. Los dos símbolos en la parte superior derecha corresponden a las características de orden (primer símbolo) y de tipo (segundo). En la parte izquierda de las tareas del usuario aparece el número de tarea definido como el número de su tarea padre '.' número secuencial de creación (Losada et al., 2009c)

	**Tarea de Usuario** (acción de usuario) **Tipo de tarea**: Unitaria (sin símbolo), Opcional (?) y Repetitiva (*) **Orden**: Secuencial (sin símbolo), Elección (o), Indiferente (#)
	**Tarea del Sistema** (respuesta del Sistema)
	Cambio de Orden de Navegación (siguiente)
	**Presentación de las Hijas:** Nueva ventana (recuadro simple)
	**Presentación de las Hijas:** Nueva sección, misma ventana (recuadro doble)
	Tarea Concurrente

Figura 4.7 Notación gráfica empleada en la herramienta WebDiagram 3.0

Para recoger la información de los modelos SE-HCI generados, WebDiagram utiliza un lenguaje interno basado en XML, tal y como se muestra en la Figura 4.8. Esta notación XML facilita el paso modelo a modelo, para la creación del modelo de presentación y funcionalidad.

```
- <tareasDesglosadas>
  - <tareas nuneracionInicial="0" profundidadDesglose="0">
    - <tarea max="1" min="1">
        <id>0</id>
        <nombre>Comprar filetes al corte</nombre>
        <padre>-1</padre>
        <tipo>0</tipo>
        <comentarios/>
      - <hijos>
          <hijo>8</hijo>
          <hijo>1</hijo>
          <hijo>2</hijo>
          <hijo>3</hijo>
        </hijos>
        <codigo>0</codigo>
        <desplegable>false</desplegable>
        <nuevaSeccion>false</nuevaSeccion>
        <tareaSistema>false</tareaSistema>
        <siguiente/>
        <desglosada>false</desglosada>
        <raiz>true</raiz>
      </tarea>
```

Figura 4.8 Notación XML empleada en la herramienta WebDiagram 3.0.

Además, la herramienta permite traducir este lenguaje a XML Teresa (Berti et al., 2004a), UIML ("OASIS User Interface Markup Language (UIML) TC," n.d.), y XIML("XIML.org," n.d.) como se observa en la Figura 4.9, Figura 4.10 y Figura 4.11.

```
<TaskModel NameTaskModelID="C:\Tesis\Memoria\FIGURAS\WebDiagram3.0\ComprarFiletes.xml">
  <!--Exportado desde WebDiagram-->
  <!--Dia: mié, 17 abr 2013-->
 -<Task Category="Interaction Task" Frequency="null" Identifier="Comprar filetes al corte ID:0" Iterative="false"
  Optional="false" PartOfCooperation="false">
    <Name>Comprar filetes al corte </Name>
    <Type></Type>
    <Description></Description>
    <Precondition></Precondition>
   -<TimePerformance>
      <Max></Max>
      <Min></Min>
      <Average></Average>
    </TimePerformance>
   -<SubTask>
     -<Task Category="Interaction Task" Frequency="null" Identifier="webcam carniceria ID:8" Iterative="false"
      Optional="false" PartOfCooperation="false">
        <Name>webcam carniceria </Name>
        <Type></Type>
```

Figura 4.9 Transformación XML Teresa del Modelo SE-HCI para UO4, construído con WebDiagram 3.0

```
<uiml>
  <!--Exportado desde WebDiagram-->
  <!--Dia: mié, 17 abr 2013-->
  <head>Comprar filetes al corte</head>
 -<interface>
  -<structure>
   -<part class="G:TopContainer" id="P0">
    -<style>
      <property name="g:title" part-name="P0">Comprar filetes al corte</property>
     -<property name="g:size" part-name="P0">
        <width>600</width>
        <height>600</height>
      </property>
      <property name="description" part-name="P0">Comprar filetes al corte</property>
    </style>
   -<part class="G:Text" id="rut0">
    -<style>
     -<property name="g:size" part-name="rut0">
        <height>30</height>
```

Figura 4.10 Transformación UIML del Modelo SE-HCI para UO4, construído con WebDiagram 3.0

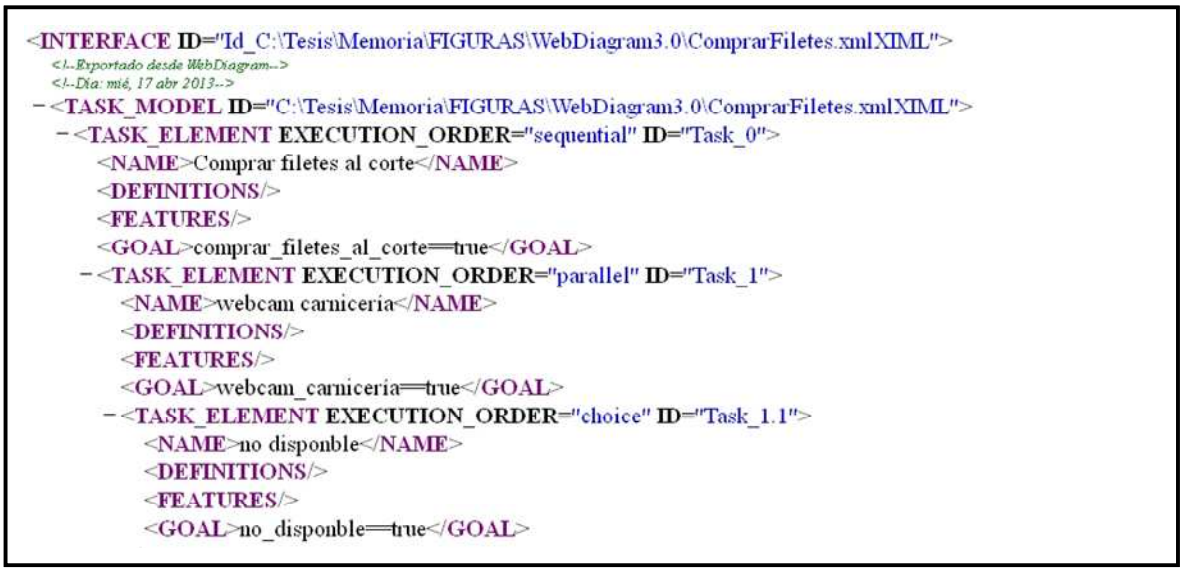


Figura 4.11 Transformación XIML del Modelo HTA para UO4, construído con WebDiagram 3.0

Estas notaciones únicamente pueden cubrir las descripciones del modelo de tareas de usuario, ya que no disponen de todas las posibilidades del modelo SE-HCI.

4.3.2.3 Uso de prototipos en las evaluaciones del modelo SE-HCI de WebDiagram

Una vez construido el modelo SE-HCI, WebDiagram permite simular este modelo con unos prototipos que muestran las posibles interacciones del usuario y del sistema mediante botones de un tipo general (Losada et al., 2009a). Esto es, las tareas del usuario y sistema se han traducido a botones del mismo tipo y tamaño, y se han incluido en nuevas ventanas o en nuevas secciones de la pantalla, según establezca el modelo de prototipo correspondiente. Las acciones producidas al pulsar los botones (Figura 4.12), permiten simular las características modeladas (tales como la comunicación, navegación, comportamiento y presentación). Este tipo de simulación permite a los diseñadores, usuarios y desarrolladores llevar a cabo la evaluación de forma conjunta. Esto es, gracias a los prototipos generados automáticamente, se facilita la aplicación de los métodos de test propuestos por InterMod en la sección 4.2.3, para la evaluación de este modelo.

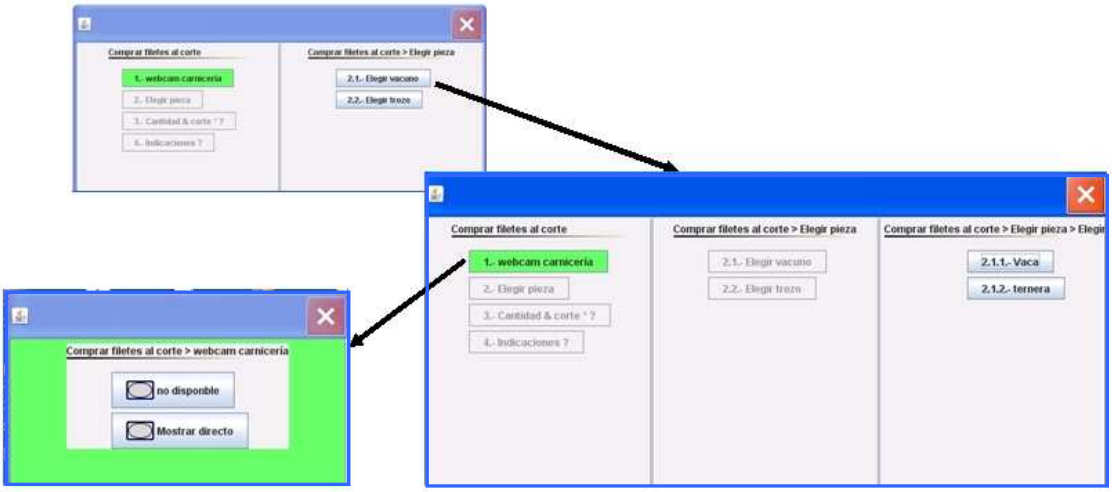


Figura 4.12 Presentación de prototipos generada a partir del modelo SE-HCI del UO4 de *WebButcher*

4.4 Gestión de modelos en el proceso InterMod

Para desarrollar un UO desde la perspectiva Model-Driven es necesario crear y evaluar los siguientes modelos: M-1.*Modelo de Requisitos*, M-2.*Modelo de Presentación* y M-3. *Modelo de Funcionalidad.* Es decir, los modelos correspondientes a UOi son: {M-1(i), M-2(i), M-3(i)}.

Estos modelos pasan por diferentes estados a lo largo de un proyecto:

- ***pendiente*** - es una visualización abstracta del modelo mental del usuario que queda definido con su nombre y descripción. Se crea un modelo (vacío) con ese estado en el mismo momento de la formalización de un UO, o se actualiza a ese estado por necesidades de revisión total del modelo.
- ***posible*** - El modelo pasa a estado *posible* cuando las características de desarrollo del problema aseguren la adecuada coherencia para su creación, según las recomendaciones DCU asumidas en InterMod y explicadas en la sección 3.3.1.2
- ***enDesarrollo*** – El modelo está *enDesarrollo* cuando se ha planificado que algún equipo de desarrollo realice la actividad para la creación y evaluación de dicho modelo. Esta condición se establece en la *Planificación de Actividades* (ver sección 3.5).

- ***creadoEvaluado*** – Un modelo creado se considera *creadoEvaluado* cuando pasa todas las fases de evaluación positiva.
- ***enRevisión*** – El modelo está *enRevisión* cuando está pendiente de revisión debido a la incorporación de nuevas necesidades.

El transcurso habitual de los estados de un modelo para cada UO viene determinado por la secuencia: {*pendiente*, *posible*, *enDesarrollo*, *creadoEvaluado*}. Además, eventualmente, un modelo puede pasar a estado *enRevisión*. En la Figura 4.13 se visualizan todas las reglas de gestión de modelos que provocan las transiciones de estado de los modelos, marcadas con un número.

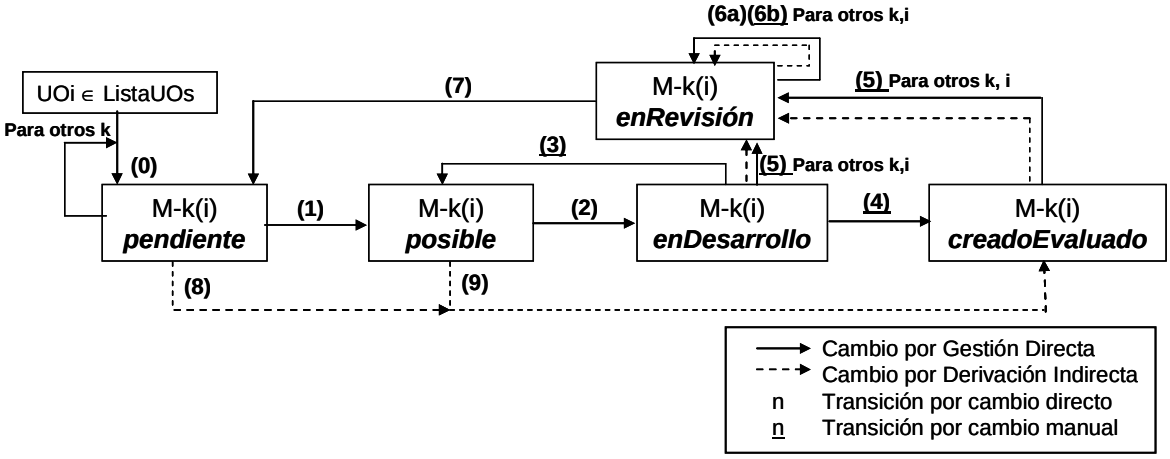


Figura 4.13 Flujo de estado de los modelos en InterMod

Por una parte, están las transiciones o cambios de estado debido a la Gestión Directa de los estados de los modelos, producidos directamente sobre el mismo UO mediante reglas de gestión directa-RGD, mostrados con flechas de línea continua. Por otra parte, están los cambios debidos a la Derivación Indirecta de los modelos de otros UOs, producidos por reglas RDI y mostrados con flechas de línea discontinua. Las reglas RGD y RDI producen transiciones producidas de forma automática (transiciones sin subrayar) y transiciones manuales producidas por criterio del equipo gestor (transiciones subrayadas-RGM).

Dentro de las reglas RDI, se distingue un grupo de reglas que provocan la creación indirecta de modelos y que se denominan RCI.

Los momentos del proceso en los que se producen los cambios de estado de los modelos son los siguientes:

- Paso 1.i: *Construir la Lista de UOs*. La actualización de modelos se realiza después de la construcción de la Lista de UOs. La formalización de nuevos UOs provoca los cambios considerados en la transición (0). Además, se tiene en cuenta el paso a *pendiente* de los modelos *enRevisión* con la transición (7). Finalmente, se tendrán en cuenta las transiciones por creación indirecta RCI (8) producida tras una división de UOs.
- Paso 2.i*: Planificar la Iteración Paralela*. Se planifica la iteración teniendo en cuenta las precedencias DCU de InterMod reflejadas en la transición (1). Una vez aplicadas las reglas de planificación y establecido el plan para cada equipo, las actividades seleccionadas pasarán a estado *enDesarrollo* por la transición (2).
- Paso 3.i: *Realizar las Actividades de la Iteración*. Durante la realización de las actividades, los modelos pueden o no haberse evaluado positivamente; esto es, (3) para el caso de evaluación negativa o no finalización en la creación del modelo, y (4) en caso contrario. Además, se consideran las transiciones (5), (6a) y (6b) para aquellos modelos que pasen a estado *enRevisión*. Tras la evaluación de los modelos, la fusión de UOs y los UOs incrementados provocan la transición por creación indirecta RCI (9).

A continuación, se exponen y explican estas reglas de gestión de modelos, agrupadas por su momento y causa de activación en el proceso. Los ejemplos que se utilizan en este capítulo son desarrollos parciales y/o posibles del proyecto WebButcher. Además de los predicados ya definidos y empleados para explicar las reglas de planificación de actividades (sección 3.5), se usa el predicado *divididoEn*:

divididoEn(UOi, {UOl, UOm, …}): UOi se divide en UOl, UOm,...

4.4.1 Transcurso habitual de los estados de los modelos

Los estados de los modelos pueden modificarse de forma automática a través de reglas de Gestión Directa (RGD) y reglas de Derivación Indirecta (RDI) debido a: la creación de UOs y su inclusión en la ListaUOs (transición 0), la aplicación de la perspectiva DCU de InterMod para el desarrollo de UOs (transición 1), el reparto de actividades (transición 2) y la revisión

de un modelo del mismo o distinto UO (transiciones 6a y 7). Además, en el paso 3.i, se toman decisiones de gestión en las que se actualizan manualmente los estados de los modelos mediante reglas de Gestión Manual (RGM) para el avance del proyecto (transiciones 3, 4, 5) y por necesidades de reutilización (transición, 6b).

4.4.1.1 Paso 1.i. Construir la Lista de UOs

En este paso, los modelos de los UOs nuevos y aquéllos en revisión, pasan a estado *pendiente*.

Modelos pendientes por creación de nuevos UOs

Una vez que un UO ha sido incluido en la ListaUOs, sus tres modelos: {M-1(i), M-2(i), M-3(i)}, quedarán en estado *pendiente*. Esta norma se ejecuta con estas dos reglas de Gestión Directa (transición 0):

RGD-0.1. Si un UOi ha sido incluido en Lista_UOs, su modelo M-1(i) pasa a un estado *pendiente.* Formalmente, se expresa de la siguiente manera:

RGD-0.1. $\forall i$ [UOi$\in$ ListaUOs $\rightarrow$ *pendiente*(M-1(i))]

RGD-0.2. Si un modelo para un UO está pendiente, también lo estarán los modelos de numeración superior para ese UO. Formalmente, se expresa de la siguiente manera:

RGD-0.2. $\forall i \; \forall k$ [($(1 \leq k \leq 2) \wedge$ *pendiente*(M-k(i))) $\rightarrow$ ($j=k+1 \wedge$ *pendiente*(M-j(i)))]

donde i es el nº del UO y j, k son el tipo de modelo

Por ejemplo, en el momento de formalizar e introducir UO2 en Lista_UOs, los tres modelos aparecerán como pendientes: M-1(2) por RGD-0.1, M-2(2) por RGD-0.2 al estar pendiente M-1(2) y, finalmente, M-3(2) por RGD-0.2 al estar pendiente M-2(2).

Cambio de estado por revisión de modelos

El paso de un modelo del estado *enRevisión* a *pendiente* se efectúa mediante la siguiente regla Directa, al inicio de cada iteración (transición 7):

RGD-7.Un modelo que está *enRevisión* pasa automáticamente a estado *pendiente.* Formalmente:

RGD-7. $\forall i \; \forall k \; [\; (\; (1 \leq k \leq 3) \wedge \textit{enRevisión}(\text{M-k(i)}) \;) \rightarrow \textit{pendiente}(\text{M-k(i)}) \;]$

donde i es el nº del UO y k es el tipo de modelo

Este cambio de estado de un modelo asociado a un UOi, puede implicar la reconsideración de otros modelos asociados a ese mismo UO. Por ejemplo, supongamos que todos los modelos del UO2 están en estado *creadoEvaluado*, y que al finalizar una iteración se considera necesario revisar el modelo M-2(2). Es decir, pasa a un estado *enRevisión*, transición (5). En el paso 1.i y por la regla RGD-7, vuelve a cambiar de estado a *pendiente*. En ese paso de la metodología, se aplicará también la regla RGD-0.2 por la que M-3(2), también pasará a *pendiente*. Esto es, todo el proceso definido por las RGD llevará a un desarrollo y revisión que sigue la perspectiva DCU de InterMod.

4.4.1.2 Paso 2.i. Planificar la Iteración Paralela

En este paso, los modelos se actualizan al estado *posible* debido a la perspectiva DCU, y a *enDesarrollo* por el reparto de actividades.

Cambios de estado por la perspectiva DCU

Estas reglas expresan el paso estado *pendiente* a *posible.* La perspectiva DCU integrada en InterMod implica que el desarrollo de la aplicación debe seguir un orden de precedencia en la creación y evaluación de los modelos que debe asegurarse para cada UOi del proyecto, tal y como se presenta a continuación:

$$\textbf{\textit{creadoEvaluado}}(\text{M-1(i)}) < \textbf{\textit{creadoEvaluado}}(\text{M-2(i)}) < \textbf{\textit{creadoEvaluado}}(\text{M-3(i)})$$

donde "<" quiere decir "precede "o "se crea antes que"

Esta perspectiva se recoge en las siguientes dos reglas de Gestión Directa:

RGD-1.1. El *Modelo SE-HCI* de cualquier UO siempre puede empezar a especificarse, es decir, M-1(i) pasa automáticamente de un estado *pendiente* a un estado *posible.* Formalmente:

RGD-1.1. $\forall$i [*pendiente*(M-1(i)) $\rightarrow$ *posible*(M-1(i))]

RGD-1.2. El *Modelo de Presentación* y el *Modelo de Funcionalidad* de cualquier UO, sólo pueden empezar a crearse, si sus modelos precedentes correspondientes, según la perspectiva DCU, están en estado *creadoEvaluado.* Formalmente:

RGD-1.2 $\forall$i $\forall$k $\exists$j [((2≤k≤3) $\wedge$ *pendiente*(M-k(i)) $\wedge$ (j= k-1) $\wedge$ *creadoEvaluado*(M-j(i))) $\rightarrow$ *posible*(M-k(i))] *donde i es el nº del UO y j, k son el tipo de modelo*

Por ejemplo, concretando para un determinado UO2:

si UO2 ∈ ListaUOs & *pendiente*(M-1(2)) ***entonces*** *posible*(M-1(2))-RGD-1.1

En el momento (iteración) en que se evalúa positivamente M-1(2):

si *pendiente*(M-2(2)) & *creadoEvaluado*(M-1(2)) ***entonces*** *posible*(M-2(2))-RGD-1.2

Y, de igual modo, al crearse y evaluarse M-2(2):

si *pendiente*(M-3(2)) & *creadoEvaluado*(M-2(2)) ***entonces*** *posible*(M-3(2))-RGD-1.2

Cambios de estado por reparto de actividades

Igualmente, en el paso 2.i y tras construir el plan de Actividades para cada equipo en la iteración, si una actividad está en el Plan pasará automáticamente a *enDesarrollo* (transición 2). Esto es:

RGD-2. $\forall$i$\forall$k [((1<=k<=3) $\wedge$ plan (Actividad-k(i))) $\rightarrow$ *enDesarrollo*(M-k(UOi))]

donde i es el nº del UO y k es el tipo de modelo o actividad

4.4.1.3 Paso 3.i Realizar las Actividades de la Iteración

En este paso, se producen cambios de estado a *posible*, *creadoEvaluado* y *enRevisión* durante el proceso de realización de las actividades planificadas.

Cambios de estado para el avance del proyecto

En el paso 3.i de InterMod se toman decisiones de desarrollo que llevan al gestor del proyecto a la actualización manual del estado de los modelos.

RGM-3. Si el desarrollo del modelo no se completa en la iteración, o bien su evaluación no es positiva, se actualiza su estado a *posible* (transición 3).

RGM-4. Si el modelo se evalúa positivamente, se cambia su estado a *creadoEvaluado* (transición 4).

RGM-5. Si al evaluar el modelo de un UO se considera necesario cambiar el estado a *enRevisión* de otros modelos *enDesarrollo* o *creadoEvaluado*, de éste u otros UOs (transición 5).

Cambios de estado a enRevisión por necesidades del desarrollo

Un modelo de un UO se etiqueta *enRevisión* cuando precisa ser reconsiderado de manera general. Hay que tener en cuenta que poner a revisión modelos de algunos UOs, implica la revisión de modelos de otros UOs relacionados por cuestiones de división o fusión y que surgen en iteraciones **posteriores** al UO revisado. deben ser actualizados según las siguientes reglas de Derivación Indirecta:

RDI-6a.1 Si un modelo M-k(i) se pone en estado *enRevisión* y UOi se ha dividido en {UOl, UOm, …}, todos los modelos tipo k de los UOs división también pasan a estado *enRevision.* Expresado formalmente:

RDI-6a.1. $\forall i\ \forall r\ \exists k\ [\ (\ (1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(i)) $\wedge$

divididoEn(UOi, {UOl, UOm, …})) $\rightarrow$ (*enRevisión*(M-k(r)) $\wedge\ r \in$ {l, m, …})]

donde i,r,l,m son nº de UOs y k es el tipo de modelo

RDI-6a.2. Si un modelo tipo k asociado a un UOr se pone en estado *enRevision* y UOr está fusionado en UOi, el modelo M-k(i) también pasa a estado *enRevision.* Expresado formalmente:

RDI-6a.2. $\forall i\ \exists r\ \exists k\ [\ (\ r \in \{l, m, \ldots\} \wedge (1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(r))

$\wedge$ *fusionDe* (UOi, {UOl, UOm, …})) $\rightarrow$ *enRevisión*(M-k(i))]

donde i,r,l,m son nº de UOs y k es el tipo de modelo

La aplicación de estas dos reglas se observa en una situación posible de un desarrollo parcial de WebButcher (Figura 4.14), cuando M-1(1) está *enRevisión*. Como UO1 ha sufrido procesos de división (UO4, UO5, UO6 y UO7) y fusión (UO8) posterior, también se pondrán *enRevisión* sus modelos M-1 por las reglas RDI-6a.1 y RDI-6a.2 respectivamente.

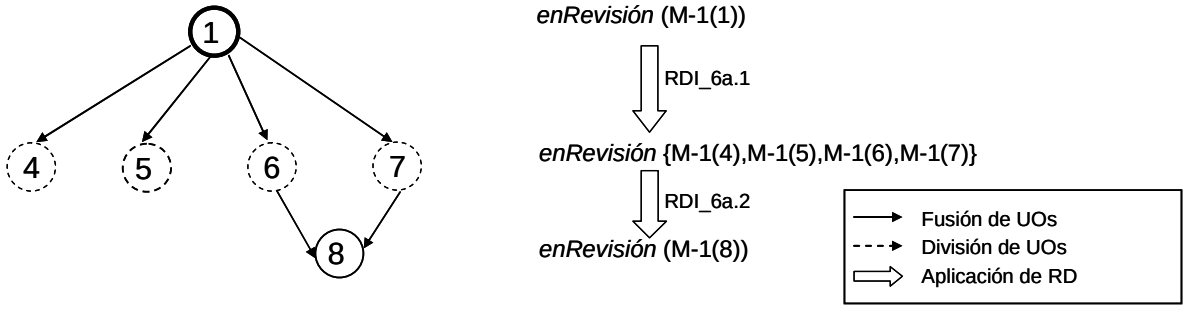


Figura 4.14 Ejemplo de aplicación de las RDI 6a.1 y 6a.2

Cambios de estado por necesidades de reutilización de modelos y UOs

Con la finalidad de dejar modelos y UOs reutilizables para este u otros proyectos, es necesario que, tras una revisión, se actualicen todos los modelos y UOs implicados, incluídos los desarrollados con anterioridad a la revisión y relacionados con el UO afectado. Por lo tanto, es conveniente que los UOs que surgen en iteraciones **anteriores** a la revisión sean actualizados en función de las siguientes reglas manuales:

RGM-6b.1. Si un modelo M-k asociado a un UOr se pone en estado *enRevision*, el modelo M-k del UOi del que es parte (división) UOr, también pasa a estado *enRevision*

RGM-6b.1. $\forall i\ \exists r\ \exists k\ [\ (\ divididoEn$(UOi, {UOl, UOm, …}) $\wedge\ r \in \{l, m, \ldots\} \wedge$ **(**$1 \leq k \leq 3$)

$\wedge$ *enRevisión*(M-k(r))) $\rightarrow$ *enRevisión*(M-k(i))]

donde i,r,l,m son nº de UOs y k es el tipo de modelo

RGM-6b.2. Si un modelo M-k asociado a un UOi se pone en estado *enRevision*, todos los modelos M-k de los UOs fusionados en UOi, {UOl, UOm, …} también pasan a estado *enRevision.*

RGM-6b.2. $\forall i\ \exists k\ \forall r\ [\ (\ \textit{fusionDe}\ (UOi, \{UOl, UOm, \ldots\}) \wedge (1 \leq k \leq 3) \wedge$

$\textit{enRevisión}(M\text{-}k(i))\ ) \rightarrow \textit{enRevisión}(M\text{-}k(r)) \wedge r \in \{l, m, \ldots\}\ ]$

donde i,r,l,m son nº de UOs y k es el tipo de modelo

RGM-6b-3. Además, tras la revisión de un modelo de un UOi, si el objetivo del proceso es conseguir la reutilización posterior de algún modelo del UOi o del UOi completo, es necesario revisar el proceso de modelado desde el M-1. Esto es, si el cambio afecta directamente a los requisitos se revisa el modelo M-1(i). Sin embargo, si se detecta la necesidad de revisión en un modelo diferente, InterMod recomienda que el modelo M-1(i) pase al estado *enRevision*, aunque esto se deja a decisión del equipo.

RGM-6b.3. $\forall i\ \exists k\ [\ (\ (2 \leq k \leq 3) \wedge \textit{enRevisión}(M\text{-}k(i))\ ) \rightarrow \textit{enRevisión}(M\text{-}1(i))\ ]$

donde i es el nº del UO y k es el tipo de modelo

Estas reglas 6b podrían ser de aplicación automática por reglas de Derivación Indirecta (RDI) en el caso de 6b.1 y 6b.2, y de Gestión Directa (RGD) para 6b.3, si el gestor lo decide al inicio del proyecto para asegurar que todos los modelos y UOs del proyecto son reutilizables. Por defecto se consideran de aplicación manual, mediante reglas de gestión manual (RGM), por el gestor del proceso en cada aplicación de la regla, para la agilidad del proyecto.

En la Figura 4.15 se observa la aplicación de estas reglas en otra posible situación en el desarrollo de WebButcher. Supongamos que el modelo M-2 para UO8 se pone *enRevisión.* Por ser fusión de UO6 y UO7, es aconsejable que los modelos M-2 de los UOs fusionados sean actualizados al estado *enRevisión* (regla RGM-6b.1). En ese momento, debido a que los UOs puestos *enRevision* son parte de UO1, es aconsejable que el modelo M-2(1) sea también puesto *enRevisión* (regla RGM-6b.2).

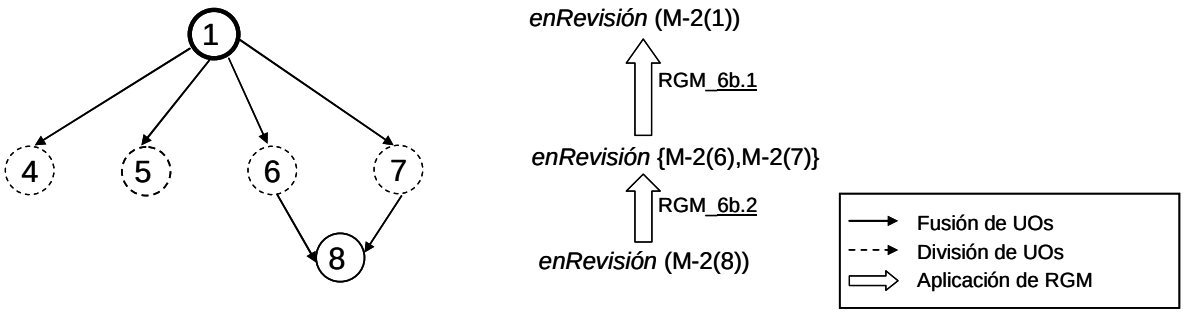


Figura 4.15 Ejemplo de aplicación de las RGM 6b.1 y 6b.2

4.4.2 Cambios de estado por Creación Indirecta

La creación indirecta de estados produce que los modelos de ciertos UOs provoquen automáticamente saltos en el transcurso habitual de estados de otros UOs. Los modelos que se actualizan son aquéllos asociados a UOs divididos, fusionados e incrementados.

4.4.2.1 Creación Indirecta por division de UOs

Esta regla provoca el cambio de estado de un modelo de *pendiente* a *creadoEvaluado*, y se aplica en el paso 1.i tras la construcción de la lista de UOs. Los modelos, creados y evaluados positivamente para un UOi, que se divide en nuevos UOs, también se dan por creados y evaluados positivamente para los nuevos UOs. La formalización de esta situación queda recogida en la regla RCI-8:

RCI-8. $\forall i\ \forall r\ \exists k\ [\ (\ \textit{divididoEn}(UOi, \{UOl, UOm, \ldots\}) \wedge (1 \leq k \leq 3) \wedge \textit{creadoEvaluado}(M\text{-}k(i))$
$\wedge\ r \in \{l, m, \ldots\} \wedge \textit{pendiente}(M\text{-}k(r))\ ) \rightarrow \textit{creadoEvaluado}(M\text{-}k(r))\ ]$

donde i,r,l,m son nº de UOs y k es el tipo de modelo

Se muestra a continuación la aplicación de esta regla en dos ejemplos. Los modelos que cambian a estado *creadoEvaluado* por derivación indirecta se muestran en oscuro. En el primer ejemplo, visualizado en la Figura 4.16, la actividad DA-1(1) crea y evalúa positivamente el modelo M-1(1). En la siguiente iteración, UO1 se divide en UO4, UO5, UO6 y UO7. Al introducirse esos nuevos UOs en ListaUOs, los modelos {M-1(4), M-1(5), M-1(6),

M-1(7)}, se incluyen en estado *pendiente* por RGD-0.1; y, por RCI-8, todos pasan a estado *creadoEvaluado*, ya que M-1(1) está en estado *creadoEvaluado*.

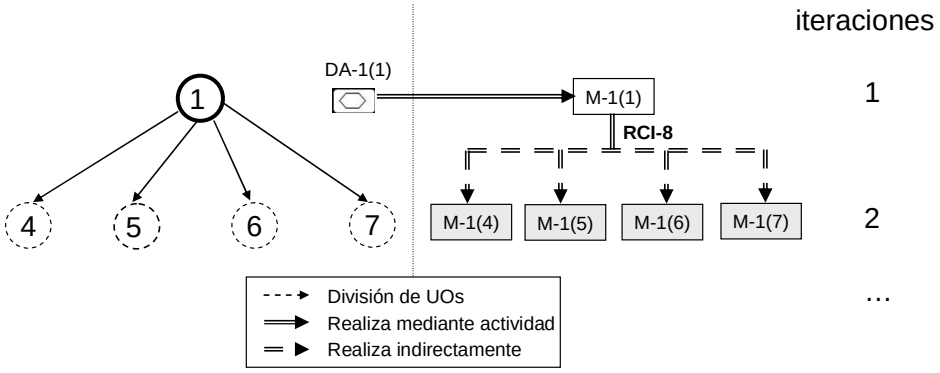


Figura 4.16 Ejemplo de Creación indirecta y automática de modelos tras una división

En el ejemplo de la Figura 4.17 se muestra el comportamiento de la regla RCI-8 cuando suceden varias divisiones de un UO a lo largo de diferentes iteraciones.

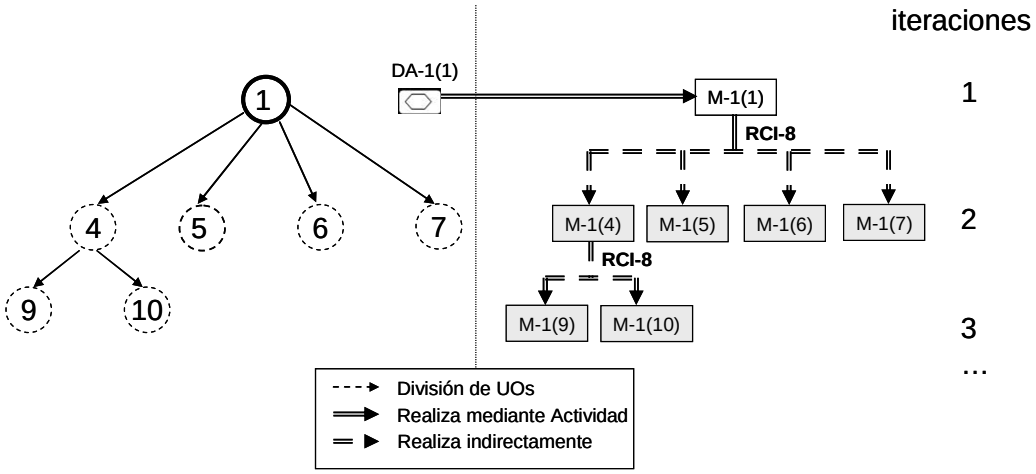


Figura 4.17 Ejemplo de Creación de modelos, indirecta y automática, tras divisiones sucesivas de un UO

En la iteración 2, UO1 se divide en {UO4, UO5, UO6, UO7}. Supongamos que posteriormente, UO4 se divide en {UO9, UO10} en la iteración 3. En la iteración 1, la actividad DA-1(1) crea y evalúa positivamente M-1(3) que, como consecuencia, adquiere un estado *creadoEvaluado.* La división de UO1 provoca la activación de RCI-8, y los modelos {M-1(4), M-1(5), M-1(6), M-1(7)} pasan automáticamente a estado *creadoEvaluado*. Pero además, la división de UO4 activa de nuevo RCI-8 y los modelos {M-1(9), M-1(10)} se consideran también en estado *creadoEvaluado.*

4.4.2.2 Creación Indirecta por fusión de UOs

Esta regla provoca el paso automático de *posible* a *creadoEvaluado* y se aplican en el paso 3.i de la iteración. Cuando el modelo M-k(i) para un UOi fusión de {UOl, UOm, …} se crea y evalúa positivamente, se asume que los M-k para sus UOs fusionados {M-k(l), M-k(m),…} están también en estado *creadoEvaluado*. Al igual que sucede con los UOs divididos, esto es debido a que M-k(i) incluye a los otros modelos. Esto se formaliza con la siguiente regla:

RCI-9.1. $\forall i\ \exists r\ \exists k\ [\ (\ fusionDe(UOi, \{UOl, UOm, \ldots\}) \wedge (1 \leq k \leq 3) \wedge$

$creadoEvaluado(M\text{-}k(i)) \wedge r \in \{l,m,\ldots\} \wedge posible(M\text{-}k(r))\ )$

$\rightarrow creadoEvaluado(M\text{-}k(r))\ ]$

donde i,r,l,m son nº de UOs y k es el tipo de modelo

La aplicación de esta regla se observa en los tres ejemplos siguientes, que son desarrollos diferentes para WebButcher. Los modelos que cambian a estado *creadoEvaluado* por creación indirecta aparecen gráficamente en oscuro.

En el primer ejemplo (Figura 4.18), UO6 y UO7 se fusionan en UO8, y la actividad DA-1(8) que se realiza en la iteración 3 crea y evalúa positivamente el modelo M-1(9).

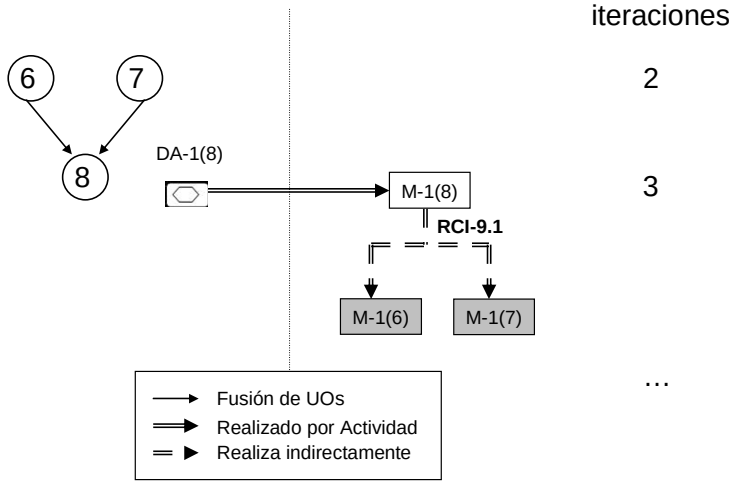


Figura 4.18 Ejemplo de Creación de modelos, indirecta y automática tras una Fusión

En ese momento según la Regla de Creación Indirecta RCI-9.1, los modelos M-1(1) y M-1(2), en estado *posible*, quedan automáticamente en un estado *creadoEvaluado*.

En el segundo ejemplo, en la Figura 4.19, se aplica varias veces la regla RCI-9.1.

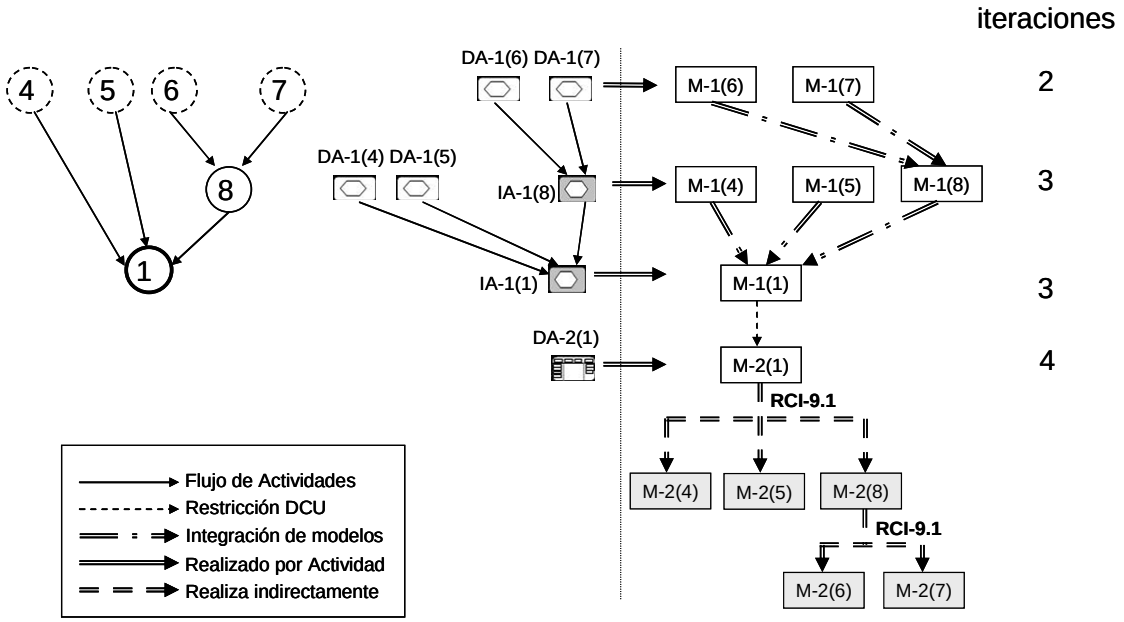


Figura 4.19 Ejemplo de Creación de modelos, indirecta y automática, tras fusiones sucesivas

UO1 es la fusión de {UO4, UO5, UO8} y, por otra parte, UO8 es la fusión de {UO6, UO7}. La actividad DA-2(1) crea y evalúa positivamente el modelo M-2(1) en la 4ª iteración, por lo que en ese momento se activa RCI-9.1 actualizando automáticamente M-2(4), M-2(5) y M-2(8) al estado *creadoEvaluado*. Nuevamente se activa RCI-9.1 y los modelos M-2(6) y M-2(7) también quedarán creados y evaluados, por estar M-2(8) en estado *creadoEvaluado*

El tercer ejemplo (Figura 4.20) muestra el caso de un UO Fusión en el que algún modelo se materializa mediante una actividad incremental. En la iteración 3, se realiza la actividad IA-1(9) dejando el modelo M-1(9) en estado *creadoEvaluado*. Posteriormente, se realiza una actividad de integración incremental IA-2(9) que produce M-2(9) a partir del modelo M-2(2). M-2(3) es *posible* pero aún no está creado. En este momento, la aplicación de la regla RCI-9.1 hace que el modelo M-2(3) quede automáticamente en estado *creadoEvaluado*.

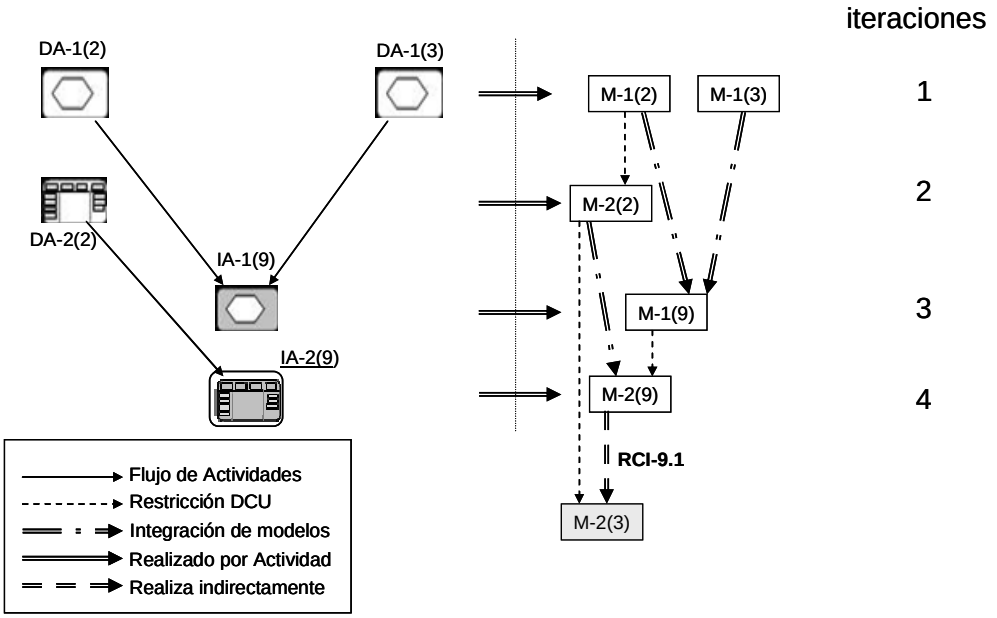


Figura 4.20 Ejemplo de Creación de modelo, indirecta y automática, tras fusión y Actividad Incremental

4.4.2.3 Creación Indirecta por UOs Incrementados

Los modelos de los UOs Incrementos siempre se obtienen por derivación indirecta porque no tienen autonomía. Cuando se crea y evalúa positivamente un modelo de un UO incrementado, los modelos correspondientes a los UOs Incremento se consideran también *creadosEvaluados*. Esto se representa formalmente con esta regla:

RCI-9.2 $\forall i \exists j \forall r \exists k$ [(*incrementadoDe*(UOi, UOj, {UOl, UOm, …}) $\wedge$

creadoEvaluado (M-k(i)) $\wedge$ posible(M-k(r)) $\wedge$ r$\in$ {j,l,m,…}) $\rightarrow$ *creadoEvaluado*(M-k(r))]

La Figura 4.21 muestra la creación de modelos que se produce cuando interviene un UO Incrementado. En este caso, UO12 es un UO Incremento de UO1 que crea el UO Incrementado UO13. M-1(1) y M-2(1) están en estado *creadoEvaluado* antes de la integración incremental. La actividad incremental IA-1(13) crea y evalúa el modelo M-1(13) y, por RCI-9.2, M-1(12) pasa a estar *creadoEvaluado.* Del mismo modo, cuando se realiza la actividad incremental IA-2(13) se crea y evalúa el modelo M-2(13) y, por RCI-9.2, M-2(12) pasa a estar *creadoEvaluado.* Finalmente, se realiza la actividad DA-3(13) que crea y evalúa el modelo M-3(13) y, por RCI-9.2, M-3(1) y M-3(12) pasan a estar *creadoEvaluado.*

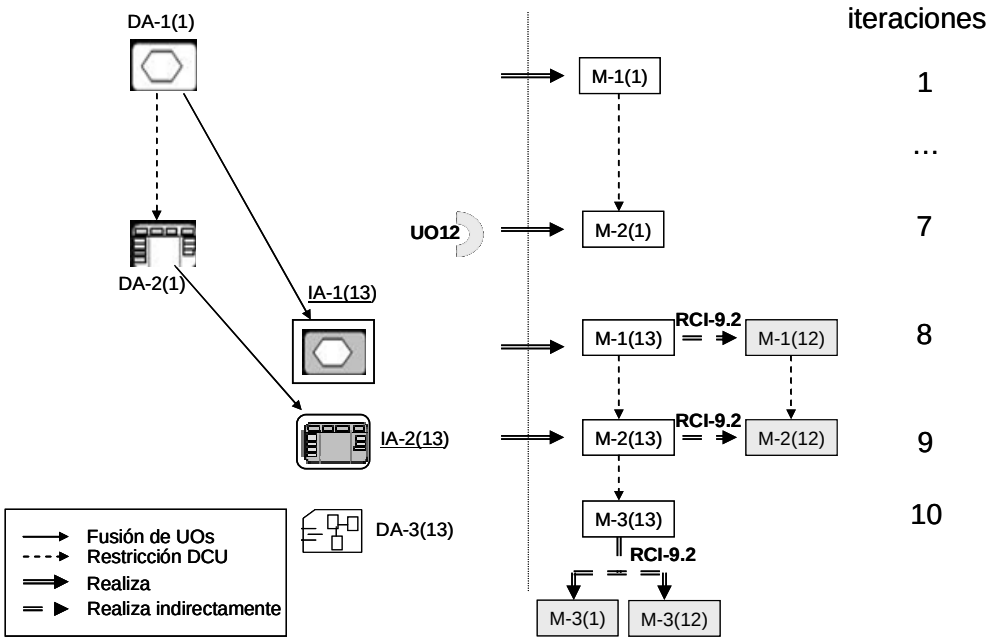


Figura 4.21 Ejemplo de Creación de modelos, indirecta y automática, a partir de un UO Incrementado

4.4.3 Evolución de un proyecto por modelos

La Hoja de Gestión, correspondiente al estado del proyecto y descrita parcialmente en la sección 3.4.5, se completa con la información de la creación indirecta, derivada de modelos por fusión y división e incremento de UOs. En el proyecto *WebButcher* esta Hoja de Gestión quedaría como se muestra en la Tabla 4.2.

Tabla 4.2 Hoja de Gestión del proyecto *WebButcher*: Estado del proyecto

	UO1	**UO2**	**UO3**	**UO4**	**UO5**	**UO6**	**UO7**	**UO8**	**UO9**	**UO10**	**UO11**	**UO12**	**UO13**
M-1	X1	X1	X1	D2	D2	D2	D2	D3	I3	X5	I6	F8	I8
M-2	I6	X2	F4	X3	X3	X2	X2	I3	I4	X6	I7	F9	I9
M-3	F10	F5	F5	F10	F10	F10	F10	F10	X5	X7	I9	F10	X10

Xi: Realizado por una actividad de Desarrollo en la iteración i
Ii: Realizado por una actividad de Integración
Ii: Realizado por una actividad de Integración incremental
Fi: Realizado indirectamente a través de un UO Fusión
Di: Realizado indirectamente a través de un UO División
Fi: Realizado indirectamente a través de un UO Incrementado

Tal y como se observa en el diagrama de actividades finalizadas (Figura 3.18), durante el proceso se realizan divisiones y fusiones de UOs (y divisiones e integraciones de modelos) que ocasionan cambios en los estados de los modelos afectados de forma indirecta. Esto se refleja en la Hoja de Gestión como Di, si en la iteración i se ha producido un cambio de estado del modelo por división; Fi, si en la iteración i el cambio es debido a una fusión de UOs y Fi, si es debido a un UO Incrementado. Por ejemplo, de las creaciones indirectas mostradas en la Figura 4.21, se observa la creación de M-1(12) en la iteración 8 y por ello aparece en la celda (M-1,UO12) con el valor F8.

4.5 Discusión y Trabajos Relacionados

En los últimos años, la industria ha avanzado hacia la estandarización de las notaciones de modelado visual. UML es el producto de este esfuerzo, y es un estándar del Object Management Group ("Object Management Group - UML," n.d.) (OMG) desde 1997. La mayoría de las técnicas de modelado de software y enfoques dirigidos por modelos utilizan UML. UML ofrece muchas posibilidades a los desarrolladores para especificar sistemas de software, ya que un modelo UML puede describir la estructura o el comportamiento de un sistema desde diferentes puntos de vista y abstracciones. Esto es deseable porque se gestionan mejor las complejidades de la descripción de un sistema utilizando varios modelos, donde cada uno de ellos recoge un aspecto diferente de la solución.

El modelo propuesto por UML para describir las interacciones usuario-sistema, vinculadas con un objetivo funcional del usuario, es el de los casos de uso. Los casos de uso de un sistema contienen los requisitos funcionales deseados o existentes, los actores (usuarios del sistema) y las relaciones que unen a actores y funcionalidades. Sirven, por lo tanto, de soporte para las etapas de modelado, desarrollo y validación (Debrauwer and Heyde, 2005). Sin embargo, el modelo de caso de uso convencional falla al no proporcionar suficiente información del usuario ni sobre la usabilidad para ser capaz de diseñar de forma fiable sistemas interactivos usables (Harmelen, 2001). Aunque el equipo de Obrenovic ha investigado el uso de modelos conceptuales expresados en UML para describir interfaces de usuario multimodales (Obrenovic et al., 2004), Paternò argumenta que UML tiene

limitaciones en la modelización de interfaces de usuario, en el sentido de lo expresado por Harmelen (Paterno et al., 2008).

Otros autores extienden y refinan los casos de uso para el diseño de sistemas interactivos. Por ejemplo, Wisdom (Nunes and Cunha, 2001) y los casos de uso esenciales (Constantine and Lockwood, 2002). Larry Constantine y Lucy Lockwood proponen el método de diseño centrado en el Uso, cuyos casos de uso esenciales son un refinamiento del modelado con casos de uso. Los casos de uso esenciales expresan la interacción del usuario y el comportamiento del sistema de forma abstracta, sin detalles de implementación e independientes de la tecnología de implementación. Esto permite a los diseñadores concentrarse en la funcionalidad del sistema propuesto sin distraerse con detalles de implementación. Esta enfoque aborda una de las necesidades del diseño de sistema interactivos: diseñar los contenidos y capacidades de los sistemas interactivos, sin detallar soluciones de diseño prematuras que luego pueden no ser las adecuadas.

Los modelos utilizados en IS, mayoritariamente UML, expresan conceptos en el dominio de la aplicación, y se abstraen de las formas de computación, almacenamiento y proceso, inherentes a los lenguajes de programación. Los modelos de HCI son también abstracciones de la solución, pero se centran en modelar la interacción usuario-sistema para facilitar su evaluación con el usuario, especialmente en lo concerniente a la usabilidad. Estos modelos son la base del modelo de requisitos SE-HCI, propuesto por InterMod.

Al igual que en nuestra propuesta para el modelo SE-HCI, Propp y sus colegas (Propp et al., 2008) comienzan el proceso de desarrollo de aplicaciones interactivas con los *Modelos de Tareas de Usuario*. A continuación, definen la estructura de navegación y, finalmente, crean una Interfaz de Usuario Abstracta (Abstract User Interface - AUI) independiente del dispositivo y una o más Interfaces de Usuario Concretas (Concrete User Interface - CUI). Las diferentes iteraciones siguen los pasos de la cadena: Modelo de Tareas de Usuario-Modelo de diálogo-AUI-CUI. Durante el proceso de desarrollo llevan a cabo varias evaluaciones de usabilidad. A diferencia nuestra, no hay división del trabajo y el proyecto se lleva a cabo en un bloque.

En este mismo sentido, el grupo de Paternò estudia cómo los modelos de HCI soportan el desarrollo de aplicaciones interactivas. Han desarrollado varias herramientas, como TERESA

(Berti et al., 2004a) o MARIA (Paternò et al., 2009), que permiten la descripción de interfaces de usuario a nivel abstracto y concreto. Comienzan con descripciones lógicas de las tareas en CTT, que muestran el sistema interactivo desde la perspectiva del usuario en términos de actividades lógicas y los objetos que manipulan. A continuación, esto se traduce a una interfaz abstracta de usuario que proporciona una descripción de la interfaz de usuario en XML, independiente de la modalidad de la plataforma. Después, se define la interfaz de usuario concreta, dependiente de la modalidad de la plataforma. Finalmente, se obtiene la implementación final en un lenguaje para el desarrollo de la interfaz de usuario. El modelo SE-HCI es un modelo HCI que describe una interfaz de usuario a nivel abstracto aunque también incluye algunos elementos genéricos de presentación que facilitan su evaluación temprana, y la semántica de la aplicación (cambio de la navegación si se producen determinados errores) que dirige la implementación futura. La diferencia fundamental con el trabajo de Paternò reside en el desarrollo organizado por Objetivos de Usuario (UOs), lo que permite diferenciar las tareas de usuario de los objetivos de programación, y por lo tanto, especificar métodos de evaluación diferentes.

4.6 Resumen

En este capítulo se describen los modelos en InterMod y sus interdependencias, incidiendo especialmente en el *Modelo SE-HCI* o modelo de requisitos en InterMod. Este modelo engloba a un conjunto de modelos cuyo objetivo es validar los requisitos con los implicados del proyecto en etapas tempranas de su desarrollo y adelantar en esa validación aspectos parciales de presentación y el comportamiento en la interfaz de la aplicación interactiva. En este capítulo se indican las técnicas propuestas para la evaluación de cada modelo y se muestra la gestión de modelos en InterMod.

InterMod propone tres tipos de modelos a desarrollar para cada UO del proyecto. Los modelos son: M-1.*Modelo de Requisitos*, M-2.*Modelo de Presentación* y M-3.*Modelo de Funcionalidad*. Para facilitar la gestión iterativa del proyecto se han establecido diferentes estados para los modelos de un UOi, cuyo transcurso habitual viene determinado por la secuencia: {*pendiente*, *posible*, *enDesarrollo*, *creadoEvaluado*}. Además, eventualmente, un modelo puede pasar a estado *enRevisión*.

Se han diseñado un conjunto de reglas de gestión de modelos que permiten actualizar sus estados tras cada paso del proceso. Los estados de los modelos pueden modificarse de forma automática a través de reglas de Gestión Directa (RGD) que se aplican: al crear los UOs e incluirlos en la ListaUOs, al aplicar el requisito DCU de InterMod para el desarrollo de cada UO y al establecer el plan de actividades para cada equipo. A lo largo del desarrollo del proyecto también se toman decisiones de gestión en las que se actualizan manualmente los estados de los modelos mediante reglas de Gestión Manual (RGM) que aplica el gestor del proyecto tras los procesos de evaluación de los modelos, y tras procesos de revisión que afectan a otros modelos de éste u otros UOs. Por otra parte, la derivación indirecta de estados produce automáticamente saltos en el transcurso normal de los cambios de estado (RDI), motivados por procesos de revisión y por procesos de creación indirecta (RCI). La Tabla 4.3 agrupa el conjunto de reglas de gestión que se aplican en los pasos de la metodología y que actualizan el estado de los modelos. La actualización de los estados de los modelos mediante estas reglas de gestión, junto con las reglas de planificación (mostradas en el capítulo 3), aseguran un progreso del proyecto con criterios de calidad, formalidad y adaptadas al usuario final.

Tabla 4.3 Resumen de las Reglas de Gestión de Modelos

PASO DE APLICACIÓN	**REGLAS de GESTIÓN DE MODELOS**
Paso 1	$\forall i$ (UOi$\in$ ListaUOs $\rightarrow$ *pendiente*(M-1(i))) **(RGD-0.1)**
	$\forall i\ \forall k$ [((1$\leq$k$\leq$2) $\wedge$ *pendiente*(M-k(i))) $\rightarrow$ (j=k+1 $\wedge$ *pendiente*(M-j(i))] **(RGD-0.2)**
	$\forall i\ \forall k$ [((1$\leq$k$\leq$3) $\wedge$ *enRevisión*(M-k(i))) $\rightarrow$ *pendiente*(M-k(i))] **(RGD-7)**
	$\forall i\ \forall r\ \exists k$ [(*divididoEn*(UOi, {UOl, UOm, …}) $\wedge$ (1$\leq$k$\leq$3) $\wedge$ *creadoEvaluado*(M-k(i)) $\wedge$ r$\in$ {l, m, …}$\wedge$ *pendiente*(M-k(r))) $\rightarrow$ *creadoEvaluado*(M-k(r))] **(RCI-8)**
Paso 2	$\forall i$ [*pendiente*(M-1(i) $\rightarrow$ *posible*(M-1(i))] **(RGD-1.1)**

PASO DE APLICACIÓN	REGLAS de GESTIÓN DE MODELOS
	$\forall i\ \forall k\ \exists j\ [(\ (2 \leq k \leq 3) \wedge$ *pendiente*(M-k(i)) $\wedge$ (j= k-1) $\wedge$ *creadoEvaluado*(M-j(i))) $\rightarrow$ *posible*(M-k(i))] **(RGD-1.2)**
	$\forall i \forall k$ [((1<=k<=3) $\wedge$ plan (Actividad-k(i))) $\rightarrow$ *enDesarrollo*(M-k(UOi))] **(RGD-2)**
Paso 3	Si el desarrollo del modelo no se completa en la iteración, o bien su evaluación no es positiva, se actualiza su estado a *posible* (**RGM-3)**
	Si el modelo se evalúa positivamente, se cambia su estado a *creadoEvaluado.* (**RGM-4)**
	Si al evaluar el modelo de un UO se considera necesario cambiar el estado a *enRevisión* de otros modelos *enDesarrollo* o *creadoEvaluado*, de éste u otros UOs. **(RGM-5)**
	$\forall i\ \forall r\ \exists k$ [($(1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(i)) $\wedge$ *divididoEn*(UOi, {UOl, UOm, …})) $\rightarrow$ (*enRevisión*(M-k(r)) $\wedge$ r $\in$ {l, m, …}) **(RGD-6a.1)**
	$\forall i\ \exists r\ \exists k$ [(r$\in$ {l, m, …} $\wedge$ $(1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(r)) $\wedge$ *fusionDe* (UOi, {UOl, UOm, …})) $\rightarrow$ *enRevisión*(M-k(i))] **(RGD-6a.2)**
	$\forall i\ \exists r\ \exists k$ [(*divididoEn*(UOi, {UOl, UOm, …}) $\wedge$ r$\in$ {l, m, …} $\wedge$ $(1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(r))) $\rightarrow$ *enRevisión*(M-k(i))] **(RGM-6b.1)**
	$\forall i\ \exists k\ \forall r$ [(*fusionDe(* UOi ,{UOl, UOm, …}) $\wedge$ $(1 \leq k \leq 3) \wedge$ *enRevisión*(M-k(i))) $\rightarrow$ *enRevisión*(M-k(r)) $\wedge$ r$\in$ {l, m,…}] **(RGM-6b.2)**
	$\forall i \exists k$ [($(2 \leq k \leq 3) \wedge$ *enRevisión*(M-k(i))) $\rightarrow$ *enRevisión* (M-1(i))] **(RGM-6b.3)**

PASO DE APLICACIÓN	REGLAS de GESTIÓN DE MODELOS
	$\forall i\ \exists r\ \exists k\ [\ (\ \textit{fusionDe}(UOi, \{UOl, UOm, \ldots\}) \wedge (1 \leq k \leq 3) \wedge$ $\textit{creadoEvaluado}(M\text{-}k(i)) \wedge r \in \{l, m, \ldots\} \wedge \text{posible}(M\text{-}k(r))\ ) \rightarrow$ $\textit{creadoEvaluado}(M\text{-}k(r))\ ]$ **(RCI-9.1)**
	$\forall i \exists j \forall r \exists k\ [\ (\ \textit{incrementadoDe}(UOi, UOj, \{UOl, UOm, \ldots\}) \wedge$ $\textit{creadoEvaluado}\ (M\text{-}k(i)) \wedge \text{posible}(M\text{-}k(r)) \wedge r \in \{j,l,m,\ldots\}\ ) \rightarrow$ $\textit{creadoEvaluado}(M\text{-}k(r))\ ]$ **(RCI-9.2)**

i,l,m,r es el nº del UO - j,k es el tipo de modelo.

En el siguiente capítulo se puede ver la aplicación de InterMod en el desarrollo paso a paso de una aplicación para móvil, haciendo especial hincapié en los puntos de integración con la Ingeniería de Usabilidad.

CAPÍTULO 5. Integración de la Ingeniería de la Usabilidad en InterMod. Desarrollo paso a paso de una aplicación para móvil

5.1 Introducción

InterMod, como el resto de métodos ágiles ("Manifesto for Agile Software Development", 2001), surge de la necesidad de encontrar formas mejores de desarrollar software. El desarrollo ágil de software es una alternativa a los enfoques tradicionales de desarrollo que surge después de numerosos fracasos en los proyectos de software. Un fracaso no es únicamente un retraso o una cancelación en la entrega de un proyecto, sino que puede significar que no se cumplen las expectativas del usuario final y, por lo tanto, que nunca será usado. Desde este punto de vista, la integración de la Ingeniería de Software con la Ingeniería de la Usabilidad (IngUS) es la solución propuesta en InterMod (Losada et al., 2012) (Losada et al., 2013b).

Los procesos ágiles se adaptan rápidamente a los cambios de un proyecto y realizan una entrega incremental de software en periodos cortos. Por lo tanto, se analizan los requisitos

continuamente y a lo largo del proyecto, en lugar de recogerlos totalmente antes de proceder a la implementación. Esta característica ágil, propia también de InterMod, es el mayor punto diferenciador con los métodos tradicionales utilizados en IngUS y, por tanto, es el reto principal en la integración propuesta.

Junto con lo anterior, en la propuesta de integración se ha de tener también en cuenta que los métodos ágiles enfatizan la mejora continua del producto y el incremento de funcionalidad a lo largo del proyecto. Sin embargo, el compromiso de entrega continua de piezas de software puede provocar un descuido en el desarrollo de la Interfaz de la aplicación. Para solucionar esto, InterMod propone seguir el enfoque de la Arquitectura Dirigida por Modelos, y desarrollar software interactivo basado en modelos generados y evaluados durante el progreso de un proyecto según los Objetivos de Usuario, tal y como se muestra en los capítulos 3 y 4.

Este capítulo se centra en cómo se integra la IngUS en la metodología ágil InterMod. Para explicar y validar el proceso de integración, se utiliza el siguiente caso a estudio: el desarrollo paso a paso de una aplicación para móvil, llamada FindMyPlace. Este software ayuda a los usuarios a encontrar espacios físicos y personas dentro de la Facultad de Informática de la UPV/EHU; como por ejemplo clases, laboratorios, despachos, seminarios, etc. Además, muestra información del personal: nombre, número de despacho, teléfono, e-mail y su ubicación habitual en el centro. La aplicación utiliza principalmente planos del edificio. Además está pensada para utilizar GPS y tecnología de triangulación wifi para alcanzar su objetivo. A continuación, se muestra la evolución paso a paso de *FindMyPlace* y se ofrecen algunas vistas del progreso del proyecto, indicando qué técnicas de evaluación de IngUs se aplican en cada momento. Después, se explica con detalle cómo se realizaron las evaluaciones de usabilidad durante el proceso de desarrollo ágil.

5.2 Puntos de Integración de la Ingeniería de Usabilidad en InterMod

Como todas las metodologías ágiles, InterMod propone organizar un proyecto como una serie de iteraciones, y distribuir el trabajo en cada iteración de acuerdo con diferentes actividades

asociadas a los UOs. Cada UO es una unidad lógica a desarrollar como parte del resultado final y debe ser validada a través de sus modelos. Todos estos procesos, junto con el uso de un modelo de requisitos específico, el *modelo SE-HCI*, permite que los principios del DCU y el desarrollo ágil se puedan integrar de forma óptima.

Los diferentes pasos de la metodología hacen posible fragmentar coherentemente el proyecto en UOs, y los UOs en actividades, facilitando aplicar las técnicas de evaluación de la usabilidad desde etapas tempranas del proceso, como propone la IngUs (Mayhew, 1999). InterMod propone comenzar a utilizarlas en el paso 0-"*Analizar el proyecto en General*" para capturar las necesidades del usuario y las decisiones de diseño generales de la aplicación.

Las evaluaciones de Usabilidad son particularmente relevantes para los modelos creados de los UOs Directos, ya que éstos reflejan directamente las necesidades de los usuarios y por tanto, un grupo de usuarios finales deben, necesariamente, estar involucrados en su evaluación. Sin embargo, con la finalidad de acelerar el proyecto, los modelos de los UOs Indirectos pueden ser evaluados únicamente por expertos en usabilidad. Los métodos de test, indagación e inspección pueden combinarse para obtener un mejor resultado en el análisis de la usabilidad de los modelos creados en el Paso 3.i – "*Realizar las Actividades de la Iteración*". Igualmente, se proponen otros métodos de evaluación del DCU que se basan en medidas de satisfacción del usuario, eficiencia y eficacia, en la realización de las tareas.

La Figura 5.1 resume el enfoque InterMod y destaca en los recuadros de línea discontinua gruesa los puntos en los que se integran las técnicas de Ingeniería de Usabilidad.

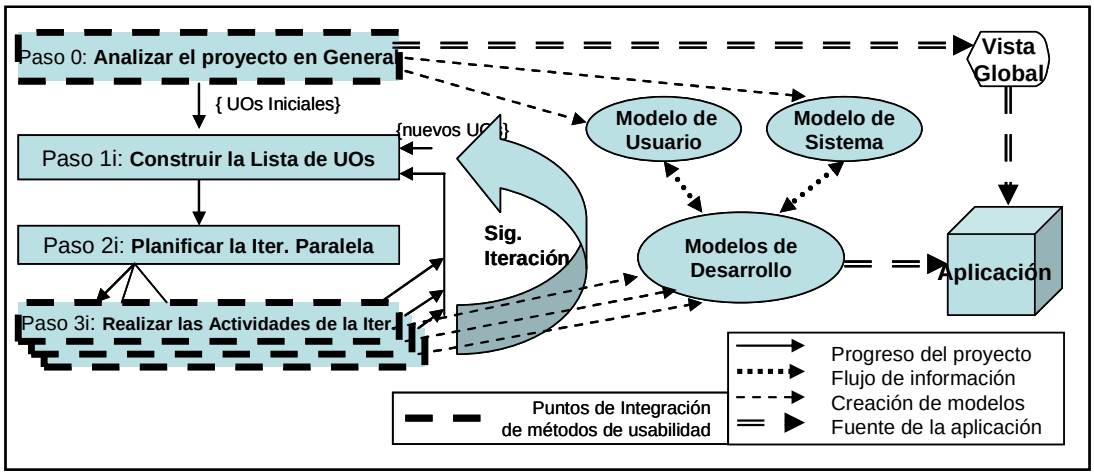


Figura 5.1 Puntos de integración de la Ingeniería de Usabilidad en InterMod

5.3 InterMod paso a paso en FindMyPlace

Desde las primeras etapas del desarrollo de un proyecto en InterMod, todas las actividades de desarrollo e integración incluyen evaluaciones adecuadas con respecto al tipo de modelo creado (ver apartado 4.2.3). Estas evaluaciones se llevan a cabo por un grupo multidisciplinar en el que los usuarios finales están presentes. De hecho, ellos están presentes desde el comienzo del proyecto (paso 0).

En esta sección se describe, paso a paso, el proceso de desarrollo de la aplicación móvil FindMyPlace siguiendo la metodología propuesta. Además se indican qué técnicas de evaluación de usabilidad se aplican: al comienzo (Paso 0) para captar las necesidades del usuario y las decisiones globales del diseño de la aplicación, y a lo largo de las iteraciones para evaluar los modelos creados (Paso 3.i).

5.3.1 Paso0-Analizar el proyecto en general

Se realizó una primera reunión de trabajo para establecer el alcance de la aplicación FindMyPlace y definir el conjunto inicial de posibles necesidades de los usuarios. Los siguientes aspectos generales fueron determinados en esta reunión:

- *Tecnología*: Se decidió desarrollar la aplicación para la plataforma móvil Android.
- *Los desarrolladores del proyecto*: Cuatro personas participarían en el desarrollo del proyecto. Estaban organizados en dos equipos: el primer equipo incluía un desarrollador de software y un diseñador, y el segundo equipo incorporaba dos evaluadores, expertos en usabilidad, para realizar las actividades de evaluación.
- *Modelo de Sistema:* Se asumieron las normas de estilo Android para el diseño de los elementos de la interfaz de usuario. No se hicieron otras restricciones iniciales acerca de la interfaz de usuario.
- *Modelo de usuario*: Aunque la aplicación prevista era potencialmente útil para una amplia gama de usuarios, se consideró a los estudiantes de primer año de la universidad como el grupo con mayores necesidades. Todos los posibles usuarios finales de la aplicación eran hispanohablantes, aunque el euskera es un idioma oficial que también debía ser

considerado. Además, se tuvieron en cuenta otras características de los usuarios como el tipo de teléfono móvil que usan (teléfono Smartphone o básico), la plataforma de telefonía móvil (por ejemplo, Android), conexión a Internet y el uso de mapas en sus teléfonos móviles.

- *Visión global*: La visión global de la aplicación se basa en tres opciones de búsqueda: "Personas", "planos del edificio" y "Guía GPS".

Los miembros del equipo acordaron preparar una sesión con usuarios finales para capturar sus deseos y los Objetivos de Usuario. En este punto, los métodos de indagación son técnicas adecuadas: cuestionarios, entrevistas, encuestas, observación de campo, etc (ver apartado 4.2.3.1).

Los deseos no funcionales identificados fueron: *la ubicación debe ser lo más exacta posible*, *la aplicación debe ser fácil de usar y consumir poca energía*, y *los menús deben ser simples y atractivos*. Éstos fueron tenidos en cuenta para ser integrados con los deseos funcionales correspondientes. Los deseos funcionales identificados se recogieron en una lista con cinco UOs Directos iniciales. Todos los UOs tuvieron la misma prioridad al comienzo del proceso de desarrollo y fueron los siguientes:

- UO1-*Guíame a un lugar determinado*: Quiero que la aplicación me guíe hacia un lugar determinado dentro del edificio.
- UO2-*Muéstrame la distribución de los espacios en el plano del edificio con diferentes niveles de detalle*: Quiero ver todas las plantas del edificio y su diseño interior con posibilidad de zoom.

Tres nuevos UOs se derivaron de la necesidad de los estudiantes para localizar los despachos de los profesores, laboratorios, aulas y otras ubicaciones no consideradas anteriormente, por ejemplo biblioteca, servicio de reprografía, cafetería, etc

- UO3-*Localízame la oficina de un profesor en el plano del edificio:* Quiero ver convenientemente señalado sobre un plano, el despacho del profesor buscado a través de una selección, caja de texto, etc.

- UO4-*Localízame un laboratorio en el plano del edificio:* Quiero ver convenientemente señalado sobre un plano, un laboratorio buscado a través de una selección, caja de texto, etc.
- UO5-*Localízame un aula o un lugar especial en el plano del edificio:* Quiero ver convenientemente señalado sobre un plano, un aula o un lugar especial como la cafetería, el servicio de reprografía, la secretaría, etc, buscado a través de una selección, caja de texto, etc.

5.3.2 Iteración 1

Después de terminar el paso 0, y en la misma sesión, comenzó la iteración 1 con los pasos que se presentan a continuación.

5.3.2.1 Paso1.1- Construir la Lista de UOs

Los desarrolladores del proyecto incluyeron los cinco UOs iniciales en la Lista de UOs:

Lista de UOs = {UO1, UO2, UO3, UO4, UO5}

5.3.2.2 Paso2.1- Planificar la Iteración Paralela

La necesidad de trabajar con GPS y los planos del edificio enfocó el trabajo en el desarrollo de UO1 (GPS) y UO2 (planos del edificio). Para esta iteración, se planificó la actividad de desarrollo "DA-1: *Análisis y diseño de navegación*" por ser la única posible, según la regla de planificación RPA-1:

RPA-1: [posible(M-1(UO1,UO2) →ListaPosiblesA (DA-1(1,2))]

Como sólo se iban a crear los modelos M-1(1) y M-1(2) (por DA-1(1) y DA-1(2) respectivamente), se decidió que se evaluaran sólo por el segundo equipo :

Plan = {Primer equipo -[DA-1 (1), DA-1 (2)], segundo equipo -[Evaluación M-1 (1, 2)]}

5.3.2.3 Paso3.1- Realizar las Actividades de la Iteración

La actividad DA-1, desarrollada para UO1 y UO2, crea el M-1.*Modelo de Requisitos*. Los modelos para ambos UOs fueron diseñados con la herramienta WebDiagram. Se muestra en la Figura 5.2 el modelo M-1 creado para UO1.

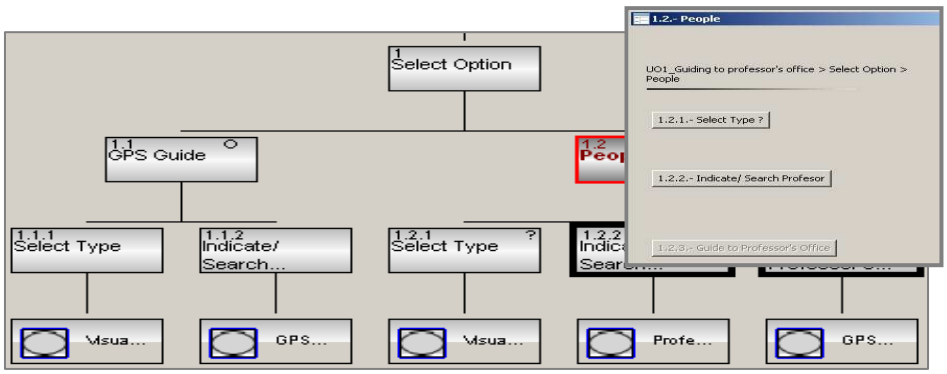


Figura 5.2 Modelo SE-HCI para UO1 construído con la herramienta Diagram

Esta herramienta permite crear y visualizar gráficamente el modelo SE-HCl, y genera automáticamente prototipos listos para ser evaluados.

Estos prototipos simulan algunas de las características del sistema tales como la navegación, el comportamiento y la presentación. La Figura 5.2 muestra las dos maneras posibles de alcanzar el deseo representado en UO1, tal y como se decidió previamente: "Guía GPS" (rama izquierda) y "Personas" (rama derecha). Por ejemplo, la segunda identifica una vía directa: seleccionar opcionalmente el tipo de local- despacho del profesor, laboratorio, aula, etc - y después, indicar el código del local. UO2 sólo tiene un camino de acceso desde la opción "Plano del Edificio".

Se evaluaron con expertos los modelos M-1. *Modelos del Requisitos,* creados para UO1 y UO2, basándose en los prototipos generados por WebDiagram (ver la ventana a la derecha en la Figura 5.2). Estos modelos fueron evaluados mediante recorridos cognitivos combinados con técnicas de “pensar en voz alta” (ver apartado 4.2.3.2). A pesar de ser UOs Directos, se decidió realizar una evaluación previa con expertos para detectar los errores de navegación mayores, antes de realizar la prueba con usuarios finales.

5.3.3 Iteración 2

Esta iteración se inició después de terminar la evaluación de los modelos M-1 (1) y M-1 (2).

5.3.3.1 Paso1.2- Construir la Lista de UOs

Además de los objetivos de la iteración 1, en esta iteración se decidió dividir UO2 en dos nuevos UOs: UO6 y UO7, debido a que la capacidad de zoom asociada con el diseño de la navegación en UO2 requirió un esfuerzo importante de desarrollo ya que no se encontraron bibliotecas específicas. Por lo tanto, los nuevos UOs fueron:

- UO6- *Muéstrame la distribución de los espacios en el plano del edificio*: Quiero ver en un plano del edificio todo el conjunto de los espacios, teniendo en cuenta todas las plantas diferentes.
- UO7- *Házme zoom del plano del edificio*: Quiero ver los planos del edificio con más detalle y moverse en ellos.

Por lo tanto, la Lista de UOs para esta iteración es:

Lista de UOs (iteración 2) = Lista de UOs (iteración1) + {UO6, UO7}

5.3.3.2 Paso 2.2- Planificar la Iteración Paralela

En la reunión de trabajo se decidió realizar DA-1 sobre {UO3, UO4, UO5} (en ListaPosiblesA por RPA-1). Primero, los modelos M-1 (3, 4, 5) se evaluaron por el segundo equipo con los prototipos de WebDiagram. Después, M-1 (1), creado y evaluado en la iteración 1, se volvió a evaluar conjuntamente con M-1 (3, 4, 5) en la iteración 2 con usuarios finales. Se escogieron prototipos de papel (Snyder, 2003) debido a problemas de fidelidad, ya que su apariencia se asemeja a la de un teléfono móvil de forma más precisa que los prototipos creados por la herramienta WebDiagram. Por otra parte, también se decidió llevar a cabo las actividades de desarrollo "DA-2: Construcción de la interfaz" y "DA-3: Codificación de la Lógica de negocio" para UO6, ya que sus modelos estaban en estado posible por la aplicación sucesiva de las reglas de precedencia DCU (ver apartado 4.4.1.2).

Las actividades se distribuyeron entre los equipos de la siguiente manera:

Plan = {Primer equipo [DA-2 (6) → DA-3 (6)],
Segundo equipo [DA-1 (3,4,5), Evaluación {M-1 (1*, 3, 4, 5), M-2(6),
M-3(6)}]; *donde *: Evaluación repetida*}

5.3.3.3 Paso 3.2- Realizar Actividades de la Iteración

El primer equipo realizó DA-2 (6) y, a continuación DA-3 (6), lo que implicó el desarrollo de un prototipo de trabajo funcional de UO6. La evaluación de estos modelos se realizó por expertos, utilizando técnicas de inspección y pruebas de robustez de la funcionalidad sobre teléfonos móvil Samsung Galaxy S: los planos del edificio se observaron correctamente, y los movimientos de los dedos sobre la pantalla hicieron posible seleccionar eficazmente el piso del edificio en que se mostraban. No se hizo ninguna evaluación con usuarios finales porque UO6 no era un objetivo directo de los usuarios, ya que era parte de UO2.

El primer equipo diseñó también los modelos M-1 de UO3, UO4 y UO5. Los diagramas de navegación obtenidos fueron evaluados inicialmente por el segundo equipo mediante el uso de la técnica de reflexión en voz alta o thinking aloud, en las interfaces de usuario generadas por la herramienta WebDiagram. A continuación, se realizó una sesión de usuario final con prototipos de papel, para evaluar la exactitud de los modelos propuestos (véase la subsección 5.4.2).

5.3.4 Visión completa del proyecto

El desarrollo, iteración a iteración, de *FindMyPlace* se resume en las siguientes figuras y tablas. Incluye los UOs creados (paso 1.i), el plan de la iteración con las DAs e IAs (paso 2.1), y el progreso de los modelos (paso 3.i).

La Figura 5.3 muestra el progreso del proyecto, teniendo en cuenta la creación de los diferentes tipos de UOs en los pasos 1.i de todas las iteraciones.

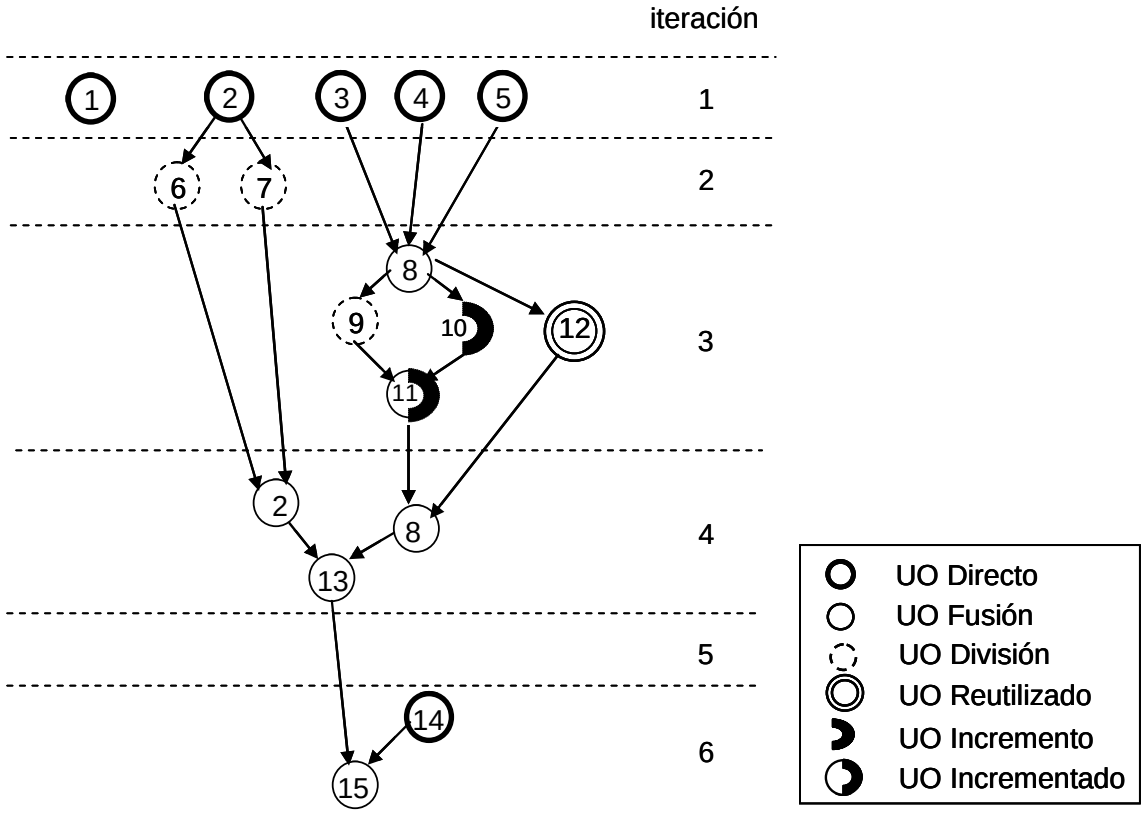


Figura 5.3 Diagrama de UOs creados (Paso 1.i) mostrando el progreso de FindMyPlace

Se observa en el diagrama que en la iteración 3 se decidió desarrollar conjuntamente UO3, UO4 y UO5, en UO8 porque el implementador consideró que su base de implementación era similar. Dada la complejidad encontrada en el marcado de planos, se decidió dividir UO8 en UO9 (visualización), UO10 (marcado) y UO12 (zoom). UO12 es un UO reutilizado, concretamente es el UO7 de este proyecto. En la 5ª iteración, surge UO14, UO Directo, que se fusionará con el resultado final. UO11, UO13 y UO15 surgen de la necesidad de fusionar objetivos hacia una aplicación única final.

Los nuevos UOs son:

- UO8-*Muéstrame un lugar X de nombre Y en el plano del edificio, con suficente nivel de detalle*: Quiero que la aplicación sea capaz de visualizar un lugar X (despacho, laboratorio, aula, otros) de nombre Y (Lab 0.6, Despacho 253, etc) en el plano del edificio, escribiendo su código (253) o nombre (Juan Pérez) (a través de una casilla de texto, búsqueda, etc), y con posibilidad de zoom.

- UO9-*Visualízame el plano de la planta donde se encuentra el lugar X de nombre Y*: Quiero ver el plano de la planta dónde se encuentra X-Y.
- UO10-*Márcame en el plano del edificio la ubicación X con el nombre Y*: Quiero ver convenientemente marcada en el plano de la planta la ubicación X-Y.
- UO11- *Indícame en el plano de la planta el lugar X de nombre Y*: Quiero ver el plano de la planta, y en él, convenientemente marcado, la ubicación X-Y.
- UO12- (UO Reutilizado = UO7)- *Házme zoom del plano de la planta*: Quiero ver el plano de la planta con más detalle y moverme en ella.
- UO13- *Muéstrame la distribución de espacios y localízame un lugar en el plano del edificio con diferentes niveles de detalle*: Quiero que muestre la distribución de los espacios de todos los pisos del edificio y que marque un lugar concreto en el plano del edificio, con posibilidades de zoom.
- UO14- *Muéstrame el listado de profesores y su información*: Quiero acceder a al terléfono, e-mail, localización y localización gráfica en el plano del personal del centro.
- UO15- *Muéstrame e indícame la información relevante de espacios y del personal del centro*: Quiero ver los planos, marcado un lugar buscado (laboratorios, aulas, etc), con suficiente nivel de detalle, y la información del personal del centro.

La Lista de UOs se actualizó como sigue:

Lista UOs (iteración 3) = Lista UOs (iteración 2) + {UO8, UO9, UO10,UO11,UO12}

Lista UOs (iteración 4) = Lista UOs (iteración 3) + {UO13}

Lista UOs (iteración 5) = Lista UOs (iteración 4)

Lista UOs (iteración 6) = Lista UOs (iteración 5) + {UO14, UO15}

La Figura 5.4 ilustra el progreso del proyecto, mostrando las actividades que se planificaron en los pasos 2.i de cada iteración.

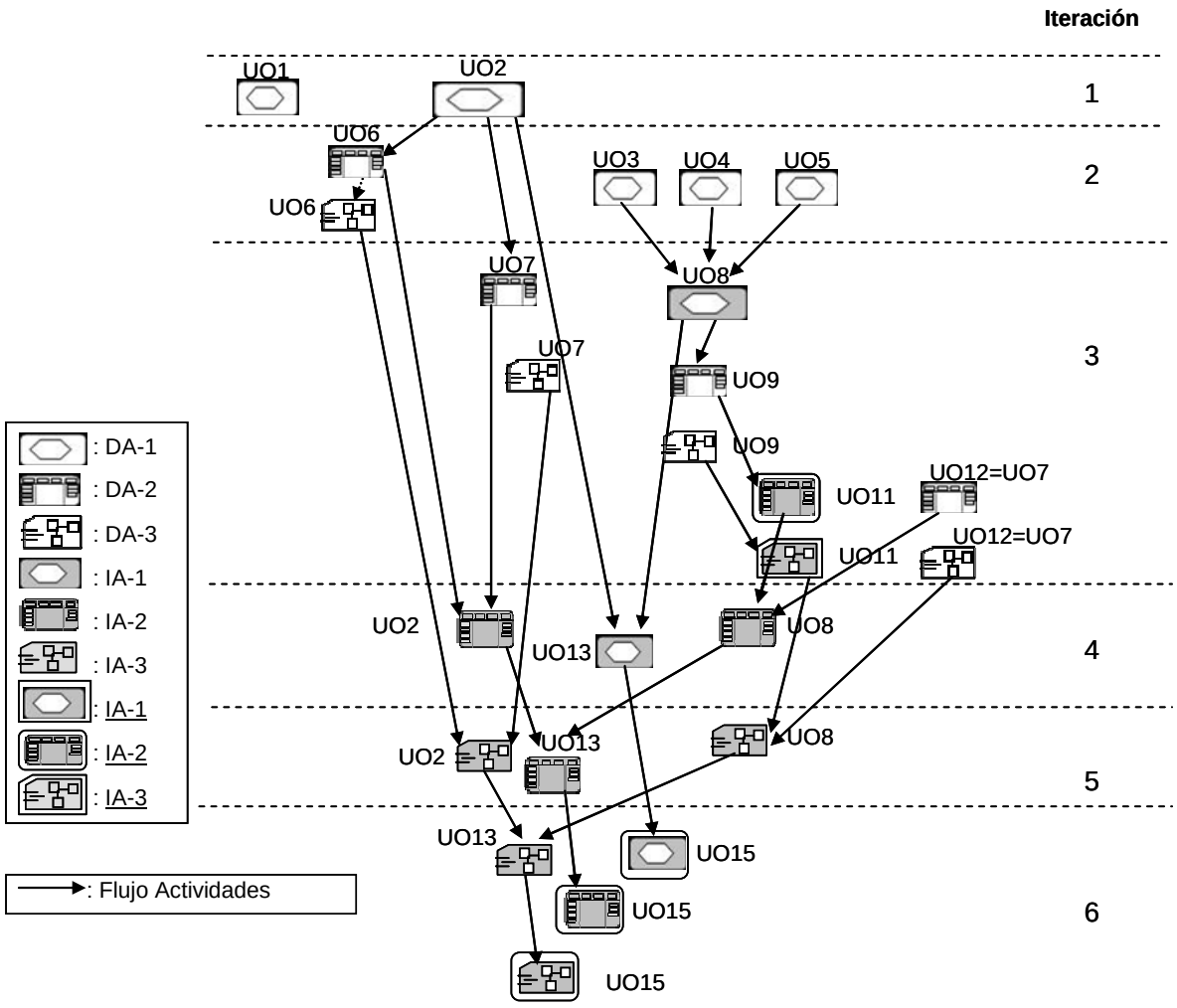


Figura 5.4 Diagrama de actividades planificadas y realizadas en el proyecto FindMyPlace

Los planes de trabajo correspondientes a las siguientes iteraciones fueron:

- En la iteración 3, el primer equipo encontró algunas dificultades para acoplar los planos con la zona a marcar. Además, el programador del primer equipo cayó enfermo y la iteración se retrasó. Estos inconvenientes forzaron a replanificar y rechazar el desarrollo individual de UO8. Por lo tanto, se decidió: (1) dividir la complejidad de UO8 en UO9 y UO10, y (2) desarrollar IA-2(11) y IA-3(11), teniendo en cuenta M-2 (9) y M-3(9) respectivamente. Finalmente, en la iteración 3: (1) fue desarrollada la actividad de integración IA-1 para {UO3, UO4, UO5}, creando así M-1(8), (2) fueron desarrolladas DA-2 y, a continuación DA-3 para UO7; (3) se realizaron DA-2 y, a continuación DA-3

para UO9, y (4) se realizaron IA-2 y, a continuación IA-3 para UO11 desarrollando las necesidades de UO10 sobre UO9. El plan quedó como sigue:

Plan (iteración 3) = {primer equipo [DA-2 (7) → DA-3 (7), DA-2 (9) → DA-3 (9), IA-2(11) → IA-3(11)], segundo equipo [IA-1 (8), Evaluación (M-1 (8), M-2 (7,9,11), M-3 (7,9,11))] }.

Las evaluaciones de todos los modelos creados, fueron realizadas por expertos que utilizaron el protocolo de pensar en voz alta con la técnica de recorrido cognitivo: con los prototipos generados por WebDiagram para M-1 y los prototipos finales para M-2 y M-3. En esta versión, se suscitó una relación especial dependiente de la programación entre el primer y el segundo equipo, porque M-1(8) debía ser evaluado y aprobado antes del inicio de DA-2 (9).

- Plan (iteración 4) = {primer equipo [IA-2 (2), IA-2 (8)], segundo equipo [IA-1 (13), Evaluación (M -1 (13), M-2 (2, 8))] }.

  Además de la evaluación de M-1(13) con prototipos de WebDiagram, se llevaron a cabo evaluaciones heurísticas (Nielsen and Mack, 1994) a través de las interfaces de usuario funcionales desarrolladas con M-2(2) e IA-2(8), para comprobar -antes de la evaluación con los usuarios finales- si la interfaz de usuario cumplía las convenciones de estilo de Android (ver apartado 5.4).

- Plan (iteración 5) = {primer equipo [IA-3(2), IA-3(8), IA-2(13)], segundo equipo [Evaluación progresiva (M-3(2, 8)), M-2(13))] }.

  La funcionalidad de la interfaz de usuario para UO2, evaluado en la iteración 4, se completó mediante la codificación de la lógica de negocio. Los datos de los registros (logs) grabados de las actividades de los usuarios permitió medir aspectos de eficiencia y eficacia (Dumas and Redish, 1993) en el modelo M-3(2). Por último, este modelo se evaluó con un cuestionario de satisfacción y una pequeña entrevista personal desarrollada al final de cada sesión de uso (ver apartado 5.4.4). Los modelos M-3(8) y M-2(13)

siguieron evaluaciones por expertos al estilo de las ya comentadas, ya que tanto UO8 como UO13 son UOs Indirectos.

- Plan (iteración 6) = {primer equipo [IA-3 (13), IA-2 (15) → IA-3(15)], segundo equipo [IA-1(15), Evaluación (M-1(15), M-2(15), M-3 (13,15))] }.

  En esta iteración surge el UO Directo UO14, que se realiza sobre UO15, resultado final de la aplicación y que debe ser evaluado con detalle en todos sus modelos. Al trabajar los dos equipos sobre el mismo UO, deben coordinarse adecuadamente para realizar el trabajo en el orden propuesto. Actualmente, se ha realizado una prueba con usuarios finales y se están analizando los resultados.

La Tabla 5.1 muestra la información de la Hoja de Gestión en la que se resume el estado del proyecto y la planificación de las iteraciones.

Tabla 5.1 Hoja de Gestión del proyecto *FindMyPlace*: Planificación de las iteraciones

Iteración	1	2	3	4	5	6
Lista UOs	1 2 3 4 5	6 7	**8** 9 ~~10~~ ~~**11**~~ *12*	**13**		14 **15**
1er equipo	DA-1(1) DA-1(2)	DA-2(6) DA-3(6)	DA-2(7, 9) → DA-3(7, 9,) IA-2(11) → IA-3(11)	IA-2(2) IA-2(8)	IA-3(2) IA-3(8) IA-2(13)	IA-3(13) IA-2(15)→ IA-3(15)

Iteración	1	2	3	4	5	6
2º equipo	***Evaluación***{ M-1(1, 2)}	DA-1(3) DA-1(4) DA-1(5) ***Evaluación*** { M-1*(1, 3, 4,5), M-2(6), M-3(6)}	IA-1(8) ***Evaluación***{ M-1(8), M-2(7, 9, 11), M-3(7, 9, 11)}	IA-1(13) ***Evaluación***{ M-1(13), M-2 (2,8)}	***Evaluación***{ M-2(13), M-3 (2,8)}	IA-1(15) ***Evaluación***{ M-1(15), M-2(15), M-3 (13,15)}

n: UOn Directo n: UOn división Indirecto **n**: UOn Fusión Indirecto
n: UOn Reutilizado ~~n~~: UOn Incremento ~~**n**~~: UOn Incrementado
DA-k(n): Actividad de Desarrollo donde k = {1,2,3} y n es el número de UO
IA-k(n): Actividad de Integración
IA-k(n): Actividad de Integración Incremental

Finalmente, la Tabla 5.2 es una parte de la Hoja de Gestión, que resume los modelos que fueron desarrollados y evaluados en las diferentes iteraciones del proyecto FindMyPlace.

Tabla 5.2 Modelos desarrollados en *FindMyPlace*. Estado del Proyecto.

UOs	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Fusión		**13**	**8**	**8**	**8**			**13**			**8**	**8**	**15**	**15**	
División						**2**	**2**		**8**	**8**		**8**			
Increm.									**11**	**11**					
M-1	X1	X1	X2	X2	X2	D2	D2	I3	D3	D3	D3	D3	I4	F6	I6
M-2		I4	F4	F4	F4	X2	X3	I4	X3	F3	I3	X3	I5	F6	I6
M-3		I5	F5	F5	F5	X2	X3	I5	X3	F3	I3	X3	I6	F6	I6

Xi: Realizado por una actividad de Desarrollo en la iteración i
Ii: Realizado por una actividad de Integración
Ii: Realizado por una actividad de Integración incremental
Fi: Realizado indirectamente a través de un UO Fusión
Di: Realizado indirectamente a través de un UO División
Fi: Realizado indirectamente a través de un UO Incrementado
El número i expresa la iteración en la que se evaluó satisfactoriamente dicho modelo. Las celdas sombreadas corresponden a la realización indirecta de modelos .

Esta tabla permite confirmar la relación de precedencia establecida entre los momentos de creación de los modelos para cada UOi: M-1(i) < M-2(i) < M-3(i), donde '<': orden de realización (creación y evaluación). Sin embargo, en algunas iteraciones, se observa que M-1(i) <= M-2(i) <= M-3(i). Esto es debido a que aunque en este proyecto se muestran las iteraciones cruciales que han ocasionado los planes de actuación principales, en aras de mayor agilidad, cada una de las iteraciones suponía un subconjunto de iteraciones internas, esto es, con evaluaciones rápidas del equipo de desarrolladores.

Esta relación de orden en la realización de los modelos no se cumple para los diferentes UOs; por ejemplo, se observa en las columnas de la tabla que M-3(6) (iteración 2) < M-2(7) (iteración 3) < M-1(13) (iteración 4). La evolución del proyecto no es predecible; en las reuniones de trabajo, los desarrolladores y gestores del proyecto elegirán las actividades a realizar en cada punto del proceso de desarrollo, teniendo en cuenta los avances y las necesidades del proyecto. Además, es necesario indicar que el desarrollo de UO1 ha sido indefinidamente aplazado, ya que salvo la finalización pendiente de la evaluación de la primera versión de FindMyPlace , el proyecto está finalizado.

5.4 Aplicación de técnicas de evaluación de usabilidad en el proyecto FindMyPlace

Nielsen define la disciplina de la Ingeniería de Usabilidad como el proceso de la Ingeniería de Software que incluye consideraciones de usabilidad en el proceso de desarrollo de software (Nielsen, 1993). Una de las características importantes de la IngUS es la inclusión de una especificación de usabilidad. Por lo tanto, la especificación de los requisitos debe formalizar las características de la interacción usuario-sistema que contribuye a la usabilidad del producto (Dix et al., 2003). En este sentido, el modelo SE-HCI captura este tipo de interacciones.

En cuanto a los desarrollos móviles, Nielsen realizó una serie de estudios que indicaron que los usuarios de móviles se enfrentan a graves problemas de usabilidad en el intento de alcanzar sus objetivos en los sitios web, ya sea a través de los sitios para móviles especializados o en los sitios optimizados para escritorio tradicionales, que se presentan a

través de un navegador móvil (Nielsen, 2011). De hecho, los métodos y guías tradicionales usados en las evaluaciones de usabilidad para aplicaciones de escritorio no pueden ser aplicados directamente en un entorno móvil. Por lo tanto, no es trivial para los métodos de evaluación cumplir con la necesidad de integrar adecuadamente situaciones reales o contextos simulados durante el proceso de evaluación. Ha surgido una gran controversia en torno a la bondad y relevancia de los resultados de las pruebas de usabilidad móvil cuando se consideran factores de movilidad y contextuales. Se realizaron varios estudios para contrastar los estudios de laboratorio con las pruebas de campo, y sus conclusiones sugieren que, en la mayoría de los casos, se obtiene una respuesta más relevante a partir de estudios en laboratorio (Kaikkonen et al., 2005). Otros autores llegaron a la conclusión de que algunas prácticas del DCU, por sí solas, no son suficientes para los teléfonos móviles debido a su pequeño tamaño y sus capacidades de entrada de usuario limitadas (Kangas and Kinnunen, 2005). Sin embargo, cabe señalar que ha habido una gran evolución en estos dispositivos, que se evidencia en grandes tamaños de pantalla, y un aumento significativo en las capacidades de interacción.

Todos los modelos obtenidos por las actividades de desarrollo e integración fueron evaluados con diversas técnicas de usabilidad (Losada et al., 2013b). En esta sección se expone cómo se llevaron a cabo algunas evaluaciones de usabilidad en el proceso de desarrollo ágil para FindMyPlace. En concreto, se explica cómo se utilizaron las técnicas de usabilidad en los siguientes hitos de InterMod: Paso 0. - *Analizar el proyecto en general* y en las evaluaciones de M-1. *Modelo de Requisitos*, M-2. *Modelo de Presentación* y M-3. *Modelo de Funcionalidad.*

5.4.1 Análisis inicial del proyecto mediante técnicas de usabilidad (Paso 0)

En el paso 0 del proyecto, se preparó una sesión de usuario con el fin de obtener información sobre los usuarios finales e identificar los UOs principales de la aplicación. La colaboración de un grupo de usuarios finales es fundamental para determinar los UOs de partida que se recogen de la información obtenida por medio de métodos de IngUS. Los criterios de

selección de UOs se basan en el descubrimiento de las funcionalidades necesarias y sus prioridades.

Objetivos: Obtener información acerca de los usuarios finales, sus necesidades y prioridades. Esta información será la base para la definición de la lista inicial de UOs, y comenzar así el proceso de desarrollo de la aplicación FindMyPlace.

Participantes: 20 voluntarios participaron en la sesión de usuario. Todos ellos eran estudiantes en primer año de Grado en Informática de la Universidad del País Vasco. También se eligieron como posibles usuarios finales de la aplicación. Además, todos los desarrolladores del proyecto asistieron a la reunión con el fin de responder a las preguntas de los usuarios.

Instrumentos: Cuestionario desarrollado en formato Google Docs. El elevado número de usuarios de la sesión y la libertad de expresión de las opiniones, sin interferencia de otros usuarios, fueron factores que se consideraron para seleccionar esta técnica. Se dividió en tres partes: (A) Características demográficas de los usuarios: conexión a Internet en el teléfono móvil, uso de un Smartphone o un teléfono básico, uso del GPS en el teléfono, etc, (B) un cuestionario abierto recogió, a través de un cuadro de texto, todas las características posibles que a los usuarios les gustaría que tuviera la aplicación, (C) un cuestionario estructurado sobre 13 posibles necesidades de usuario, propuestas por los desarrolladores del proyecto. Mediante el uso de escalas Likert, los usuarios tuvieron que evaluar cada necesidad, indicando su grado de acuerdo y prioridad.

Procedimiento: Todos los estudiantes fueron convocados a una reunión con el fin de completar el cuestionario. Primero, recibieron una breve explicación sobre el propósito de FindMyPlace y luego, se expusieron la sesión en general y las diferentes partes del cuestionario. Finalmente, se les animó a hacer preguntas sobre cómo completar el cuestionario, que fueron respondidas por los desarrolladores del proyecto, presentes en el laboratorio. Todo el proceso duró aproximadamente 20 minutos. Se realizó una segunda reunión de trabajo, con desarrolladores y gestores del proyecto, para analizar la información de los usuarios con el fin de extraer las necesidades o deseos funcionales y no funcionales a tener en cuenta en la aplicación.

Resultados: La información recogida permitió clasificar a los usuarios finales en tres grupos según su inmersión en el uso de tecnología móvil avanzada (profunda, media, baja). Además, se identificaron sus deseos funcionales y no funcionales con respecto al comportamiento de la aplicación. El 80% de ellos tenía teléfonos móviles con conexión a Internet, el 70% tenía un teléfono inteligente y el 45% utilizaba, habitual u ocasionalmente, mapas en sus teléfonos móviles. El 60% utilizaba la plataforma Android, lo que confirmó su elección como plataforma de la aplicación. Los deseos funcionales identificados dieron lugar a una lista con los cinco UOs iniciales (apartado 5.3.1).

5.4.2 Evaluación del M-1. Modelo de Requisitos

Las actividades DA-1 desarrolladas para UO1 y UO2 generaron los modelos de Requisitos M-1 en la iteración 1. Estos modelos fueron evaluados sobre prototipos generados con la herramienta WebDiagram (Losada et al., 2009c), descrita en el apartado 4.3.1. Los evaluadores realizaron las tareas incluidas en los modelos SE-HCI de ambos UOs, y expresaron sus opiniones de acuerdo con la técnica de pensar en voz alta (C. Lewis, 1982). Se identificaron varios aspectos problemáticos relacionados con la terminología utilizada en la interfaz, especialmente para UO1. Después de corregir estos problemas, se propuso una evaluación complementaria con usuarios finales y prototipos más realistas para UO1. Sin embargo, UO2 estaba listo para continuar su desarrollo.

En la iteración 2, se aplicaron diferentes técnicas de usabilidad con usuarios finales para evaluar M-1 para los UOs {1, 3, 4, 5}. Se utilizaron prototipos de papel (Mayhew, 1999) desarrollados a partir de los diagramas de navegación SE-HCI anteriores, la técnica de pensar en voz alta (C. Lewis, 1982), la observación y las entrevistas [Nielsen, Mack 94]. La sesión de usuario se llevó a cabo de la siguiente manera:

Objetivo: Detectar problemas en la interfaz de usuario relacionadas con la navegación y la interacción antes de desarrollar código.

Participantes: Participaron 18 voluntarios (de los 20 iniciales) y estuvieron presentes 3 expertos en usabilidad para guiar la sesión y actuar como evaluadores.

Instrumentos: Los expertos desarrollaron un prototipo de papel compuesto por 16 pantallas que representaban las interfaces de usuario para todos los UOs a evaluar. Las pantallas

incluían las características establecidas en el modelo SE-HCl sobre la navegación, el comportamiento y otros aspectos de la interfaz de usuario. La Figura 5.5a. muestra algunas de las pantallas y un grafo de navegación dependiente de las acciones realizadas en la interfaz. Las 16 pantallas se compilaron en un cuaderno, y se utilizó una serie de "post-it" para simular correctamente la secuencia de navegación a través del prototipo (Figura 5.5 b.). Se prepararon tres cuadernos, uno para cada evaluador.

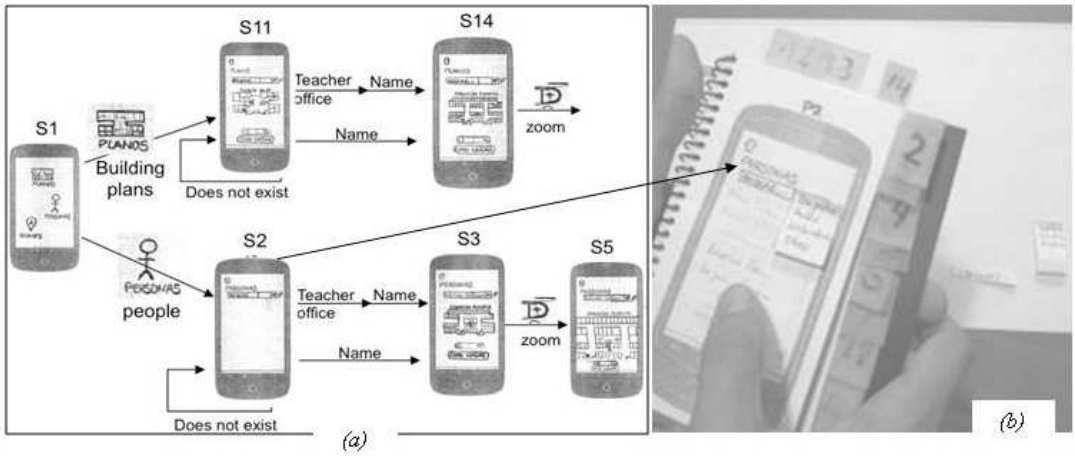


Figura 5.5 Evaluación de UO3: (a) secuencia de pantallas y (b) el prototipo de papel para S2

Procedimiento: Un evaluador, al lado del usuario, presentó el cuaderno de prototipos y realizó la función de simulador. De este modo, podía replicar la respuesta del sistema para cada acción del usuario mediante la adición de post-it en la pantalla adecuada, y a continuación, navegar a través de las pantallas-prototipo.

Los evaluadores presentaron una serie de escenarios de Objetivos de Usuario para ser ejecutados a través del prototipo de papel. La Tabla 5.3 muestra cada escenario de UO en la que X e Y representan los nombres hipotéticos de un profesor y un laboratorio, respectivamente. Todos los usuarios realizaron todos los escenarios de UOs ordenados en una secuencia generada aleatoriamente. De esta manera se evitan los posibles efectos de correlación derivados del orden de ejecución. Los evaluadores anotaron todas las acciones de los usuarios y sus comentarios. Al final, se hizo una entrevista breve para recoger la opinión de los usuarios sobre el sistema y los aspectos susceptibles de mejora o inclusión. La sesión duró aproximadamente 30 min / usuario.

Tabla 5.3 Escenarios de UOs y porcentaje de culminación

UO	Escenarios de UOs	Logro
1	Estoy en la entrada y quiero que me guíe al despacho del profesor X	16.67%
3	Muéstrame el despacho del professor X en el plano del edificio	88.89%
4	Muéstrame la localización del laboratorio Y en el plano del edificio	94.44%
5	Muéstrame la localización de la cafetería en el plano del edificio	100%

Resultados: La Tabla 5.3 incluye además el porcentaje de finalización y logro de los escenarios de UO. A partir de los datos, es evidente que UO1 requiere un importante proceso de rediseño iterativo, ya que la mayoría de los usuarios no pudieron completarlo. La posibilidad de determinar este hecho en una etapa tan temprana de su desarrollo es una clara ventaja que proporciona el modelo SE-HCI. Por otra parte, los UOs relacionados con la búsqueda de localizaciones específicas en los planos del edificio fueron completados casi siempre, aunque aún se necesitan cambios menores.

Además de las medidas de eficacia indicadas, los evaluadores también recolectaron datos para evaluar la eficiencia (véase Tabla 5.4). Esta tabla resume el porcentaje de los caminos no óptimos, los clics incorrectos efectuados en los caminos correctos, y el tecleado incorrecto de información. Debe tenerse en cuenta que estos datos proporcionan información sólo para los escenarios de UO que se han completado con éxito.

Tabla 5.4 Medidas de eficiencia para los Escenarios de UOs

Medidas de eficiencia	UO3	UO4	UO5
Caminos no óptimos	6.25	11.76	16.67
Clics en elementos incorrectos dentro del camino correcto	0	1.33	1.00
Datos de entrada de texto incorrectos	0	1.67	3.00

"Caminos no óptimos" expresa el porcentaje de usuarios que siguió un camino no óptimo para cumplir con un escenario de UO. Se observó también, el número de clics que los usuarios hicieron en los elementos de la interfaz de la ruta correcta, pero que no dieron lugar a la realización correcta del escenario. Por último, también fue registrado el número de veces que los usuarios escribieron datos incorrectos en un cuadro de texto.

Las medidas de eficiencia proporcionan una valiosa información sobre el uso de la interfaz y los aspectos que se deben mejorar. Por ejemplo, el escenario de UO5 fue donde los usuarios introdujeron más datos incorrectamente (Tabla 5.4), a pesar de ser el más eficaz (100% en la Tabla 5.3). Por lo tanto, en este caso, las medidas de eficiencia indican la necesidad de mejorar la interfaz de usuario, pero también demuestran que es lo suficientemente buena para que los usuarios se recuperen de los errores que cometen. También se aplicó un t-test para la tasa de escenarios de UO completados, rutas no óptimas seguidas y clics incorrectos efectuados, para los tres grupos de usuarios. En todos los casos el valor de $p > 0{,}20$, y al aplicar un t-test a los datos incorrectos, el valor de $p > 0{,}08$. Estos hechos son destacables ya que indican que no existen diferencias significativas, estadísticamente hablando, en el rendimiento del usuario cuando se considera la plataforma móvil, el tipo de teléfono y el uso de mapas.

5.4.3 Evaluación del M-2. Modelo de Presentación

En la iteración 4 se utilizó el método de inspección de la evaluación heurística para determinar la idoneidad del modelo M-2 de UO2 (ver Figura 5.4). El proceso de evaluación realizado se explica a continuación:

Objetivo: Detectar problemas de usabilidad en la interfaz de usuario al principio del proceso de desarrollo.

Participantes: Tres expertos en usabilidad realizaron la evaluación.

Instrumentos: Se instaló una versión funcional de la aplicación que implementa UO2 en la plataforma móvil de un Samsung Galaxy S. Cada evaluador utilizó un teléfono móvil para evaluar la interfaz de usuario de acuerdo con los ocho principios heurísticos de Bertini, que se dividieron en 35 sub-heurísticos (Bertini et al., 2009). Los sub-heurísticos se evaluaron según la escala de clasificación de gravedad (SRS) (Nielsen and Mack, 1994), que clasifica los

errores de usabilidad en función de su importancia. Los sub-heurísticos se analizaron también para determinar si eran aplicables o no aplicables (NA).

Procedimiento: Como primer paso, cada evaluador estudió la interfaz y después se comprobó todos los heurísticos y sus correspondientes sub-heurísticos. Cada sub-heurístico aplicable fue clasificado por los expertos de acuerdo con la escala SRS de Nielsen, que va de 0-4 (no hay problema de usabilidad en absoluto a la catástrofe total en usabilidad). Una vez terminados los procesos individuales de evaluación, los evaluadores se reunieron, debatieron sus conclusiones y los acordaron en una lista.

Resultados: La Tabla 5.5 muestra los resultados. Aunque el objetivo de la evaluación fue la identificación de posibles errores de usabilidad según los sub-heurísticos aplicables, los sub-heurísticos NA también fueron representados. Los resultados no mostraron ninguna catástrofe de usabilidad. Cuarenta y tres por ciento de los sub-heurísticos no presentó ningún problema de usabilidad, y más de una cuarta parte de los sub-heurísticos fueron NA. Los dos principales problemas de usabilidad encontrados estaban relacionados con el número limitado de formas en que la aplicación podía ser manipulada. Por ejemplo, la manipulación con una sola mano, un escenario muy probable en un contexto móvil, es una habilidad motora difícil para el usuario. Se detectaron algunos problemas de usabilidad menores como:

- La imagen es pequeña en modo horizontal,
- Las imágenes no se muestran como una galería de fotos, carecen de continuidad,
- La distribución de botones en la pantalla de inicio, en modo horizontal, es inadecuada,
- Hay una falta de opciones de personalización en la aplicación.

Por último, todavía quedaban algunos pequeños problemas sobre la lectura y la navegación por los contenidos de la pantalla, lo que requería:

- Una mayor flexibilidad en la inicialización de los elementos de la interfaz de usuario,
- Mejorar la estética de los iconos cuando se seleccionen y se muestren,

- Aclarar dos mensajes de error relacionados con la visualización de los planos del edificio.

Tabla 5.5 Resultados de la evaluación heurística en el prototipo

SRS	NA	0	1	2	3	4
Número de sub-heuristicos	9	15	4	5	2	0
Porcentaje	0.26	0.43	0.11	0.14	0.06	0.00

5.4.4 Evaluación del M-3. Modelo de Funcionalidad

El modelo M-3 de UO2 se evaluó en la iteración 5 por medio de técnicas combinadas de inspección como el recorrido cognitivo, métodos de indagación como la observación directa y los cuestionarios, y otros métodos basados en test como las medidas de rendimiento mediante análisis de logs.

Objetivo: Medir la eficacia, eficiencia y satisfacción en el uso del UO desarrollado

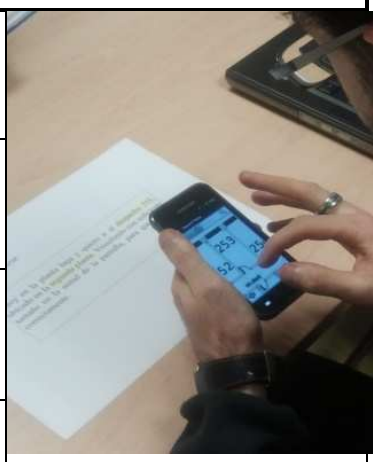
Participantes: En el momento en que se llevó a cabo esta evaluación, se trataba de un nuevo año académico, por lo que fue necesario reclutar a un nuevo grupo de usuarios finales. El grupo estaba formado por 12 estudiantes en su primer año de universidad. Se comprobó su adecuación con el modelo de usuario. Todos los usuarios finales nuevos disponían de smartphones en propiedad, el 91% de los cuales eran teléfonos móviles con conexión a Internet y el 66% tenían plataforma Android. Además, el 83% de los usuarios utilizaban mapas, habitualmente u ocasionalmente, en sus teléfonos móviles.

Materiales e instrumentos: Una vez más, se utilizaron teléfonos móviles Samsung Galaxy S. Tenían una aplicación que ejecutaba todas las funcionalidades relacionadas con UO2, y un módulo para recoger los registros de uso durante la interacción del usuario en función de la biblioteca de Android-Logging-Log4J. Se usó el cuestionario demográfico utilizado en el paso 0, para comprobar la adecuación del nuevo grupo de usuarios. Al final de la sesión de pruebas, los usuarios tenían que completar el cuestionario de satisfacción CSUQ (Lewis, 2001). Esto incluía 19 sentencias que los usuarios clasificaron de 0 a 9 dependiendo de su

grado de acuerdo. Los usuarios también podían marcar una sentencia como no aplicable para la tarea, y hacer cualquier otro comentario al respecto.

Procedimiento: En primer lugar, los usuarios finales rellenaron el cuestionario demográfico para comprobar que se ajustaban al modelo de usuario. Luego, se llevaron a cabo cuatro tareas en la aplicación sin límite de tiempo. Al igual que en las evaluaciones de M-1 en la iteración 2, los evaluadores presentaron a los usuarios una serie de escenarios de UO a realizar, también en un orden seleccionado al azar (ver Tabla 5.6).

Tabla 5.6 Escenarios de evaluación para UO2. Escena de un usuario durante la evaluación de la tarea 4

Tarea	**Escenarios de evaluación - UO2**	
1	Estoy en la planta baja y quiero ir a la cafeteria, situada en la planta baja. Visualízala en el centro de la pantalla, con el tamaño adecuado.	
2	Estoy en la planta baja y quiero ir al laboratorio E08, situado en la entreplanta. Visualízalo en el centro de la pantalla, con el tamaño adecuado.	
3	Estoy en la planta baja y quiero ir a la secretaría del centro, situada en la primera planta. Visualízala en el centro de la pantalla, con el tamaño adecuado.	
4	Estoy en la planta baja y quiero ir al despacho 253, situado en la segunda planta. Visualízalo en el centro de la pantalla, con el tamaño adecuado.	

Después de completar todas las tareas, los participantes rellenaron el cuestionario de satisfacción, y luego hubo una breve entrevista con los usuarios para que pudieran expresar su opinión sobre la aplicación. La sesión duró aproximadamente 15 min. / usuario.

Proceso y resultados del estudio: El archivo de logs obtenido por la aplicación necesitó un pre-proceso para facilitar su estudio posterior, ya que era demasiado extenso (Figura 5.6a).

Hay dos tipos principales de acciones del usuario sobre la interfaz: presionar algunos botones, y tocar los planos del edificio para moverlos o agrandarlos. Por lo tanto, sólo se requería una parte de la información recopilada en el archivo de logs, esto es: el tiempo, el botón pulsado (por ejemplo, "planos del edificio") y la identificación de la acción en la pantalla expresado por un código numérico (por ejemplo, 0, 261, 2, etc.).

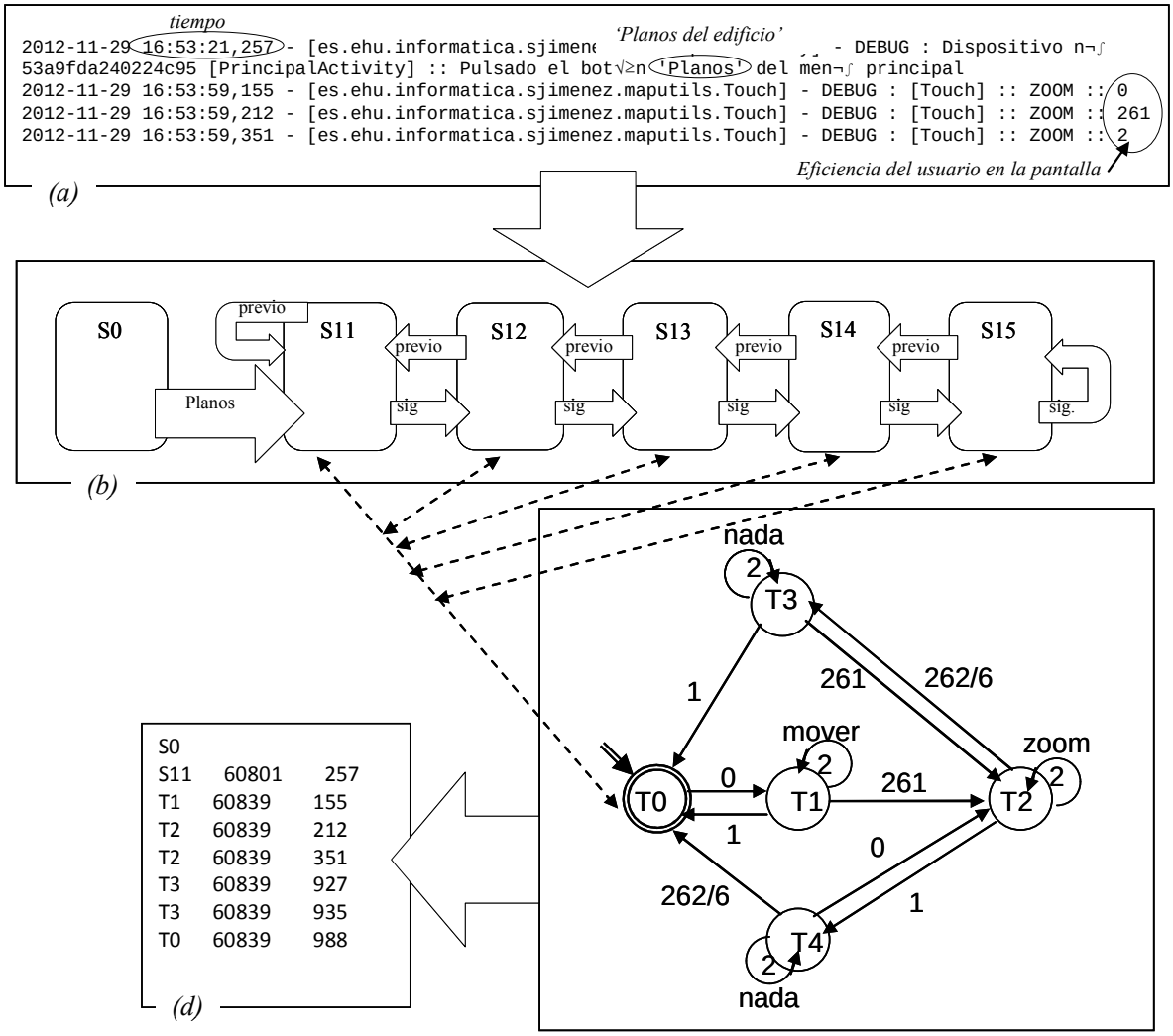


Figura 5.6 Pre-proceso del fichero de logs: (a)Fragmento *(c)* del fichero, (b) Secuencia de pantallas para UO2 generadas por la pulsación de botones, (c) Finite State Machine, y (d) nuevo fichero de logs

Teniendo en cuenta los botones que podrían ser pulsados, el modelo de navegación de UO2 está representado por la secuencia de pantallas que se muestran en la Figura 5.6b (por ejemplo, S0, S11, S12, etc) y los botones "planos del edificio" y plano “siguiente”/“anterior”. Cada pantalla (estado) permite que el usuario pulse en una planta del edificio con el fin de moverse o hacer zoom en dicha planta.

Las respuestas de la interfaz están representadas por una máquina de estados finitos (FSM) (Figura 5.6c). El pre-proceso interpreta los logs y asocia las actividades de los usuarios tanto con la secuencia de estados en la pantalla como con los estados de la FSM. El estado "T0" corresponde a todas las pantallas posibles (de S11 a S15). El nuevo fichero de estados resultante es más fácil de leer, comprender y tracear (ver Figura 5.6.d). Cada línea describe el

estado de la interacción en términos de tres datos: estado (pantalla y toque) y el tiempo (segundos y milisegundos). En el ejemplo, se observa en el nuevo fichero: (1) el estado S0 que corresponde a la pantalla principal de la aplicación FindMyPlace, (2) S11 que es el estado alcanzado después de pulsar el botón "planos del edificio", y (3) las siguientes líneas que representan una serie de seis actividades de pulsado (Ti). Estas son: colocar el primer dedo en la pantalla (T0–0→T1), colocar el segundo dedo en la pantalla (T1–261→T2), hacer zoom sobre un área de la planta (T2–2 -> T2), retirar el segundo dedo de la pantalla (T2–262/6→T3), mover el segundo dedo a través de la pantalla sin efectos sobre la interfaz (T3–2→T3), y retirar el primer dedo (T3–1→T0).

Después de terminar las evaluaciones y filtrar los archivos de logs, se obtuvieron los datos relevantes de rendimiento: (1) la eficacia se calculó teniendo en cuenta el porcentaje de finalización de las tareas y el número de interacciones de usuario erróneas, y (2) la eficiencia se estimó considerando el tiempo de realización para cada tarea y su desviación de las rutas óptimas de acuerdo con ciertos patrones dados. La Tabla 5.7 muestra la media de los datos recogidos.

Tabla 5.7 Valores medios de las medidas de eficacia y eficiencia, derivadas del análisis de logs

Tarea	**Tiempo (ms)**	**N. interacciones**	**Finalización**	**Interacciones erróneas**
1	26809.58	33.58	1	0.17
2	27341.25	36.25	1	0.75
3	22611.00	25.33	1	0.25
4	34925.00	49.67	1	0.08

Todos los usuarios completaron todas las tareas, pero algunos se perdieron en algún momento porque no sabían qué tipo de interacciones se podían realizar. El tiempo empleado para completar las tareas fue claramente bajo. Sólo una tarea duró más de 30 segundos (en valor medio). El número de interacciones realizadas parece alta, ya que va de una media de 25,33 a 49,67, pero creemos que se justifica por el hecho de que la localización de un lugar en los planos del edificio, cuando se utiliza una pequeña pantalla, requiere muchos movimientos

e interacciones de zoom. Por último, las interacciones erróneas fueron muy bajas, y sólo 6 usuarios realizaron interacciones fuera de la ruta óptima. Los usuarios efectuaron 15 interacciones erróneas de un total de 1738 interacciones realizadas para completar el conjunto de tareas; esto es, menos de un 1% del total.

En general, los resultados de la encuesta de satisfacción fueron positivos. Los usuarios consideraron que las sentencias 9 y 10, que se refieren a mensajes de error, no eran aplicables, ya que las interacciones de la mayoría de ellos no daban lugar a mensajes de error. Por otro lado, 9 sentencias se clasificaron entre 8 y 9; incluían aspectos relacionados con la facilidad con la que se podía utilizar el sistema, la utilidad de la aplicación, la comodidad y la percepción de la capacidad del usuario para completar las tareas. Las dos sentencias con respecto a la claridad de la información y la facilidad para localizar la información se clasificaron con una media de 7. Los usuarios calificaron tres preguntas con un valor promedio de entre 6 y 7: la claridad en la organización en las pantallas, si la información proporcionada era la necesaria para completar las tareas, y si la interfaz de usuario era atractiva. La comodidad al usar la interfaz obtuvo un 5,75. La sentencia 18: "Este sistema incluye todas las funcionalidades y capacidades que espero que tenga", fue el índice más bajo, aunque a los usuarios se les dijo que la evaluación cubría sólo una parte del sistema. Finalmente, la sentencia 19, que se trataba de la satisfacción general del usuario con el sistema, recibió un valor medio de 6,67 sobre 9.

Los comentarios de los usuarios incluidos en el cuestionario se complementaron con entrevistas breves. Algunos puntos interesantes surgieron de ambas técnicas. En primer lugar, algunos usuarios no eran conscientes de la planta en la que estaban. Aunque el indicador de planta se visualizaba en texto, en la parte superior de la interfaz, cerca del icono de inicio, no se reconoció como parte de la interfaz. Otra observación pertinente solicitó código de colores para los diferentes tipos de espacios (por ejemplo, oficinas, aulas o laboratorios); esto habría sido útil para la identificación de los espacios y hacer la búsqueda más rápida.

5.4.5 Lecciones aprendidas

InterMod proporciona soluciones para integrar adecuadamente la usabilidad en una metodología de desarrollo ágil. Las metodologías ágiles tradicionales, que no tienen en cuenta

los aspectos de usabilidad, tienen problemas con respecto a cuándo y cómo integrarla en el proceso de desarrollo. Estos problemas surgen del hecho de que los modelos ágiles impulsan iteraciones cortas y un diseño up-front mínimo. Hay que tener en cuenta que, en este contexto, los cambios en el proceso de desarrollo pueden ocurrir en cualquier etapa ("Manifesto for Agile Software Development", 2001). Los continuos cambios en la interfaz de usuario que se derivan de un diseño iterativo rápido pueden causar conflictos con las expectativas del usuario, ya que pueden dar lugar a incoherencias e insatisfacción (Constantine and Lockwood, 2002). La propuesta de integración de InterMod comparte algunas de las soluciones a estos problemas de integración que se han presentado en la literatura, y su proceso ágil facilita la simbiosis adecuada entre la agilidad y la evaluación de la usabilidad.

La aplicación de técnicas de evaluación de usabilidad en las primeras etapas del proceso de desarrollo ha demostrado dos ventajas principales. En primer lugar, la obtención de la lista inicial de UOs al inicio del proyecto, junto con la evaluación continua de la usabilidad con los usuarios finales, ha promovido un desarrollo adaptado a las expectativas de los usuarios. En segundo lugar, el uso de prototipos de papel antes de la implementación, cuando se utiliza en las primeras etapas del proceso de desarrollo y se centra en comprobar la navegación y el diseño, ha demostrado ser adecuado para la validación de los modelos de requisitos. Aunque FindMyPlace incluye interacciones como zoom de mapas o la manipulación directa, nuestros resultados contradicen las referencias que indican que la creación de un prototipo de papel no es adecuado para el desarrollo de aplicaciones para teléfonos móviles (Kangas and Kinnunen, 2005).

Gracias a la evaluación temprana del modelo SE-HCI, fuimos capaces de detectar desde el principio que UO1 requería un mayor proceso iterativo de rediseño, por lo que organizamos el proyecto en consecuencia.

Al mismo tiempo, hemos podido confirmar que la realización de las evaluaciones con los usuarios finales en cada paso del proceso como por ejemplo sugiere Mayhew (Mayhew, 1999), queda fuera del ámbito de aplicación en metodologías ágiles, ya que el tiempo necesario para definir y preparar las pruebas de usuario va en contra de la filosofía ágil. Por lo tanto, debe haber un equilibrio entre agilidad y evaluación con el fin de mezclar ambos aspectos en el desarrollo del proyecto. Con el fin de realizar evaluaciones con los usuarios finales, es deseable planificar la creación paralela de modelos para diferentes UOs en la

misma iteración. De esta forma, las sesiones de evaluación de usabilidad con usuarios finales pueden agruparse sin retrasar el desarrollo del proyecto. Como consecuencia directa, el tiempo que los usuarios dedican a las evaluaciones se reduce y los criterios de evaluación pueden unificarse. Por ejemplo, las evaluaciones con usuarios finales de varios modelos M-1 se realizaron conjuntamente en la iteración 2. Sin embargo, las evaluaciones con los prototipos deben llevarse a cabo en su presencia con el fin de recoger opiniones, ideas e impresiones.

La crítica más frecuente hacia las metodologías ágiles es su falta de formalidad, porque la obtención del producto parece tener prioridad sobre el tiempo invertido en su diseño. Sin embargo, el modelo SE-HCI propuesto por InterMod incluye la formalización de los requisitos recogidos y también promueve la evaluación temprana. Además, gracias a Internet y a las nuevas tecnologías, ciertos tipos de evaluaciones de usuario como los cuestionarios pueden no requerir la observación directa del evaluador.

En cuanto a los pasos posteriores en el proceso de desarrollo, el uso de la evaluación heurística nos proporcionó información limitada. En el contexto de InterMod, se aplicó sólo para desarrollos relacionados con algunos UOs. El conjunto seleccionado de heurísticos (Bertini et al., 2009) ha demostrado ser valido. Sin embargo, como se desarrollaron antes de la popularidad de los dispositivos multi-táctiles, no se ocupan de aspectos que hubieran beneficiado a la evaluación de éstos, como por ejemplo, la semántica de la interacción táctil.

Otra cuestión pertinente, relativa a las evaluaciones, radica en cómo recoger los datos en entornos móviles. Hemos desarrollado un módulo para analizar los logs de las interacciones en las pruebas de usuario. Este módulo identifica la pantalla en la que el usuario se encuentra en cada momento, y extrae la información acerca de las actividades (movimiento o zoom) que está realizando para explorar los planos del edificio. Esperamos que este módulo sea particularmente útil para estudios futuros con desarrollos para móviles.

Nuestra experiencia en el desarrollo de software con InterMod sugiere que la mayoría de las evaluaciones deben ser realizadas por un equipo multidisciplinar, compuesto por los usuarios finales y los miembros del equipo de desarrollo. Creemos que es conveniente incluir al menos un especialista en pruebas de usabilidad con el objetivo de diseñar y optimizar el proceso de evaluación adecuado, así como determinar la frecuencia de evaluación con los

usuarios finales. Las evaluaciones de usabilidad deben ser realizadas porque estos procesos aseguran una mejor toma de decisiones en los procesos de desarrollo, y evitan perder tiempo en caminos equivocados y su posterior corrección. Por lo tanto, conseguir un seguimiento adecuado del proyecto sin causar tedio al usuario final es un reto.

5.5 Resumen

En este capítulo, se trata la integración de la IngUS en InterMod, que se efectúa en el Paso 0 y el Paso 3.i de la metodología. Además, se muestra el proceso de desarrollo paso a paso de la aplicación FindMyPlace y se indica cómo se han aplicado de las técnicas de evaluación de usabilidad en los puntos de integración propuestos con InterMod. Esto permite obtener conclusiones valiosas sobre la integración de la Ingeniería de la Usabilidad con InterMod, y por extensión con las metodologías ágiles.

Cada tipo de modelo se adapta mejor a ciertos tipos de evaluaciones. Además se ha presentado un módulo para analizar los logs asociados al modelo M-3. Las técnicas de evaluación de la usabilidad aplicadas en FindMyPlace se efectúan sobre los modelos afectados en cada paso, y se resumen en la Tabla 5.8.

Tabla 5.8 Técnicas de evaluaciónde usabilidad aplicadas en FindMyPlace, por modelos

Paso	**Modelo**	**Técnicas de Evaluación de la Usabilidad**
0	M.Usuario y M.Sistema	Métodos de Indagación: Cuestionarios, encuestas, entrevistas, observación de campo
3	M-1	Métodos de Indagación: Observación de campo, entrevistas Métodos de inspección: Recorrido cognitivo sobre prototipos software y de papel Métodos de test: Medida de las prestaciones, pensar en voz alta
	M-2	Métodos de inspección: Evaluación heurística, inspección de estándares

Paso	Modelo	Técnicas de Evaluación de la Usabilidad
	M-3	Métodos de Indagación: Observación de campo, entrevistas, cuestionarios de satisfacción, registro de uso Métodos de inspección: Recorrido cognitivo Métodos de test: Medida de las prestaciones, pensar en voz alta

Finalmente, el último capítulo de esta memoria presenta la conclusión del trabajo y muestra las líneas futuras de trabajo.

CAPÍTULO 6. Conclusión y líneas futuras de trabajo

A lo largo de este documento se ha descrito la metodología InterMod para el desarrollo de aplicaciones interactivas: su motivación, características y aplicación. InterMod es una metodología ágil que tiene como objetivo ayudar en el desarrollo adecuado de software interactivo de alta calidad. La propuesta que se ha presentado aquí integra los principios de las Metodologías Ágiles (MA) con las disciplinas del Diseño Centrado en el Usuario (DCU) y Desarrollo Dirigido por Modelos (DDM). Se resume a continuación esta propuesta y se indican las aportaciones del trabajo presentado. Después, se indican las líneas futuras de trabajo y finalmente, se presentan las publicaciones presentadas en revistas, congresos y secciones de libros.

6.1 Resumen del trabajo presentado

La metodología InterMod propone que los proyectos de software interactivo se organicen en una serie de iteraciones guiadas por los Objetivos de Usuario (UOs), de una manera ágil, centrada en el usuario y dirigida por modelos. Un UO es o deriva de un deseo del usuario que se puede lograr por uno o más (al menos uno) requisitos funcionales y/o uno o más requisitos

no funcionales de usuario. Cada UO es una unidad lógica, cuyo desarrollo, a través de modelos evaluados constantemente, es una parte del resultado final. En las diferentes iteraciones del proceso, los equipos de desarrollo trabajan en paralelo y realizan diferentes actividades de desarrollo e integración para obtener los modelos que completan los UOs.

La disciplina DCU asegura que el desarrollo se ajusta a las necesidades del usuario, y el DDM formaliza el proceso y facilita las continuas integraciones. Finalmente, el proceso ágil en InterMod permite trabajar en diferentes modelos de diferentes UOs en paralelo. En concreto, se basa en tres modelos: el M-1 o modelo SE-HCI (Semantically Enriched Human-Computer Interaction) describe los requisitos de una manera enriquecida, el M-2 recoge aspectos de presentación y el M-3 describe el modelo de funcionalidad. Todos estos procesos InterMod, junto con el uso del modelo de requisitos específico, el *modelo SE-HCI*, permite que las disciplinas MA, DCU y DDM se puedan integrar de forma óptima.

El proceso de desarrollo de InterMod comprende varios pasos. Al inicio, es necesario analizar el proyecto "en su totalidad" (Paso 0) para determinar las decisiones globales de diseño y los UOs iniciales. Después, un conjunto de iteraciones completa el desarrollo de la aplicación a través de los UOs. Cada iteración tiene tres pasos: 1. Revisar la Lista de UOs, 2. Planificar la Iteración Paralela y 3. Realizar las Actividades planificadas. El Paso 1 considera las nuevas necesidades de usuario y desarrollo e incluye los UOs propuestos en iteraciones anteriores. El Paso 2 establece los UOs y actividades a realizar y su adjudicación paralela a los equipos, y el Paso 3 crea y evalúa los modelos, con la nueva perspectiva DCU. Durante las evaluaciones, pueden aparecer nuevos UOs, como nuevas necesidades del usuario o derivadas por el equipo gestor, que se considerarán inmediatamente al comienzo de la siguiente iteración. Igualmente, la evaluación permite sopesar si un modelo se valida o debe ser revisado. La asistencia de los usuarios finales es necesaria para comprobar que el producto se ajusta a sus necesidades, por lo que intervienen en muchos puntos del proceso. Los desarrolladores comprueban la corrección del proceso y, en aras de la agilidad, pueden efectuar muchas evaluaciones internas.

El proceso está regulado por un conjunto de reglas de planificación de actividades y de gestión de modelos que facilitan la toma de decisión para establecer las actividades a realizar siguiendo la nueva perspectiva ágil de DCU en el desarrollo de los UOs. Además, incluye

diagramas en diferentes niveles del desarrollo que favorecen la toma de decisiones y muestran el progreso del proyecto.

Como demostración de la integración de la Ingeniería de la Usabilidad con InterMod, se ha descrito, de forma práctica y detallada, el desarrollo de la aplicación para móvil FindMyPlace. En concreto, se ha indicado qué técnicas de usabilidad utilizar, cuándo y cómo integrarlas, indicando las ventajas y beneficios de esta integración.

6.2 Aportaciones de la metodología InterMod

La propuesta descrita presenta aportaciones interesantes en Ingeniería del Software relacionadas con los tres puntos que se integran en ella: las metodologías ágiles, el Desarrollo Dirigido por Modelos y el Diseño Centrado en el Usuario. En comparación con las metodologías clásicas, esta metodología ha demostrado que la integración MA, DDM y DCU produce estas ventajas:

- La flexibilidad en la planificación y frente a cambios inesperados en la vida del proyecto, que se obtiene por medio de "los deseos del usuario" (UOs), permite realizar un proceso ágil y robusto. La lista de UOs se renueva al comienzo de cada iteración, incluyendo los nuevos UOs a desarrollar. Por otra parte, un UO permanece vivo en la lista de UOs hasta el final del proyecto por lo que se puede seleccionar en cualquier momento para efectuar un cambio en él. Un conjunto de reglas de planificación ayuda a tomar las decisiones de desarrollo de las actividades en cada iteración.

- Los tres tipos de modelos (M-1, M-2 y M-3) que describen la recogida de los requisitos de usuario, la interfaz de usuario y la funcionalidad, permiten realizar un desarrollo de calidad. La creación de estos modelos se aborda según una perspectiva ágil y nueva del DCU, lo que consigue dotar de formalidad y calidad al proceso. Esto es, los requisitos de cada UO deben validarse antes de confeccionar y validar la interfaz, y sólo entonces se podrá realizar y validar la lógica de negocio. Sin embargo, se pueden realizar en paralelo diferentes modelos de distintos UOs. Un conjunto de reglas de gestión actualiza el estado de los modelos durante el proceso.

- El modelo específico SE-HCI, propuesto para recoger y validar los requisitos, es intuitivo y sencillo, y describe de forma enriquecida la interacción usuario-sistema que produce cambios en la interfaz de usuario. Además, permite realizar una evaluación temprana de los requisitos, así como adelantar parte de la presentación y funcionalidad.
- La especificación concreta de las técnicas de evaluación de la IngUS que son más adecuadas en cada momento del proceso ágil y cómo llevarlas a cabo, supone un avance en la integración del DCU con las MA y acerca el resultado a las necesidades reales de los usuarios finales.
- Finalmente, la gestión del proyecto mediante herramientas como la Hoja de Gestión, el Diagrama de Creación de UOs, los Diagramas de Actividades Planificadas, Realizadas y Finalizadas, permiten documentar el progreso del proyecto paso a paso. De esta manera, se podrán visualizar gráficamente los aspectos relevantes del proceso para realizar el seguimiento y la planificación adecuada del proyecto en cada iteración.

6.2.1 Objetivos de Usuario

El proceso InterMod está enfocado a la realización de desarrollos que satisfagan las expectivas o deseos de los usuarios, expresados mediante los UOs. A lo largo del proceso, se verifica su cumplimiento mediante la aplicación de técnicas de evaluación de la usabilidad. Es posible que el resultado de estas evaluaciones implique revisar cualquier punto del proceso, lo que redundará en un mejor ajuste a las necesidades del usuario. Esta concepción de los UOs, y su desarrollo evaluado a lo largo del proceso con técnicas de usabilidad, promueve la adaptación del resultado a las expectivas de los usuarios.

Vistos desde el punto de vista ágil, los Objetivos de Usuario son la medida de avance del proyecto. Las aceptaciones de las evaluaciones continuas de los modelos permiten completar los UOs, esto es, los desarrollos parciales. Cada UO es una unidad lógica en el desarrollo que tiene su origen y fin en el usuario final, aunque a lo largo del proceso ágil pueden producirse fusiones, divisiones y ampliaciones. Esto ha dado lugar a varias clasificaciones de UOs en función de su origen (Directo, Indirecto), autonomía (identidad completa, identidad

incompleta) y reutilización (reutilizado, no reutilizado). Esta clasificación permite ajustar la técnica de evaluación a aplicar sobre cada UO.

El ***UO Directo*** es un objetivo a desarrollar que surge directamente del usuario final y exige, por lo tanto, una evaluación con usuarios finales. Un ***UO Indirecto*** es un UO que surge por necesidades internas del desarrollo y su evaluación puede ser llevada a cabo de forma interna, por expertos, lo que agiliza el proceso. Estas necesidades pueden provocar una división o una fusión de UOs. En el caso de división darían lugar a varios ***UOs Indirectos división*** y en el caso de fusión se obtendría un ***UO Indirecto fusión***.

Un ***UO Incremento*** es un UO de identidad incompleta y se formaliza tras una petición de incremento de funcionalidad, que se efectuará sobre un UO de identidad completa. Un ***UO Incrementado*** es un UO obtenido mediante la ampliación de un UO de identidad completa con uno o más UOs Incremento y puede ser Directo o Indirecto, si supone un incremento sobre un UO Directo o Indirecto, respectivamente.

Un **UO Reutilizado** es un UO creado y evaluado, total o parcialmente, en otro proyecto o en el actual, que puede ser reutilizado. Los UOs Reutilizados pueden ser Directos o Indirectos.

6.2.2 Modelos de desarrollo de UOs: M-1, M-2, M-3

InterMod propone tres tipos de modelos a desarrollar para cada UO del proyecto. Los modelos son: M-1.*Modelo de Requisitos*, M-2.*Modelo de Presentación* y M-3.*Modelo de Funcionalidad*.

La perspectiva DCU propuesta en InterMod, exige crear y evaluar los modelos que configuran un UO en un orden concreto. Esto es, para un UOi se debe cumplir que:

creadoEvaluado(M-1(i)) < ***creadoEvaluado***(M-2(i)) < ***creadoEvaluado***(M-3(i))
donde "<" quiere decir "precede "o "se crea antes que"

Sin embargo, a diferencia de la opinión mayoritaria en DCU, InterMod no requiere que el diseño y la evaluación de la usabilidad de la interfaz de usuario se realicen por adelantado

para todo el proyecto, antes de la implementación de la lógica de negocio. A pesar de esta relajación de restricciones y gracias a la nueva perspectiva DCU aportada, se asegura la usabilidad del producto.

Para facilitar la planificación de las actividades a realizar en cada iteración se han establecido diferentes estados para los tres modelos de un UOi, cuyo transcurso habitual viene determinado por la secuencia: {*pendiente*, *posible*, *enDesarrollo*, *creadoEvaluado*}. Además, eventualmente, un modelo puede pasar a estado *enRevisión*.

Para facilitar la planificación de las actividades a realizar en cada iteración se han establecido diferentes estados para los tres modelos, M-1, M-2, M-3, de un UO:

- ***pendiente*** - Se crea un modelo (vacío) con ese estado en el mismo momento de la formalización de un UO, o se actualiza a ese estado por necesidades de revisión total del modelo.
- ***posible*** - El modelo pasa a estado *posible* cuando las características de desarrollo del problema aseguren la coherencia adecuada para su creación, según la perspectiva IngUS establecida.
- ***enDesarrollo*** – El modelo está *enDesarrollo* cuando se ha planificado que algún equipo de desarrollo realice la actividad para su creación.
- ***creadoEvaluado*** – Un modelo creado se considera *creadoEvaluado* cuando pasa positivamente todas las fases de evaluación.
- ***enRevisión*** – El modelo está *enRevisión* cuando está pendiente de revisión debido a la incorporación de nuevas necesidades.

Las reglas de gestión de los modelos permiten actualizar los estados de los mismos tras cada paso del proceso.

6.2.3 Modelo de Requisitos (SE-HCI)

InterMod permite la recopilación y validación de los requisitos de forma incremental. De esta forma, las aplicaciones de semántica compleja o con factores desconocidos pueden realizarse de forma ágil y validarse poco a poco.

InterMod propone el ***modelo SE-HCI*** como modelo de requisitos enriquecido, cuyo objetivo es adelantar y validar con los implicados del proyecto en etapas tempranas: los requisitos, aspectos parciales de presentación y el comportamiento en la interfaz de la aplicación interactiva. Este modelo es el núcleo de la metodología propuesta y está influenciado, como el resto de los modelos de desarrollo, por los modelos del Usuario y Sistema. Para cada UO, el equipo de diseñadores formaliza su *Modelo SE-HCI*, que comprende el *Modelo de Tareas de Usuario*, el *Modelo del Prototipo*, el *Modelo de Comunicación* y el *Modelo del Comportamiento*. Este modelo, sencillo e intuitivo, permite la creación de prototipos que facilitan la participación conjunta de los usuarios finales, diseñadores y desarrolladores en los procesos de evaluación, según lo recomendado por el DCU y las MA. Gracias a la evaluación temprana de este modelo, los usuarios comprueban y validan sus requisitos, los diseñadores tienen una visión global de la interfaz, y los desarrolladores observan el efecto de las funcionalidades en la interfaz. Esto es interesante para no efectuar trabajo innecesario y no solicitado, o mal interpretado.

WebDiagram es una herramienta interactiva diseñada por mí para ayudar a los analistas y diseñadores a confeccionar y evaluar el modelo SE-HCI de las aplicaciones en desarrollo, de una manera sencilla.

6.2.4 Reglas de planificación de actividades

InterMod describe las reglas de planificación del proceso ágil, de tal forma que en cada iteración se puedan planificar las actividades de desarrollo e integración que realizarán los equipos.

La posibilidad de distribuir el trabajo en paralelo incrementa la velocidad de la realización del proyecto, aunque el propio proceso requiere puntos de integración para asegurar su consistencia. Esto es, InterMod incluye actividades y validaciones de integración desde la

perspectiva de los requisitos, la interfaz y la lógica de negocio, para conseguir la coherencia de los resultados. Estas reglas consideran los estados de los modelos en el proceso para incluir las DA e IA en la lista de posibles a realizar. Además, tienen en cuenta el tipo de integración de modelos (fusión o incremento) en el caso de la planificación de una IA. Además de las reglas de planificación, y como ayuda a la decisión, InterMod sugiere estrategias de gestión en caso de que sea posible la realización de actividades diferentes para la creación de un mismo modelo.

6.2.5 Especificación de la integración de IngUS en InterMod

Las evaluaciones de usabilidad garantizan una mejor toma de decisiones en los procesos de desarrollo y por ello, ahorran tiempo y esfuerzo, evitando diseños equivocados y su posterior corrección. La integración del DCU en InterMod se plasma indicando cuáles son las técnicas de evaluación adecuadas en cada momento del proceso y describiendo cómo aplicarlas.

Las técnicas de evaluación de la usabilidad pueden aplicarse desde etapas tempranas del proceso (Mayhew, 1999), e InterMod propone comenzar a utilizarlas en el paso 0 -"*Analizar el proyecto en General*" para capturar las necesidades del usuario y las decisiones de diseño generales de la aplicación. Cada UO considera los procesos de evaluación recomendados por el DCU, lo que significa que el diseño y la evaluación de la usabilidad se llevan a cabo durante todo el proceso de desarrollo. Las evaluaciones de Usabilidad son particularmente relevantes para los modelos creados de los UOs Directos, ya que éstos reflejan directamente las necesidades de los usuarios y, por tanto, un grupo de usuarios finales deben, necesariamente, estar involucrados en su evaluación. Sin embargo, para acelerar el proyecto, los modelos de los UOs Indirectos pueden evaluarse únicamente con expertos en usabilidad. Los métodos de test, indagación e inspección pueden combinarse para obtener un mejor resultado en el estudio de la usabilidad de los modelos creados en el Paso 3.i – "*Realizar las Actividades de la Iteración*".

En cuanto a las evaluaciones con usuarios finales, se ha podido confirmar que la realización de estas evaluaciones en cada paso del proceso, como por ejemplo sugiere Mayhew (Mayhew, 1999), queda fuera del ámbito de aplicación en metodologías ágiles ya que el tiempo necesario para definir y preparar las pruebas de usuario va en contra de la filosofía ágil. Por lo

tanto, debe haber un equilibrio entre el desarrollo ágil y el tiempo destinado a las evaluaciones con usuarios finales, para que ambos aspectos convivan en el desarrollo del proyecto. Para que el costo de las evaluaciones sea asequible en un contexto de desarrollo ágil, la gestión de actividades puede agrupar las evaluaciones de modelos de diferentes UOs en iteraciones específicas, lográndose mayor eficiencia en su realización. Igualmente, la utilización de las nuevas tecnologías, como cuestionarios on-line o herramientas compartidas ayudan a agilizar este tipo de evaluaciones con usuarios finales.

Además, InterMod recomienda la presencia en el equipo de un experto en usabilidad que permita agilizar la preparación y evaluación de estas pruebas, ya sean multidisciplinares o realizadas únicamente por expertos.

6.2.6 Gestión y documentación del proyecto en InterMod

La crítica más frecuente a las metodologías ágiles es su falta de formalidad. El fundamento del Manifiesto Ágil "*working software over comprehensive documentation"*, minusvalora la documentación sobre el código ejecutable, por lo que una *historia de usuario* típica de los desarrollos ágiles suele ser la documentación que expresa los requisitos. La documentación implica la transferencia de conocimiento entre clientes y desarrolladores (la documentación de los requisitos), y entre desarrolladores nuevos y veteranos (la documentación de diseño y del código). InterMod incluye distintos modelos, como el ***modelo SE-HCI*** en las actividades de "*Análisis de la navegación y diseño*", que requiere formalizar el diseño antes de implementarlo. Sin embargo, si bien es cierto que InterMod propone una especificación concreta para el modelo SE-HCI, su implementación no está ceñida a la utilización de un lenguaje y una herramienta formal. Es posible, por lo tanto, utilizar técnicas de modelado ágil para implementar este modelo.

Por otra parte, es necesario formalizar el proceso para organizarlo y gestionarlo, así como para su revisión posterior. Las reglas de planificación y de gestión de modelos, la Hoja de Gestión, así como los distintos diagramas propuestos son herramientas que ayudan a ello. Esto permite visualizar el estado del proyecto y la distribución de actividades entre los diferentes equipos.

Durante cada iteración del desarrollo del proyecto se registran todos los acontecimientos relevantes en una Hoja de Gestión: UOs creados, Lista de UOs, actividades a realizar, etc. La *Hoja de Gestión del proyecto* ayuda a visualizar el progreso del proyecto y a tomar las decisiones manuales. Una parte de la Hoja de Gestión incluye información sobre el estado del proyecto: los UOs que se van creando con sus características de fusión y división, y el momento de creación de sus modelos. El *Diagrama de creación de UOs* y el *Diagrama de actividades finalizadas* representan gráficamente la información sobre los UOs y sus modelos, respectivamente. En cada iteración, la aplicación automática de las reglas de gestión y el gestor del proyecto, de forma manual, actualizan esta información: en el paso 1 del proceso actualizan la creación de UOs, y en el paso 1 y 3 actualizan la realización de los modelos. La otra parte de la Hoja de Gestión incluye la planificación de las iteraciones: la incorporación de nuevos UOs a la lista de UOs y la asignación de actividades a los equipos. El gestor actualiza esta parte tras los pasos 1 y 2 de cada iteración del proceso, respectivamente. El *Diagrama de actividades planificadas* es la visión gráfica de esta información.

A lo largo del proceso, el *Diagrama de actividades en desarrollo* permite visualizar el proyecto por medio de las actividades que están desarrollándose.

Tanto la Hoja de Gestión del proyecto como los distintos diagramas del proyecto se obtienen de forma automática tras aplicar las reglas de gestión y planificación en cada iteración.

6.3 Líneas Futuras de Trabajo

Varias líneas de investigación se han derivado de esta tesis y continúan su desarrollo en la actualidad. En primer lugar, el trabajo ha probado la validez de la integración del proceso Test Driven Development (TDD) con InterMod. Esta posibilidad se ha trabajado, estudiado y validado con éxito en el desarrollo de la última versión de WebDiagram, aunque quedan por solucionar aspectos de eficiencia en la automatización de pruebas. También se encuentra en desarrollo la reutilización de modelos y UOs, y su utilización como Patrones de Diseño.

Finalmente, los intereses actuales derivados de esta tesis son la confección de herramientas que faciliten la gestión de proyectos y la aplicación de InterMod en el desarrollo de aplicaciones e-learning. Exponemos en este apartado un avance de estos trabajos.

6.3.1 Integración de Test-Driven Development en InterMod

La esencia del Test-Driven Development (TDD) consiste en conseguir "*código límpio que funcione*", y para ello el proceso de programación consiste en "*escribir un test pequeño que no funcione, hacer que lo haga rápidamente y refactorizar para eliminar todas las duplicidades ocasionadas por este proceso"* (Beck, 2002). De esta forma se escribe únicamente el código que se demanda y la calidad del proceso pasa de ser un trabajo reactivo a un trabajo proactivo (Beck, 2002). La integración de InterMod con TDD es adecuada para desarrollar aplicación interactiva, pero es especialmente deseable para descubrir y formalizar las semánticas complejas de algunas ellas.

La integración de TDD en InterMod se facilita por el hecho de que los requisitos se recogen y evalúan incrementalmente, y se formalizan mediante modelos. Esta integración ayuda a formalizar la recogida de requisitos (semántica) paso a paso, a lo largo del proceso, mediante la confección de tests que facilitan el desarrollo temprano de los modelos M-1.

La integración fomenta aún más la comunicación entre los implicados en el proyecto y esto ayuda a cumplir las necesidades de los usuarios finales (Eckstein, 2004). A continuación, se indican los beneficios de TDD que redundarán en beneficios para InterMod y, posteriormente, se muestra cómo es posible integrar esta técnica con los principios de InterMod. Además, el Apéndice C muestra los resultados de la aplicación de InterMod con TDD en el desarrollo de WebDiagram 3.0.

6.3.1.1 Motivación: Test-Driven Development y sus beneficios

Escribir primero los tests de prueba obliga al desarrollador a pensar en el comportamiento del código aún no escrito. "*This thought process clarifies in the developer's mind the behavior and design of the class before it is programmed*" (Larman, 2003). Todos los tests unitarios se ejecutan repetidamente en cada iteración, buscando definir y verificar el dominio. Tal y como dice Beck: "*TDD helps you to pay attention to the right issues at the right time so you can*

make your designs cleaner, you can refine your designs as you learn." (Beck, 2002). Las ventajas directas son la mejora del diseño y la construcción de los tests desde el principio.

Generalmente, los tests en Ingeniería de Software se usan para asegurar la fiabilidad y estabilidad del código. Sin embargo, con la visión TDD, los tests permiten no sólo obtener código fiable a lo largo del tiempo sino también proporcionar piezas excelentes de documentación. Como el diseño se refina constantemente en los procesos iterativos e incrementales, los tests permiten efectuar los cambios y cubrir las necesidades de documentación (Beck, 2002). Además, la disponibilidad de los tests aceleran el progreso del proyecto ya que TDD aborda el proceso de desarrollo a través de los tests.

Son cada vez más los autores que se posicionan a favor de la integración de TDD. Éste es el caso de Eckstein que indica que los tests son la mejor forma de conseguir un software de calidad, ya que aseguran que los cambios constantes en el código no afectan negativamente al sistema (Eckstein, 2004). Además, los tests son recursos excelentes para extender la vida del sistema ya que permiten realizar una integración apropiada de los cambios mediante refactorizaciones. Sin embargo, hay algunas voces discordantes; por ejemplo (Ambler, 2002-13b) indica que "*TDD is very good at detailed specification and validation, but not so good at thinking through bigger issues such as the overall design, how people will use the system, or the UI design (for example). Agile model-driven development is better suited for this*".

6.3.1.2 Modelo de proceso de InterMod con TDD

La propuesta de desarrollo establecida en InterMod, por modelos de Objetivos de Usuario (UOs) y sus evaluaciones, es la base para su integración con TDD. La especificación del modelo SE-HCI para cada UO configura los Modelos de Requisitos propuestos en InterMod, los cuales son el fundamento de los tests unitarios en la técnica TDD. Cada test muestra un comportamiento, característica o respuesta diferente descrita en los modelos que configuran el modelo SE-HCI efectuado para cada UO. Dado que el modelo SE-HCI es el primer modelo que debe crearse y evaluarse para cada UO, la creación de tests a partir de este modelo es directa desde el principio de un desarrollo. Esto es, el modelo SE-HCI de cada UO facilita y enfoca la consecución de sus tests.

Tras la confección de los tests a partir del modelo M-1.Modelo de Requisitos, el proceso de InterMod permite que evaluar de forma temprana los modelos M-2.Modelo de Presentación y M-3.Modelo de Funcionalidad. En este apartado se muestra cómo cambia el enfoque clásico InterMod con esta propuesta de integración con TDD.

El esquema de InterMod combinado con TDD (Figura 6.1) incluye algunos cambios (resaltados en cursiva) en el esquema iterativo, clásico, de InterMod . En el Paso 0-**Analizar el Proyecto Global**, es necesario analizar el proyecto en general para determinar las decisiones globales de diseño y los UOs iniciales. En este paso, la integración del proceso TDD produce la confección de los tests iniciales de la primera iteración.

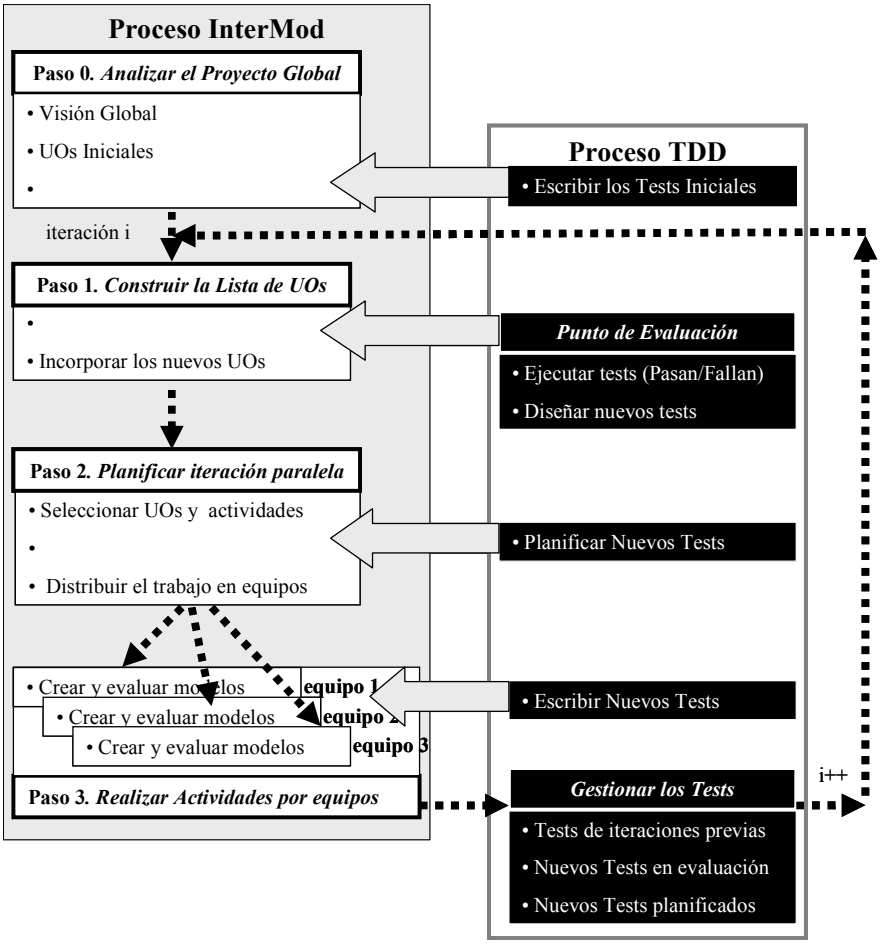


Figura 6.1Esquema de InterMod con TDD

Cada iteración comienza con el Paso 1. que se complementa con el ***Punto de Evaluación*** *conjunta*. En esta reunión, los implicados del proyecto: usuarios, desarrolladores, diseñadores

y expertos en usabilidad realizan las actividades habituales en InterMod y además, las siguientes:

- Observar la ejecución de los tests. Si no funcionan se valora su solución directamente durante la reunión; si el proceso de corrección es largo, se anota el error para corregirlo posteriormente. Los fallos ayudan a valorar el estado actual del código y del proceso en general.
- Debatir nuevos tests que surgen en la evaluación. Si es posible, se definen y escriben en ese momento. Si no, se recogen las ideas en un borrador para su desarrollo posterior.

A continuación, en el Paso 2. ***Planificar la Iteración Paralela,*** se deciden las actividades a realizar sobre los UOs seleccionados en la iteración. Además, en función de los nuevos UOs a desarrollar, se planifican los nuevos tests que deben escribirse.

En el Paso 3 ***Realizar las Actividades de la Iteración,*** los equipos realizan las actividades previstas para los UOs asignados. Las actividades *Análisis y Diseño de la Navegación* e *Integración de modelos M-1* producen los nuevos tests de los UOs planificados. Las actividades incluyen la integración de los modelos correspondientes y la refactorización del código. Este aspecto es primordial porque en los procesos TDD es necesario hacer revisiones constantes del código para eliminar la duplicidad que se ocasiona por la premura en la construcción del código y para mejorar el diseño (Beck, 2002).

Todas las iteraciones incluyen un paso final para ***Gestionar los Tests*** que agrupa y organiza:

- Los tests ejecutados en iteraciones anteriores
- Los tests nuevos escritos en el *Punto de Evaluación* de la iteración actual
- Los tests nuevos escritos en las actividades *Análisis y Diseño de la Navegación* e *Integración de modelos M-1* de la iteración actual.

6.3.2 Utilización de UOs y modelos como Patrones de Diseño

En un proyecto es frecuente que algunos desarrollos hayan sido ya realizados en el mismo o en otro proyecto. Así, un modelo de uno o varios UO(s) se define una vez en un proyecto,

pero puede ser reutilizado en diferentes puntos. Del mismo modo, un modelo desarrollado en proyectos anteriores puede reutilizarse en el proyecto actual, y se puede convertir en un patrón o en una solución a un problema de diseño. Por lo tanto, crear patrones para facilitar y agilizar los procesos de diseño es una posibilidad valiosa a tener en cuenta. La reutilización de UOs, con sus modelos ya creados y evaluados, es una ventaja ágil que asegura además la calidad de un subproducto ya verificado. Sin embargo, el proceso de reutilización del UO o modelo debe hacerse teniendo en cuenta las consideraciones de InterMod en el orden de fusión de modelos para no perder la calidad y consistencia del producto final. Un ejemplo de reutilización incorrecta se explica con un ejemplo en el apartado 6.3.2.1.

La división de UOs puede estar motivada para reutilizar UOs desarrollados previamente. Puede ocurrir que un UO Directo ya haya sido desarrollado en otro proyecto, o bien que para desarrollar un UO se utilicen UOs desarrollados en éste o en otro proyecto. Esto es, los objetivos:

UO3 - *Realizar pago: …*

UO4 - *Ver estado del pedido: …*

UO5 - *Cambios y Devoluciones:…*

{UO3,UO4,UO5} son UOs Indirectos que aparecen muchas veces en los desarrollos de los UOs del mismo o de diferentes proyectos.

En la Figura 6.2 se muestran, en blanco, los objetivos UO9 y UO10 de un proyecto que reutiliza los modelos de UO4 y UO5 de un proyecto anterior.

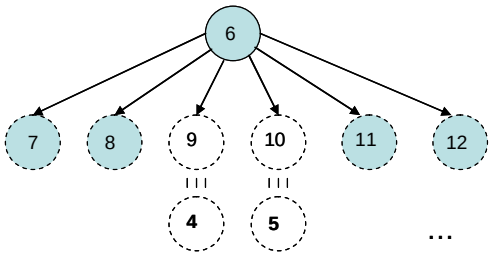


Figura 6.2 Reutilización de UOs

Para reutilizar UOs, debemos contar con sus modelos creados, evaluados y bien documentados. El procedimiento de incorporación a un proyecto del UO Reutilizado será el siguiente:

- El nuevo UO se formalizará en el proyecto con la numeración correspondiente, según su aparición, nombre y descripción.
- Se establecerá su conexión con el UO Reutilizado, indicando su procedencia: el proyecto y versión del que procede y el número de UO que en él tenía. Esta información permitirá sugerir mejoras en los proyectos ya realizados, si continúan vigentes.

A partir de este momento, sus modelos pueden ser modificados, integrados o divididos según se necesite.

La incorporación de UOs reutilizados suele implicar adaptaciones al nuevo contexto, y por lo tanto, será aconsejable que sus modelos pasen a estado *enRevisión*. Por otro lado, si se reutilizan UOs del mismo proyecto pero con cambios, estos UOs se consideran nuevos y se formalizarán indicando además, la numeración del UO original.

6.3.2.1 Ejemplo de reutilización correcta e incorrecta de un UO

El proyecto ficticio *WebButcher*, utilizado en los capítulos 3 y 4, muestra la reutilización de UO10, realizado en otro proyecto (Figura 6.3).

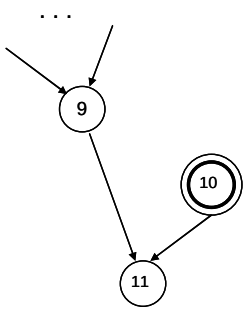


Figura 6.3 Utilización de un UO reutilizado: UO10

UO9- *Recetas y Recomendaciones*: Escribir…

UO10- *Imprimir a mi gusto:* Quiero imprimir...
Proviene de UO5 en el proyecto *Recetas v. 3.0*

UO11- *Escribir, Leer e Imprimir recetas, recomendaciones:* Escribir, leer ….

En este proyecto se ha formalizado con el número correspondiente según su aparición y con su nombre y descripción (UO10-…) indicando, además, el número de UO y proyecto del que proviene (UO5).

Un fallo habitual en el proceso de reutilización de UOs ocurre cuando no se fusionan sus modelos en el orden exigido por la perspectiva DCU que exige InterMod. Por ejemplo, la Figura 6.4 muestra una incorporación errónea del UO10 Reutilizado en el Diagrama de actividades finalizadas del proyecto WebButcher. Esto es, supongamos que en este proyecto se decide reutilizar UO10 y, para ello, se recoge únicamente su modelo M-3(10) que se fusiona con M-3(9). El doble tachado sobre la actividad DA-3(10) en la Figura 6.4 simboliza que esta actividad no se realiza y se recoge el modelo M-3(10), efectuado con anterioridad.

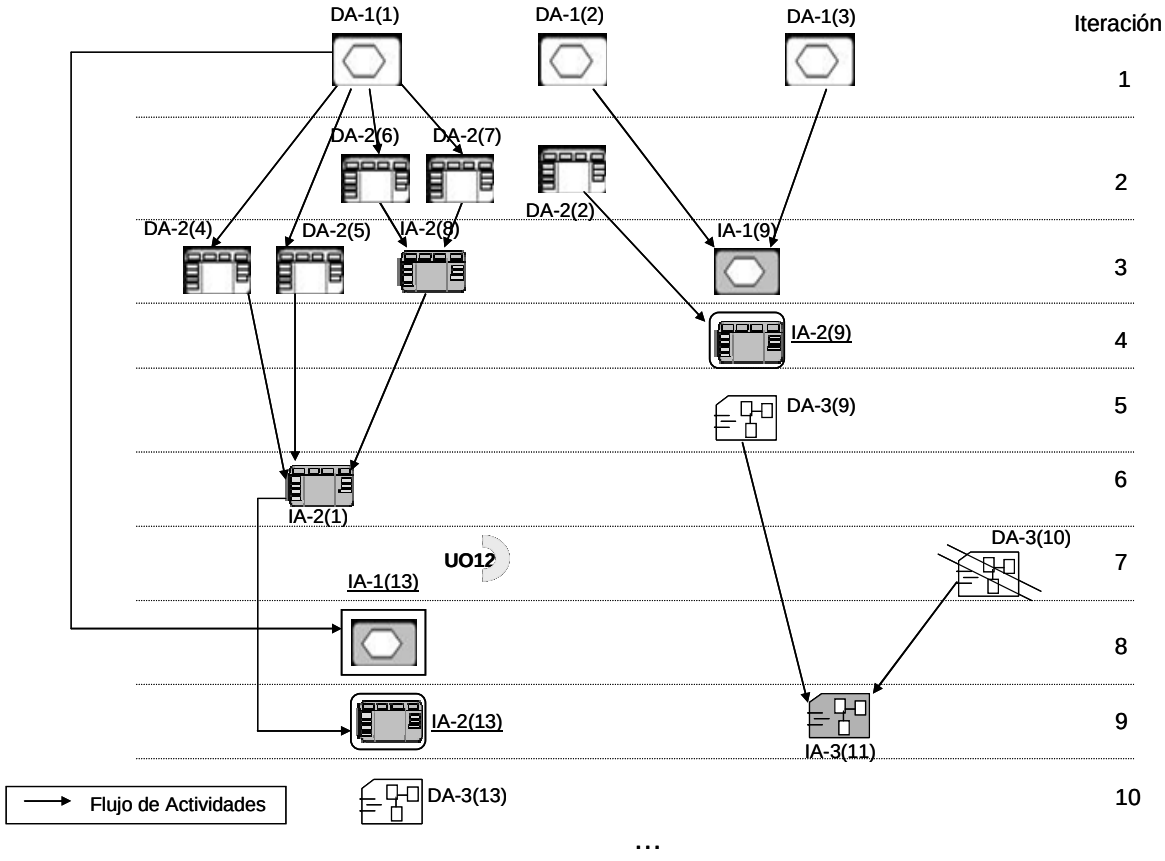


Figura 6.4 Ejemplo de fusión incompleta con un UO reutilizado

Esto puede dar lugar a problemas en la especificación o en la presentación (problemas de usabilidad) debido a que no se ha evaluado la integración de los modelos correspondientes M-1(10) y M-2(10).

La forma correcta de realizar la reutilización hubiera sido la mostrada en la Figura 6.5. Se crea UO11 como Fusión de UO9 y UO10 (UO Reutilizado). Se realizaría la Actividad de Integración IA-1(11) en base a los modelos M-1(9) y M-1(10). A continuación sería necesario

realizar la Actividad de Integración IA-2(11) en base a los modelos M-2(9) y M-2(10). Es entonces cuando se está en condiciones de realizar la integración de los modelos M-3(9) y M-3(10). Esto asegura que las validaciones de los M-1 y M-2 se han realizado correctamente antes de proceder a la integración de la lógica de negocio.

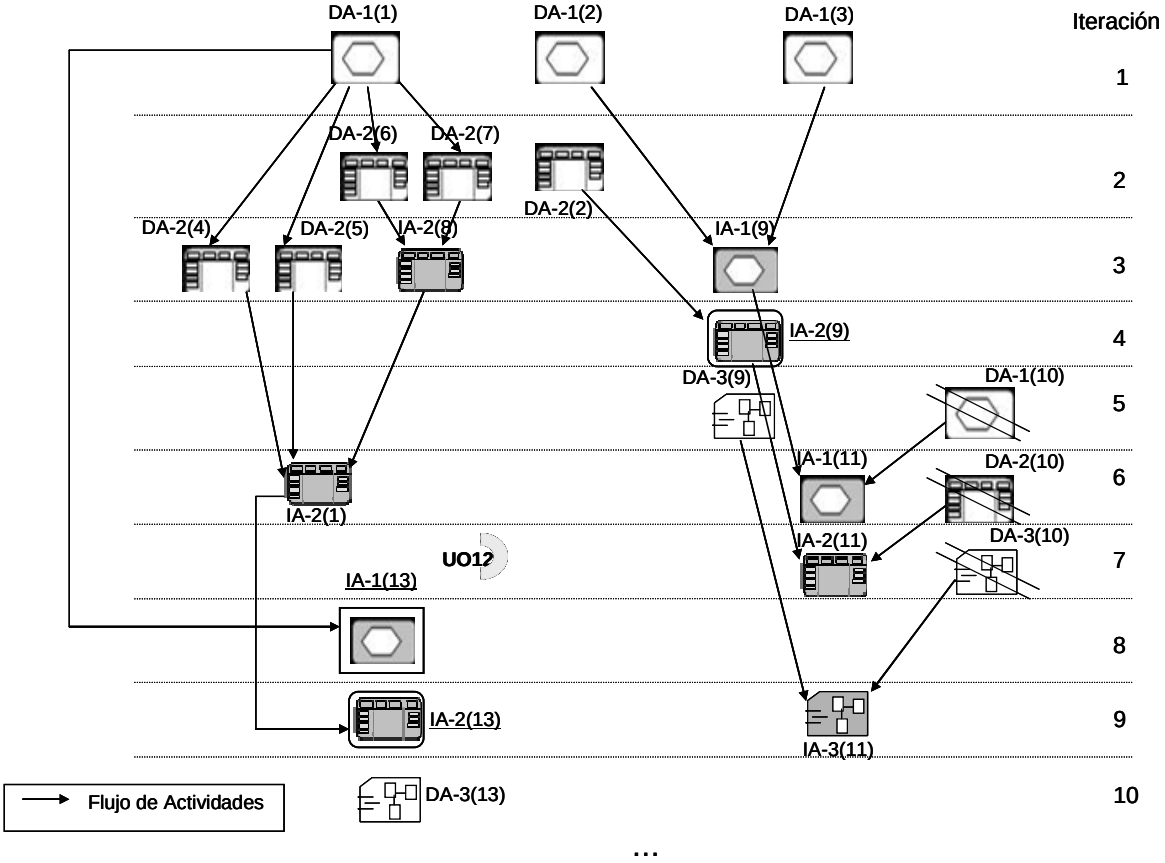


Figura 6.5 Ejemplo correcto de integración de un UO Reutilizado

Aunque la reutilización de componentes agiliza el desarrollo de un proyecto, existe la dificultad de la elección del componente o de que no se ajuste a las necesidades del usuario. Sin embargo, la propuesta de reutilizar UOs o modelos de UOs aplica técnicas de evaluación centradas en el usuario, lo que asegura la viabilidad e idoneidad del componente elegido.

6.3.3 Otros trabajos futuros

Otras líneas de trabajo en curso son la construcción de herramientas software como soporte a la la aplicación de la metodología InterMod, y la utilización de esta metodología en desarrollos de e-learning.

6.3.3.1 Construcción de Herramientas para facilitar la aplicación de InterMod

WebDiagram (apartado 4.3.1) es una aplicación interactiva cuya finalidad es ayudar a los diseñadores de software a describir, visualizar y evaluar los requisitos (modelo SE-HCI) de una aplicación a través de prototipos. Esta aplicación ha evolucionado a través de las diversas versiones para escritorio y como aplicación Web (ver Apéndice A). Además. su desarrollo ha seguido las pautas de InterMod lo que ha permitido probar la validez de la metodología y puntualizar mejoras en ella (Losada et al., 2012).

Otras herramientas en estudio se centran en proporcionar ayuda a la gestión de los proyectos InterMod. Tomando como base las reglas de la planificación de actividades y gestión de modelos, la herramienta generaría la Hoja de Gestión (ver apartado 3.4.5). Se ha realizado una versión preliminar que demuestra la validez y utilidad de la propuesta.

6.3.3.2 InterMod en desarrollos de e-learning

Las aplicaciones de enseñanza on-line utiliza software interactivo para permitir una comunicación fluída entre profesor y alumno, por lo que pensamos que InterMod, como metodología de desarrollo enfocada en Objetivos de Usuario, puede ser utilizada con éxito en ellas. Sus beneficios directos se mostrarán en la clara identificación y realización de las funcionalidades necesarias y deseadas, tanto por profesores como por alumnos, y su evaluación temprana asegurará una buena usabilidad y adaptación. Esta línea de trabajo se está considerando en una tesis en desarrollo actualmente.

6.4 Publicaciones

Publicaciones en revistas

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2013. A guide to agile development of interactive software with a "User Objectives"-driven methodology. Science of Computer Programming Journal Vol. 78, pp. 2268–2281. **Impact factor** JCR: 1.306 (2010-Aceptación), 0.568 (2012). **Journal Ranking:** Q3

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2013. Applying usability engineering in InterMod agile development methodology. A case study in a mobile application. In Journal of Universal Computer Science, Vol. 19, Issue 8, pp. 1046-1065 **Impact factor** JCR: 0.762 (2012-Aceptación, actual). **Journal Ranking:** Q3

Publicaciones en congresos

Losada, B., Urretavizcaya, M., López-Gil, J.-M., Fernández-Castro, I., 2012. Combining InterMod agile methodology with usability engineering in a mobile application development. In: Proceedings of the 13th International Conference on Interacción Persona-Ordenador, INTERACCION'12. ACM, New York, NY, USA, pp. 39:1–39:8. Mereció una **Mención Honorífica**.

Losada, B., Urretavizcaya, M., de Castro, I.F., 2011. An integrated approach to develop interactive software. In: Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part IV, INTERACT'11. Springer-Verlag, Berlin, Heidelberg, pp. 470–474. **Core Rank: A**

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2011. User Objectives as a guide to develop an interactive application. In: Actas Del XII Congreso Internacional de Interacción Persona-Ordenador, INTERACCION'11. Presented at the Interacción 2011, Garceta, Lisboa, pp. 77–86.

Losada, B., Urretavizcaya, M., Fernández de Castro, I., 2011. Agile Development of Interactive Software by means of User Objectives. Presented at the ICSEA 2011, The Sixth International Conference on Software Engineering Advances, pp. 539–545. **Core Rank: C**

Losada, B., Urretavizcaya, M., Fernández de Castro, I., 2010. N_InterMod: Una Propuesta de notación de Diálogo enriquecida para el desarrollo ágil de aplicaciones interactivas. Interacción 2010.

Losada, B., Urretavizcaya, M., Fernández De Castro, I., 2009. Efficient Building of Interactive Applications Guided by Requirements Models. In: Proceedings of the 9th International Conference on Web Engineering, ICWE'9. Springer-Verlag, Berlin, Heidelberg, pp. 481–484. **Core Rank: C**

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2009. Requirements analysis as a guide for the process of organising and developing an interactive application. Presented at the Int. Ass.Development of the Information Society, Barcelona, pp. 412–416.

Capítulo de libro

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2009. The InterMod Methodology: An Interface Engineering Process Linked with Software Engineering Stages. In: New Trends on Human-Computer Interaction: Research, Development, New Tools . Springer, pp. 53–63.

Publicaciones previas a la tesis

Lopistéguy, P., Losada, B., Dagorret, P., 1997. Hypermedia Design Methodologies Versus Hypermedia Functionality Integration. Presented at the ACM Hypertext'97, Southampton**. Core Rank: A**

Losada, B., López, D., Martínez, J., 2007. Guía de actuación en el desarrollo de interfaces de usuario según la metodología centrada en el usuario INTERGRAM. In: CEDI 2007 VII Congreso Internacional de Interacción Persona-Ordenador. Interacción 2007. ***Seleccionado para recopilación en el libro:*** *New Trends on Human-Computer Interaction: Research, Development, New Tools.*

Losada, B., Lopisteguy, P., Dagorret, P., 1997. Étude de la conception d'applications hypermédias. Presented at the INFORSID. Congrès, pp. 133–146.

Losada, B., 2006. INTERGRAM, una herramienta para el desarrollo y estudio de interfaces de usuario. In: Diseño de La Interacción Persona-Ordenador: Tendencias y Desafíos. Interacción 2006, Puertollano.

Losada, B., Martínez, J., López, D., 2007. Intergram, an User-Centered Design Process. In: Assistive Technology Research Series,. Presented at the AAATE 2007, San Sebastián, pp. 786 –790.

Lopistéguy, P., Losada, B., Dagorret, P., 1999. Metodologías de Concepción para Aplicaciones Hipermedia. : Análisis crítico. Informática Educativa Comunicaciones N.1-8 [en **e-revistas**, Editorial CSIC-Edición electrónica]. URL: http://www.erevistas.csic.es/ficha_articulo.php?url=oai:ojs.www.adie.es:article/82&oai_iden=oai_revista337 (accedido 30.7.13).

APÉNDICE A. Evolución de la herramienta WebDiagram

A.1 Introducción

El primer software concebido para ayudar en la realización de aplicaciones interactivas siguiendo la metodología InterMod (Losada et al., 2009b), se denominó T-InterMod y comprendía un conjunto de herramientas. En concreto, se trabajaron utilidades para el diseño y la evaluación de software interactivo, a través de la confección y transformación de modelos. T-InterMod traduce los modelos a una *Descripción Intermedia* (XML) y genera documentación en UIML ("OASIS User Interface Markup Language (UIML) TC," n.d.), que se va actualizando durante el desarrollo del proyecto.

Estas herramientas fueron implementadas con diferentes tecnologías y utilidades, en diferentes proyectos de fin de carrera. Por una parte, esto ha servido para evaluar la utilidad de las diferentes herramientas y por otra parte, para validar (y ajustar) la metodología InterMod durante su confección. A continuación, se expone la evolución de estas herramientas a lo largo de sus versiones y cambios de denominación.

T-InterMod se concibió inicialmente como un conjunto de herramientas autónomas pero interrelacionadas mediante los ficheros UIML que generan. Estas herramientas fueron en sus primeras versiones: *Diagram 1.0*, *Evalgram 1.0* y *Logram 1.0*, y posteriormente *Diagram 2.0* y *Diagram 3.0*, y se confeccionaron en lenguaje Java para escritorio. Estas decisiones de carácter general se tomaron cuando se realizó la primera versión, y fueron heredadas por las versiones siguientes.

A partir de *Diagram 3.0*, que incluye facilidades para la construcción del Modelo de Diálogo y su evaluación, los esfuerzos se enfocaron en desarrollar una herramienta con un solo módulo, denominada *WebDiagram,* centrada en facilitar la realización y validación del Modelo SE-HCI. La nueva herramienta se desarrolló en Java para la web. Se describe a continuación cada una de estas herramientas.

A.2 Primera versión

La primera versión de T-InterMod se diseñó como una herramienta múltiple, compuesta por tres utilidades relacionadas: *Diagram* para ayudar a la confección del diseño y, *Evalgram* y *Logram* para realizar las evaluaciones de presentación y funcionalidad.

A.2.1 Esquema de integración de los módulos de T-InterMod

La Figura A.1 muestra la relación entre los módulos de la herramienta T-InterMod. *Diagram 1.0* ayuda a representar de un modo sencillo, el modelo de Tareas de Usuario para la captura de requisitos de usuario y lo exporta al lenguaje UIML. *Evalgram 1.0* recoge la exportación que realiza *Diagram 1.0*, y lo completa con elementos gráficos concretos para efectuar un prototipo "low-fidelity". De esta manera, los implicados en el proyecto podrán efectuar evaluaciones sobre este prototipo. *Logram 1.0* es un analizador de logs que toma como entrada el fichero de descripción UIML, generado por *Diagram 1.0* y actualizado por *Evalgram 1.0*, y el fichero de registros de accesos del prototipo software. Gracias a esta información, *Logram 1.0* proporciona información temprana acerca del uso realizado por los usuarios finales, útil para el proceso de mejora del mismo.

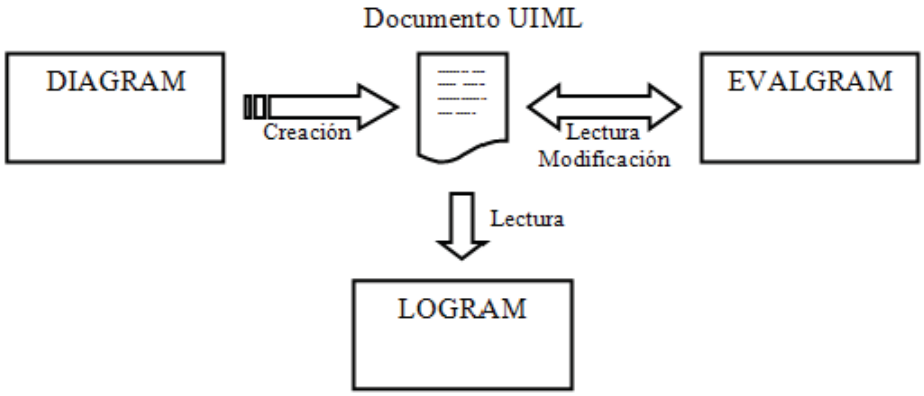


Figura A.1 Conexión entre los módulos de T-InterMod

De forma más detallada, en la Figura A.2 se muestra el esquema propuesto inicialmente. En este esquema se observan las etapas de actuación manual del diseñador (cajas cuadradas) en el desarrollo de cada UO y las etapas que pueden ser automatizadas por la herramienta a partir de la información recogida previamente (cajas redondeadas). Por ejemplo, se plantea que *Diagram* ayude al diseñador en la realización de los Modelos de Usuario, Sistema, Tareas de Usuario y Diálogo. Además, el módulo *Diagram* realizará automáticamente el control de accesibilidad y usabilidad sobre los Modelos de Tareas y Diálogo construídos. Este planteamiento indica un orden en la realización de los modelos que, con mejoras, se mantuvo en las siguientes versiones. En el esquema propuesto para *Diagram*, el orden es: 1. Modelo de Usuario-Modelo del Sistema, 2. Modelo de Tareas de Usuario, 3. Modelo del Diálogo.

Este esquema se mantuvo en esta primera versión, sin embargo *Diagram v.1* únicamente permite el diseño del Modelo de Tareas de usuario y su traducción a UIML. Algunas funcionalidades de la herramienta, que aparecen en la visión de la Figura A.1, no se llevaron a cabo finalmente. Este es el caso del control automático de estándares de usabilidad y accesibilidad.

El Modelo de Diálogo evolucionó en las versiones posteriores hacia el Modelo SE-HCI actual.

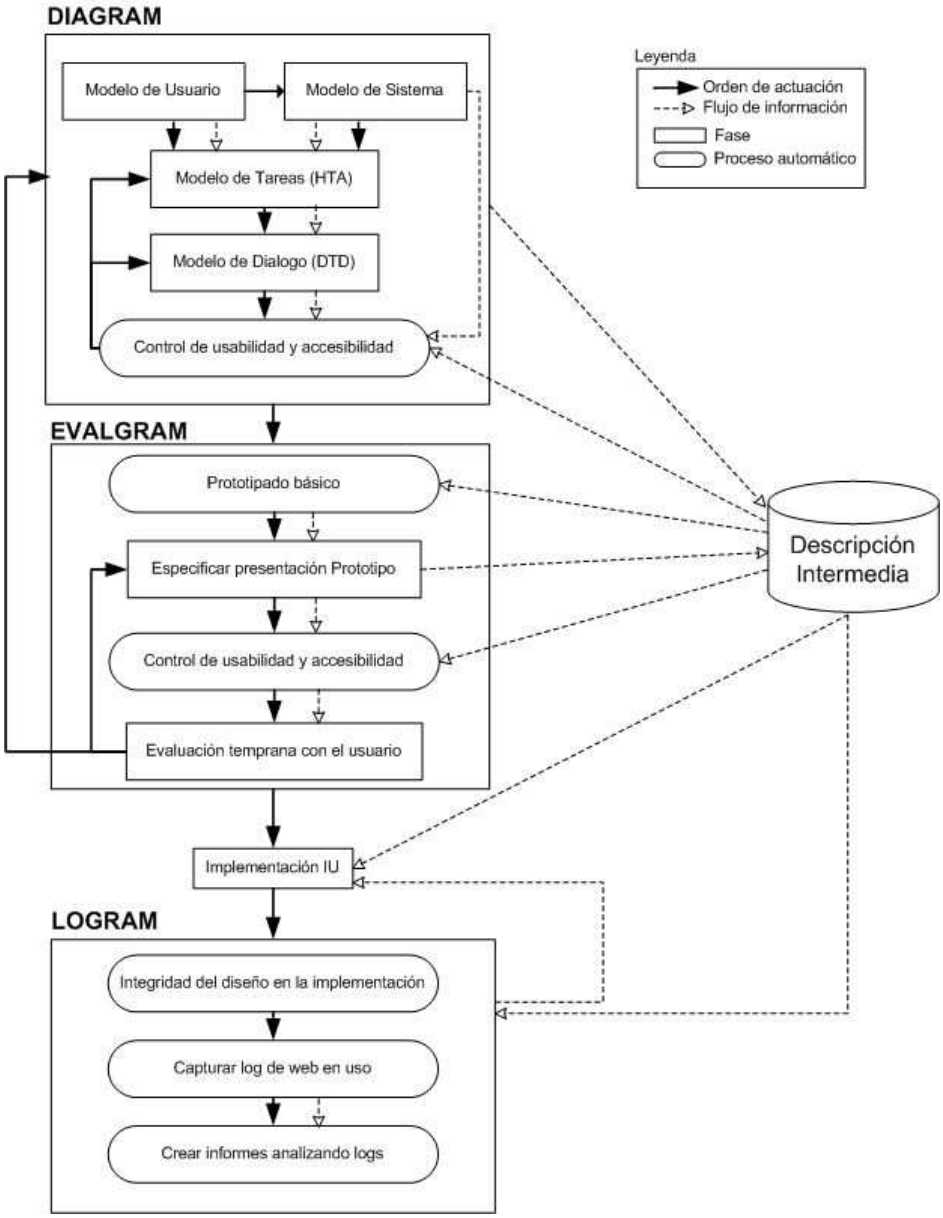


Figura A.2 Esquema inicial de T-InterMod

A.2.2 Diagram 1.0

La primera versión de Diagram tiene por objetivo facilitar la creación, análisis y evaluación del Modelo de Tareas de Usuario. La existencia de este modelo, que se describe con una notación formal, facilita el diálogo y la cooperación entre los implicados en el desarrollo de la aplicación. *Diagram 1.0* se utiliza, también, para la evaluación del modelo con el usuario. Esto es, esta herramienta se usará en diferentes momentos del desarrollo para conseguir que el

Modelo de Tareas de Usuario (cómo cree el usuario que sus tareas se van a suceder en el Sistema) se ajuste al Modelo de Tareas de Usuario del producto final.

A.2.2.1 Decisiones generales

Las primeras decisiones en el desarrollo de la herramienta fueron determinar los Modelos de Usuario y Sistema. En la Tabla A.1 se muestran las conclusiones sobre el Modelo de Usuario, extraídas de la documentación consultada y basada en un estudio de mercado de carácter público realizada por la empresa Ainda a ochenta y dos profesionales de España, Portugal y Latinoamérica (Manchón, 2002). Se indican los aspectos del perfil "Diseñador Web" y de qué manera afectaron al diseño de Diagram.

Tabla A.1 Estudio del Modelo de Usuario para Diagram 1.0

	Perfil	**Conclusión sobre el diseño de Diagram**
Edad	Joven de unos 28 años	Interfaz gráfica dinámica y atractiva para el usuario
Sexo	Mayoría hombres	-
Nivel de estudios	Estudios superiores y de grado medio	Capacidad cognitiva normal, lo que permite crear relaciones de cierta complejidad
Puesto laboral	Repartidos entre las nuevas profesiones propias del diseño de interfaces, y otras no tan directas	Hacer un diseño de la herramienta que no excluya a los no expertos en el diseño de interfaces. Tampoco usar nomenclatura propia de Ingeniería del Software.
Puesto laboral anterior	La mayoría ha trabajado como webmaster o programador, y por un periodo de uno o dos años	Familiarizado con los nombres técnicos de los elementos que forman una interfaz de usuario, por lo que podrán utilizarse y ser comprendidos
Tipo de empresa	Consultoría en su mayoría, seguido empresas y con minoría de freelances	Aspecto serio

En cuanto al Modelo del Sistema, esta primera versión se realizó en Java para poder ser ejecutada sobre diferentes Sistemas Operativos, y para escritorio por ser la forma habitual de trabajo cuando se diseñó.

A continuación, se realizó un estudio para determinar el lenguaje de descripción de la IU. Se analizaron varios lenguajes de descripción de IU, como TERESA XML (Berti et al., 2004b), XIML ("XIML.org," n.d.), UsIXML ("UsiXML - USer Interface eXtended Markup Language," n.d.), UIML ("OASIS User Interface Markup Language (UIML) TC," n.d.), XUL ("XUL," n.d.), y otros que no han continuado su desarrollo como AAIML y AUIML .

Los lenguajes XIML, TERESA XML y UsIXML recogían explícitamente el Modelo de Tareas de Usuario en su estructura y por lo tanto se tomó la decisión de que *Diagram 1.0* diera soporte a los tres. Sin embargo, a la hora de la decisión se tuvo en cuenta que el lenguaje escogido incluyera los elementos necesarios en todos los módulos de la herramienta, esto es, los elementos para recoger el análisis de tareas, su presentación y comportamiento. UIML contenía estas características y destacaba por su estructura sencilla y manuales, por lo que fue el lenguaje escogido.

Después, se determinaron los primeros UOs a desarrollar. Tras entrevistas con potenciales usuarios de la herramienta (diseñadores de aplicaciones interactivas) se llegó a la definición de los siguientes UOs:

- UO1: Quiero representar la descomposición de tareas de usuario mediante diagramas.
- UO2: Quiero imprimir un informe de las tareas que sirva para mostrar y evaluar con los usuarios implicados.
- UO3: Quiero simular la ejecución de las tareas de usuario.
- UO4: Quiero exportar la semántica del diagrama a diferentes lenguajes XML para la descripción de interfaces de usuario, con el fin de ser reutilizada en otras herramientas.

Por otra parte, se incluyeron otros requisitos no funcionales en estos UOs, relacionados con la forma de trabajar de los usuarios o sus perfiles. Por ejemplo, los usuarios finales indicaron su deseo de trabajar sobre proyectos nuevos o existentes. Además, siendo común la existencia de diseñadores de IU que no poseen una formación en este campo, y estando este hecho corroborado en el estudio del perfil de usuario realizado, se optó por favorecer un uso no experto en *Diagram 1.0*. Después de realizar un estudio sobre las

diferentes notaciones posibles, textuales y gráficas, para diseñar el Modelo de Tareas de Usuario (ver apartado 4.3.1.1), se decidió que la notación gráfica HTA (Annett y Duncan, 1967) era la que más se ajustaba a los requisitos del usuario.

Antes de proceder a la implementación de cada UO, se realizo un prototipo de papel con los UOs iniciales propuestos, que fue evaluada mediante evaluación heurística y la técnica del recorrido cognitivo, y dio lugar al aspecto físico y funcional que se presenta en el siguiente capítulo.

A.2.2.2 Descripción de Diagram 1.0

La interfaz gráfica de la aplicación está dividida en tres zonas (Figura A.3): 1. Menús de archivo, edición, y botones de creación, borrado y movimiento de tareas. 2. Zona de trabajo y visualización del diagrama. 3. Zona de configuración de las características de las tareas.

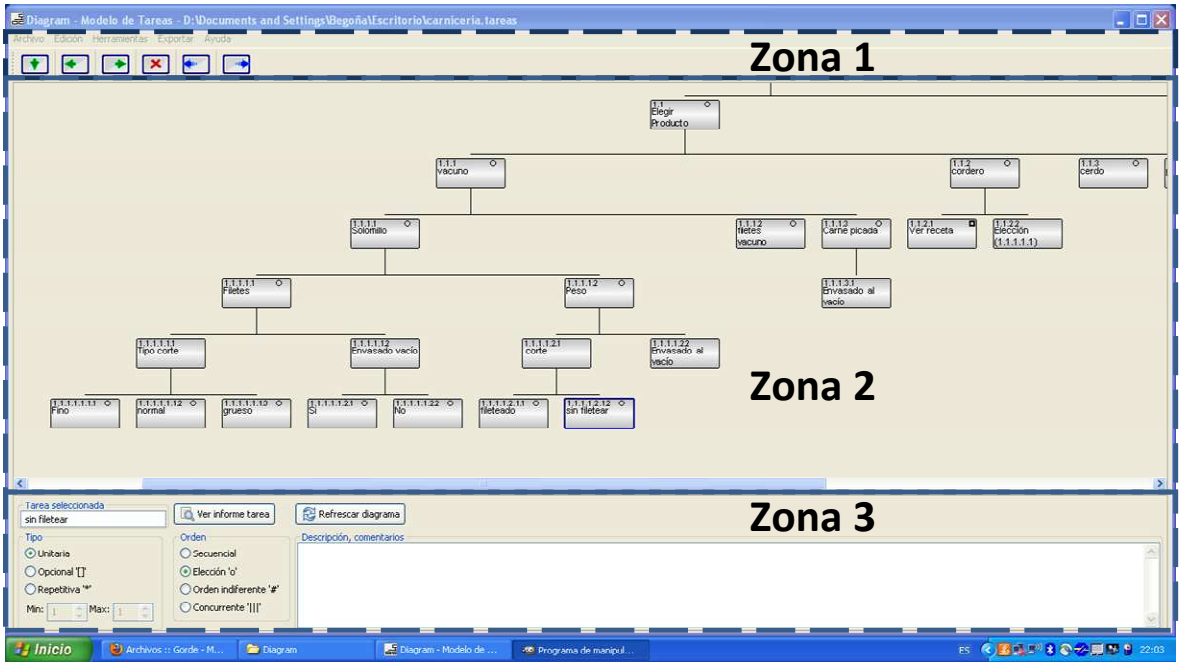


Figura A.3 Interfaz gráfica principal de Diagram 1.0

La Zona 1 permite acceder mediante un menú desplegable o mediante iconos con accesos directos, a las funcionalidades de *Diagram v.1*. Además, incluye las opciones de archivo y edición que se muestran en las tablas A.2 y A.3.

Tabla A.2 Opciones de Archivo en Diagram 1.0

Opciones de Archivo		**Efecto**
	Nuevo	Comenzar un nuevo diagrama
	Abrir	Abrir un diagrama existente
	Guardar	Guardar el diagrama en el que se está trabajando
	Guardar como	Guardar el diagrama en el que se está trabajando, especificando el nombre del archivo
	Imprimir tareas	Imprimir un informe completo de las tareas
	Salir	Salir de Diagram

Tabla A.3. Opciones de Edición en Diagram 1.0

Opciones de Edición		**Efecto**
	Nueva Tarea	Crea, para la Tarea seleccionada, una nueva subtarea. Si existían ya subtareas se colocará en la última posición.
	Nueva Tarea a la izquierda	Crea una nueva tarea inmediatamente a la izquierda de la tarea seleccionada.
	Nueva Tarea a la derecha	Crea una nueva tarea inmediatamente a la derecha de la tarea seleccionada.
	Borrar Tarea	Elimina la tarea seleccionada y sus subtareas en caso de tenerlas.
	Mover a la izquierda	La tarea seleccionada intercambia su posición con la tarea que se encuentra inmediatamente a su izquierda.
	Mover a la derecha	La tarea seleccionada intercambia su posición con la tarea que se encuentra inmediatamente a su derecha.
	Propiedades diagrama	La ventana emergente permite modificar los parámetros de separación horizontal y vertical de las Tareas en el diágrama según las preferencias del usuario.

La Zona 2 muestra la representación gráfica en forma de árbol, del modelo en curso de realización. Finalmente, la Zona 3 contiene unos botones tipo radio para indicar las

características de cada tarea. En concreto, hay dos grupos de características: Tipo y Orden . Además, un campo texto permite al diseñador escribir un texto asociado con la tarea.

El simulador de tareas (Figura A.4) permitirá al diseñador comprobar que el modelo de Tareas de Usuario, mostrado en la Zona 2, se ajusta a las conclusiones del análisis de tareas, de esta manera se pueden encontrar errores durante la elaboración del modelo.

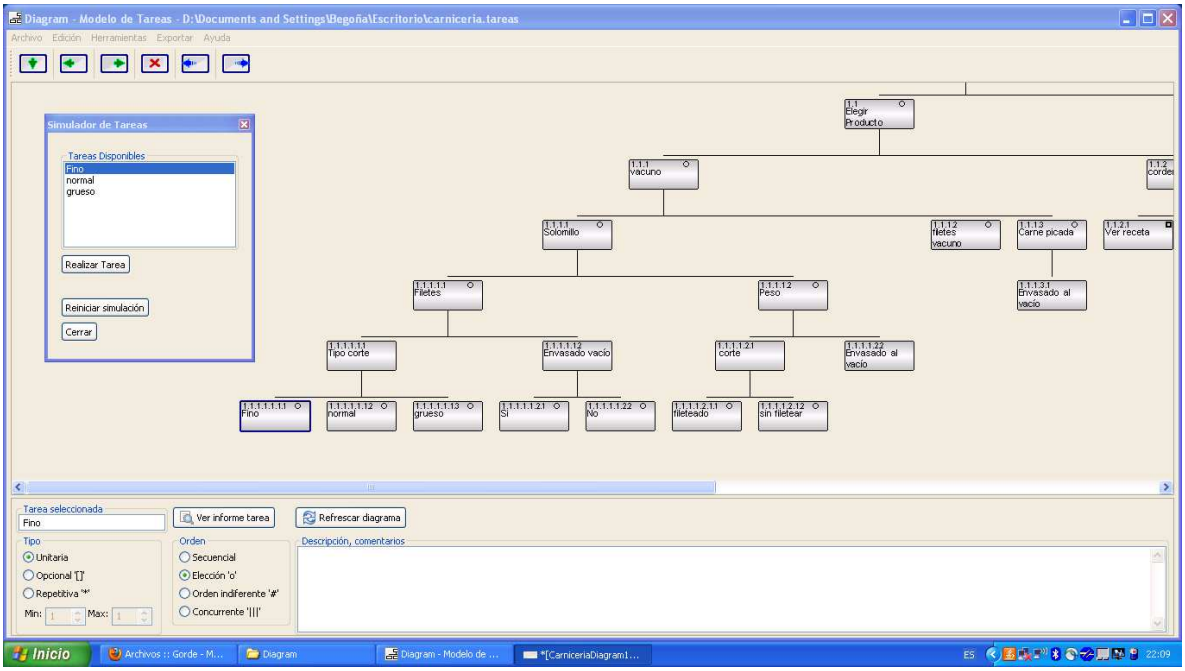


Figura A.4. Simulador de tareas de usuario en Diagram 1.0

Por otra parte, para facilitar la evaluación del modelo de Tareas de Usuario, *Diagram 1.0* es capaz de imprimir un informe de fácil comprensión, para ser presentado y discutido con los usuarios potenciales. El modelo realizado se puede exportar a diferentes lenguajes, como XIML (Puerta and Eisenstein, 2002), Teresa XML (Berti et al., 2004a) y UsIXML (Limbourg et al., 2005), ya que recogen explícitamente el Modelo de Tareas de Usuario en su estructura.

A.2.2.3 Propuestas de mejora en Diagram 1.0 y futuros UOs a desarrollar

Las pruebas con usuarios y el uso posterior de Diagram 1.0 permitió verificar su robustez y sencillez. También se observó que era posible optimizar y aumentar las funcionalidades para facilitar la construcción del Modelo de Tareas y mejorar el visionado del mismo en la

pantalla. La siguiente lista muestra los UOs, sin formalizar, que no se realizaron en *Diagram 1.0*, dejándolas para futuras versiones:

- Zoom para acercar o alejar el árbol del modelo.
- Ocultar/mostrar una rama del árbol del modelo a partir de cualquier tarea.
- Opción de ajustar la vista del modelo al ancho de la pantalla.
- Elegir el color del rectángulo de la tarea para diferenciar alguna si se desea.
- Recorrer el árbol del modelo mediante las flechas del teclado.
- Borrar una tarea mediante la tecla suprimir del teclado.
- Posibilidad de hacer/deshacer acciones de edición.
- Mover tareas o ramas del modelo arrastrándolas con el ratón.
- Identificar tareas idénticas en un mismo modelo.
- Añadir relaciones de tipo "esperar x milisegundos antes de"

A.2.3 Evalgram 1.0

La herramienta *Evalgram 1.0* facilita la evaluación temprana con el usuario, del diseño realizado con *Diagram 1.0*, en la que se diferencian dos etapas:

1. Fase de prototipado, en la que se determina la apariencia o diseño físico de la interfaz de usuario. En esta etapa se recoge el diseño de la navegación propuesto con Diagram y se indican los elementos gráficos concretos que configuran un prototipo avanzado.
2. Fase de evaluación, en la que a través del feedback proporcionado por un usuario potencial, se evalúa el diseño y la presentación de la aplicación sobre el prototipo.

A.2.3.1 Decisiones Generales

Los usuarios pidieron que *Evalgram 1.0* recogiera la descripción intermedia de la interfaz especificada en fases previas del diseño mediante Diagram, para generar el esquema de navegación del prototipo e incorporar a esta descripción el modelo de presentación y la

evaluación temprana de la interfaz. Se quería que Evalgram proporcionara un entorno polivalente que agilizase la realización de las distintas actividades relacionadas con el proceso de evaluación de un prototipo. Estos fueron, por lo tanto, los UOs desarrollados:

- UO1: Quiero realizar el modelo de presentación de la interfaz.
- UO2: Quiero realizar pruebas de evaluación temprana con el usuario final del sistema prototipado.
- UO3: Quiero anotar los problemas detectados durante el proceso para conseguir informes que detallen el estado o comportamiento de la interfaz.

Los requisitos no funcionales y funcionales que completan estos UOs son los siguientes:

- El usuario de la herramienta puede precisar fácilmente y de forma gráfica el modelo de presentación (texto, color, tamaño) de los diferentes elementos del prototipo definidos en la descripción intermedia.
- La herramienta integra un simulador que facilita la evaluación temprana de las diferentes tareas de un prototipo. Esta funcionalidad del sistema interpreta la interacción descrita por la descripción intermedia y simula la navegación existente entre las pantallas que constituyan el prototipo.
- El proceso de evaluación temprana se realiza por un usuario potencial del prototipo bajo la supervisión del diseñador. Durante el proceso, este usuario puede comprobar la navegación de la interfaz y notificar in-situ al diseñador los problemas o dificultades encontradas que obstaculizan su interacción con la interfaz prototipada.
- El diseñador puede especificar en el sistema los problemas detectados durante el proceso de evaluación quedando integrados dentro de la descripción intermedia de la interfaz.
- El usuario de la herramienta puede cargar la descripción intermedia de un prototipo y guardar los cambios o modificaciones realizadas en el modelo de presentación y en la evaluación de dicha interfaz.
- La herramienta dispone de una funcionalidad para la obtención de impresos que informan sobre el estado y comportamiento del prototipo.

Por consistencia con *Diagram 1.0*, se decidió que *Evalgram 1.0* se implementara en lenguaje Java y con el entorno de programación Eclipse. La notación intermedia empleada por los documentos utilizados en Evalgram se basa igualmente, en el formato UIML.

A.2.3.2 Descripción de la herramienta

Evalgram 1.0 emplea técnicas de prototipado como el Storyboard de Navegación, un esquema en el que se muestran tanto las pantallas como las interacciones posibles en un prototipo. Para el proceso de evaluación temprana, la herramienta dispone de un simulador de navegación. Además, el propio simulador se encarga de informar sobre el método de evaluación "Thinking Aloud" al usuario final del prototipo, indicándole las instrucciones que debe seguir durante el test de evaluación de la maqueta digital. Durante la evaluación, es necesario determinar el tipo de problemas detectados en cada una de las pantallas del prototipo. Para ello, Evalgram dispone de un apartado destinado a la anotación de problemas, que incluye los más frecuentes, recogidos de los heurísticos de Nielsen (Nielsen and Mack, 1994), facilitando la labor del evaluador. Finalmente, un informe recoge el estado en el que se encuentra el prototipo evaluado.

En la Figura A.5 se muestran las distintas secciones en la interfaz de *Evalgram 1.0*. El "*Menú superior*" está dividido en cinco opciones bien diferenciadas (Archivo, Prototipado, Evaluación, Imprimir y Ayuda). Cada una de ellas contiene las acciones que pueden llevarse a cabo en la herramienta Evalgram. La opción "Archivo" permite abrir, guardar o cerrar un modelo. La opción "Prototipado" permite especificar un modelo de presentación anotando sus características para cada una de las pantallas del Storyboard de navegación. La opción "Evaluación" permite ver la apariencia de las pantallas y lanzar el simulador, e "Imprimir" permite imprimir el storyboard y el informe general de la evaluación. Finalmente la opción "Ayuda" presenta un manual de uso.

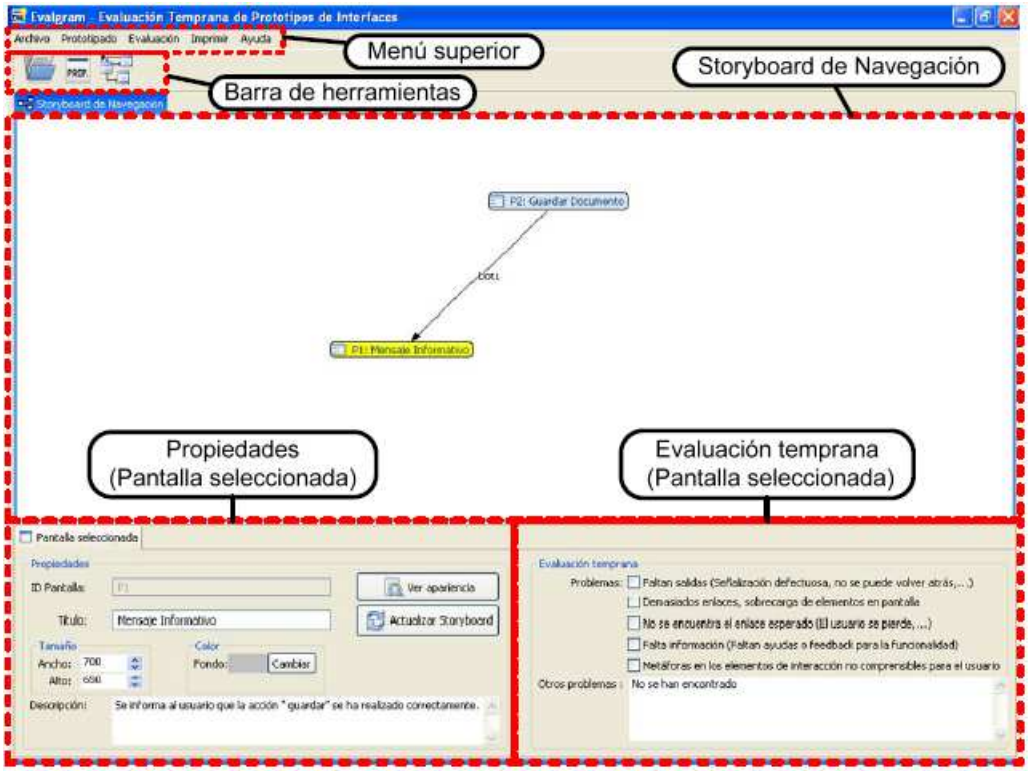


Figura A.5 Interfaz gráfica principal de Evalgram 1.0

En la barra de "*Herramientas*" se dispone un acceso directo mediante iconos a las acciones más habituales dentro de Evalgram. Una vez cargado un documento válido UIML con la descripción de una interfaz de usuario, en el "*Storyboard de Navegación"* se muestra un esquema con las pantallas que componen dicho prototipo y la navegación existente entre ellas. El Storyboard es una parte activa dentro de Evalgram, ya que gran parte de las acciones de *Evalgram 1.0* requieren la selección de una pantalla a través de este panel. Esto es, las pantallas dispuestas en el Storyboard son elementos que pueden seleccionarse y moverse, por lo que para conocer detalles precisos de una pantalla concreta habrá que seleccionar la pantalla deseada, y si la disposición automática de las pantallas en el Storyboard no resulta apropiada, podrá variarse esta colocación con un simple arrastrar y soltar.

En la sección "*Propiedades*" (Figura A.6) se muestran las propiedades de la pantalla del Storyboard de Navegación que haya sido seleccionada. Los cambios que se pueden llevar a cabo sobre una pantalla son variados: se puede modificar el título de una pantalla, su color de fondo, su tamaño e incluso se puede añadir una pequeña descripción del propósito de la pantalla. Además, existen varias acciones en este apartado, como actualizar el Storyboard

cuando se modifica el título de una pantalla y visualizar la apariencia gráfica de la pantalla elegida del Storyboard.

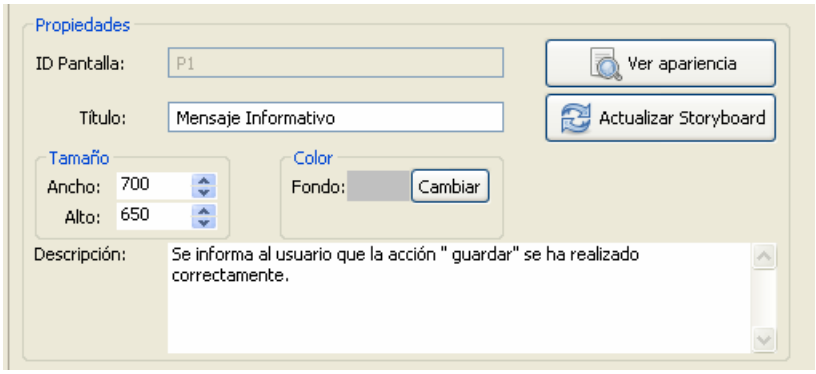


Figura A.6 Sección "Propiedades" en Evalgram 1.0

La sección "*Evaluación temprana*" (Figura A.7), con la pantalla seleccionada, está destinada a la anotación de los problemas detectados en cada una de las pantallas del prototipo, durante su evaluación temprana con un usuario potencial. Este cuadro de anotaciones posee una serie de descripciones de problemas habituales que agilizan el proceso de anotación y, si es preciso, se pueden especificar de forma textual otros problemas que no estén contemplados en las descripciones anteriores.

Como UOs futuros se plantea la evaluación a distancia para lo cual, la herramienta debe estar on-line y no en el escritorio.

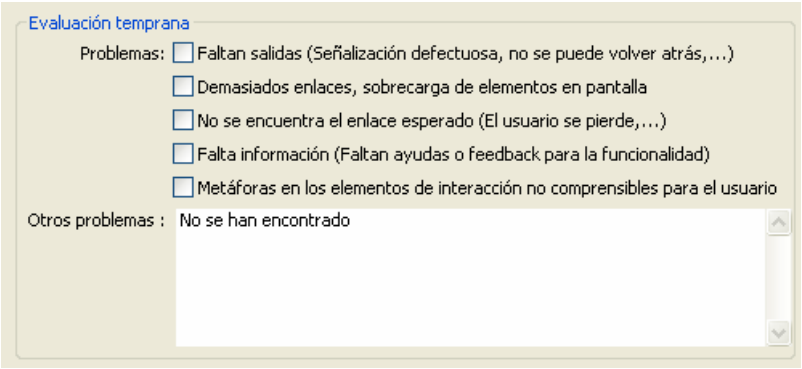


Figura A.7 Sección "Evaluación temprana" en Evalgram 1.0

A.2.4 Logram 1.0

Logram 1.0 es una herramienta para desarrolladores y supervisores de sitios web. Actúa cuando la aplicación ya está en uso. Recoge la información del diseño almacenada en UIML y los logs de uso, realizando una evaluación de la interfaz en uso en comparación con el diseño previsto.

La información que *Logram 1.0* proporciona se obtiene a partir de las acciones realizadas por los usuarios en sus accesos a un determinado sitio Web, y se trata de datos estadísticos tales como máximos, mínimos, medias, desviaciones, etc. Mediante esta herramienta se podrán aplicar criterios de mejora del sitio Web en el momento que se considere oportuno, sin tener que esperar a recibir sugerencias o quejas.

A.2.4.1 Decisiones generales

Logram comparte las decisiones sobre el modelo de Usuario y Sistema realizadas en Diagram 1.0.

Para determinar los primeros UOs, se realizó una encuesta vía e-mail (Figura A.8) a varias empresas de desarrollo de webs, preguntándoles qué les gustaría obtener de esta herramienta.

Tras el análisis y estudio de los resultados, los UOs iniciales escogidos fueron los siguientes:

- UO1: Quiero comprobar las diferencias entre el diseño final y el generado por el diseñador (Diseño original)
- UO2: Quiero obtener información sobre un nodo concreto: número de accesos totales en un rango de tiempo, número máximo de accesos en una sesión, número mínimo de accesos en una sesión, media de accesos por sesión, así como un informe general.
- UO3: Quiero comprobar las rutas de navegación: Quiero saber cuál es el camino más veces realizado, el realizado menos veces, el camino realizado más corto y el más largo. Además, quiero obtener un informe general.

- UO4: Quiero obtener información estadística sobre el idioma: el más utiliado, el menos utilizado, cambio de idioma más frecuente, cambio menos frecuente, porcentaje de sesiones con cambio de idioma, ranking por porcentajes de cambios de idioma, informe general.
- UO5: Quiero obtener información sobre los orígenes de acceso al sitio Web.

Esto forma parte de un estudio sobre una herramienta novedosa para diseñadores Web. ¿Como diseñador Web le interesaría disponer de una herramienta Web que informara sobre el uso de los usuarios finales (a qué acceden y cómo)?, por favor rellene este breve formulario sobre el tipo de información que les resultaría útil conocer una vez este la Web ya en uso:

1. Sobre tiempos de acceso:
   Mucho interés/Interés medio/poco interés
2. Sobre objetivos:
   a. Sobre cuáles han sido las acciones más realizadas.
      Mucho interés/Interés medio/poco interés.
   b. Sobre cuáles han sido las acciones menos realizadas.
      Mucho interés/Interés medio/poco interés.
3. Sobre modos de acceso
   a. ¿Cuál ha sido la desviación sobre los caminos previstos (para ir de un elemento de la Web a otro)?
      Mucho interés/Interés medio/poco interés
4. Sobre el alejamiento del diseño previsto
   a. Comprobación de que el diseño coincide son la implementación final.
      Mucho interés/Interés medio/poco interés
5. Sobre Horarios de acceso:
   a. Información estadística sobre horarios de acceso.
      Mucho interés/Interés medio/poco interés
6. Origen de conexión e idioma elegido:
   a. Información estadística sobre los orígenes de conexión (los más habituales y los menos).
      Mucho interés/Interés medio/poco interés
   b. Idiomas más seleccionados, número de cambios de idioma en una misma conexión.
      Mucho interés/Interés medio/poco interés

Observaciones (añada cualquier tipo de información que a usted le parecería interesante y que no aparece en el cuestionario):

Figura A.8 Encuesta para la obtención de los UOs iniciales de Logram 1.0

A.2.4.2 Descripción de la herramienta

En la Figura A.9 se muestra la interfaz principal de esta herramienta:

Figura A.9. Interfaz gráfica principal de Logram 1.0

La "Información sobre nodo concreto" permite generar información relativa a un nodo a partir de la navegación de los usuarios, la gramática generada durante el diseño y la información adicional introducida por el diseñador, como por ejemplo, las rutas previstas. De esta forma por ejemplo, se podrán identificar nodos inutilizados (que nadie accede) o por el contrario nodos muy utilizados para colocarlos en un sito más accesible, dentro del sitio Web. Concretamente, proporciona información sobre:

- Número total de accesos: Número de accesos realizados sobre el nodo seleccionado en el rango temporal seleccionado.
- Número máximo de accesos en una sesión
- Número mínimo de accesos en una sesión
- Media de accesos por sesión : Devuelve el número medio de accesos realizados por sesión.
- Informe General : Contiene toda la información generada por cada una de las acciones anteriores.

La "*Comprobación de Rutas*" permite obtener resultados estadísticos sobre las rutas seguidas por los usuarios al ir de un nodo a otro (los nodos son seleccionables). Además el diseñador podrá introducir la ruta prevista entre esos dos nodos (no necesariamente la más corta) y obtener comparaciones estadísticas, desviaciones típicas, etc. Esto permitirá al desarrollador reajustar la navegación del sitio Web para cumplir sus fines: publicitarios, de navegación, accesibilidad, etc. Concretamente, proporciona información sobre:

- Camino más veces realizado: Indica la ruta realizada más veces por los usuarios entre dos nodos, además de información adicional que dependerá de la información introducida sobre la ruta prevista.
- Camino menos veces realizado: Indica la ruta realizada menos veces por los usuarios entre dos nodos, además de información adicional que dependerá de la información sobre la ruta prevista.
- Camino más corto realizado: Indica la ruta más corta realizada por los usuarios para ir del nodo inicial al nodo final, además de información adicional que dependerá de la información sobre la ruta prevista.
- Camino más largo realizado: Indica la ruta más larga realizada por los usuarios para ir del nodo inicial al nodo final, además de información adicional que dependerá de la información sobre la ruta prevista.
- Informe general: Incluye un resumen de los anteriores puntos.

La opción "*Información sobre el idioma*" permite obtener información estadística sobre el idioma seleccionado para navegar por la Web. Además, informará sobre posibles cambios de idioma durante la navegación. Esto permitirá al desarrollador por ejemplo, establecer como idioma principal el más utilizado, mejorar el contenido de las páginas con los idiomas más utilizados, etc. Concretamente, proporciona información sobre:

- Idioma más utilizado: Indica el idioma más utilizado por los usuarios en sus accesos, además de información adicional que dependerá del idioma indicado como previsto.
- Idioma menos utilizado: Indica el idioma menos utilizado por los usuarios en sus accesos, además de información adicional que dependerá del idioma indicado como previsto.

- Cambio de idioma más frecuente: Indica el cambio de idioma más realizado por los usuarios, además de información adicional que dependerá del idioma indicado como previsto.
- Cambio de idioma menos frecuente: Indica el cambio de idioma menos realizado por los usuarios, además de información adicional que dependerá del idioma indicado como previsto.
- Porcentaje de sesiones con cambio de idioma: Indica el porcentaje de las sesiones en las que se ha realizado algún cambio de idioma.
- Ranking por porcentajes de cambios de idioma: Devuelve un ranking, ordenado de mayor a menor en base a su porcentaje de realización, de los cambios de idioma realizados.
- Informe general : Reúne el conjunto de toda la información generada por cada una de las acciones anteriores.

En cuanto a la opción "*Información sobre el Origen*" permite, por un lado, generar información sobre los orígenes de acceso al sitio Web y, por otro, compaginar dicha información con los idiomas seleccionados para la navegación. De esta forma, se obtendrán resultados que permitirán al diseñador conocer si su política de idiomas es la correcta, o si interesaría introducir nuevos idiomas debido a su demanda o por el contrario, eliminar alguno por su desuso.

Poder conocer los orígenes más frecuentes sobre un sitio Web permitirá a su desarrollador por ejemplo, alojar dicho sitio Web en un servidor más cercano a esos orígenes para minimizar latencias, hacer mayor publicidad del sistema Web en dichos orígenes, etc. Concretamente, la información que proporciona es:

- Ranking por porcentajes, de mayor a menor, de todos los orígenes desde los que se han realizado accesos al sitio Web.

Se plantean los siguientes UOs a realizar en un futuro:

- Poder extraer el fichero de logs de cualquier servidor a través de Internet.
- Obtener información matemática y estadística de los informes.

- Obtener gráficas en los informes.
- Mostrar la navegación del sitio Web en un árbol ‘manejable’.
- Compaginar la información sobre el idioma con la información sobre el origen

A.3 Evolución de la herramienta

En los desarrollos siguientes, los esfuerzos se enfocan en la mejora de Diagram, especialmente en el simulador de tareas y en conseguir una única herramienta que se va enriqueciendo.

A.3.1 Diagram 2.0

Diagram 2.0 es una evolución de *Diagram 1.0* en la que cambia la forma de simular el modelo de tareas. En este caso, se realiza mediante un prototipo en el que las tareas se representan como botones que se pueden pinchar siempre y cuando en ese momento estén disponibles y se puedan ejecutar.

A.3.1.1 Decisiones generales

Este proyecto hereda los UOs de Diagram 1.0 y añade otro UO directo:

- UO1: Quiero obtener un prototipo software mediante botones en una etapa muy temprana, que permita la evaluación del Modelo de Tareas de Usuario.

Se establece para ello, la primera representación gráfica de la semántica de las diferentes tareas del usuario en este tipo de prototipos. Esto es, la disposición de los botones dentro de las ventanas dependerá del orden y del tipo de las tareas a las que representan.

Además, en las tareas de orden secuencial aparecerá su numeración junto al nombre de la tarea (1.1 Mi tarea), mientras que en las tareas cuyo orden sea indiferente o concurrente, sólo aparecerá el nombre, como muestra la Figura A.10.

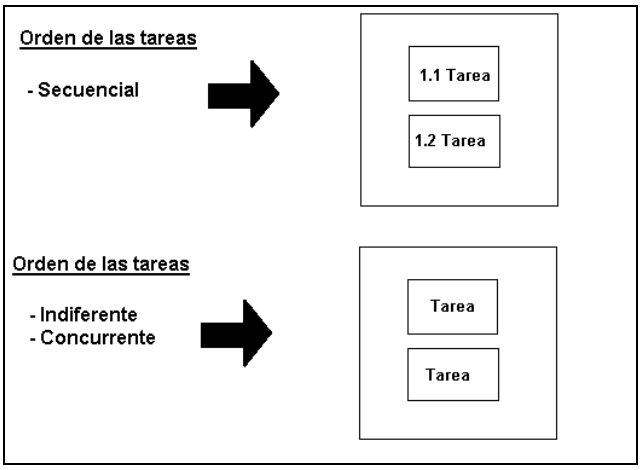


Figura A.10 Nombre de las tareas según su orden en el Prototipo de Diagram 2.0

Inicialmente, en las tareas de orden secuencial, sólo el botón de la primera tarea estará activado, y el resto se irán activando una vez completada la tarea predecesora. En las tareas de orden indiferente o concurrente, en cambio, inicialmente todos los botones aparecerán activos, desactivándose uno a uno una vez efectuada la tarea.

En las tareas de orden "**elección o**", el usuario deberá elegir una tarea entre las presentadas. Su representación será una lista horizontal de botones activos, asemejándose a un menú donde el usuario podrá seleccionar una única opción (Figura A.11).

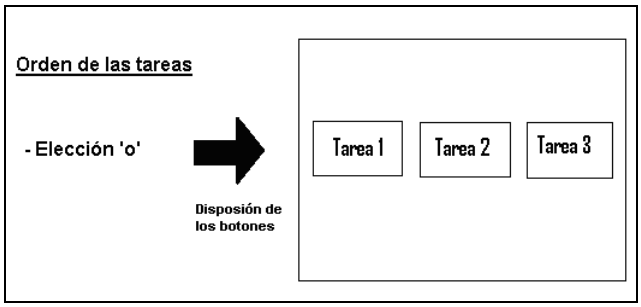


Figura A.11 Disposición de los botones en tareas de orden elección 'o' en el Prototipo de Diagram 2.0

Las tareas podrán ser de tres tipos diferentes: unitarias, opcionales o repetitivas. Con la intención de ayudar al usuario, se añaden distintos símbolos a los botones para identificar su tipo. Así, las tareas opcionales, contendrán el símbolo '?' después de su nombre como se observa en la Figura A.12.

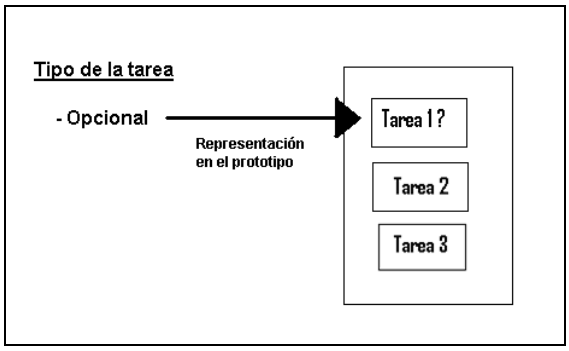


Figura A.12 Representación en el Prototipo de las tareas opcionales en Diagram 2.0

Las tareas de tipo repetitivas, es decir, aquéllas que se pueden realizar de 1 hasta n veces, dispondrán del símbolo '*' al lado del nombre de la tarea (Figura A.13).

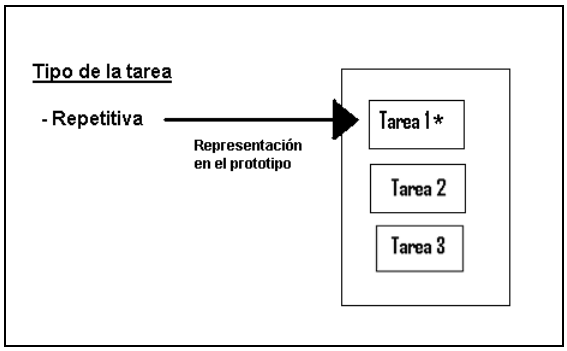


Figura A.13 Representación en el Prototipo de las tareas repetitivas en Diagram 2.0

En el caso de que la tarea sea opcional y repetitiva, es decir, que se pueda realizar de 0 a n veces, el botón que representa dicha tarea contendrá ambos símbolos.

A.3.1.2 Descripción de Diagram 2.0

La interfaz gráfica de *Diagram 2.0* no se diferencia de la versión anterior (Figura A.14). El prototipo de tareas de usuario se presenta en una ventana e incluye una línea de recorrido o "migas de pan" y el conjunto de botones activos (seleccionables) y no activos (no seleccionables).

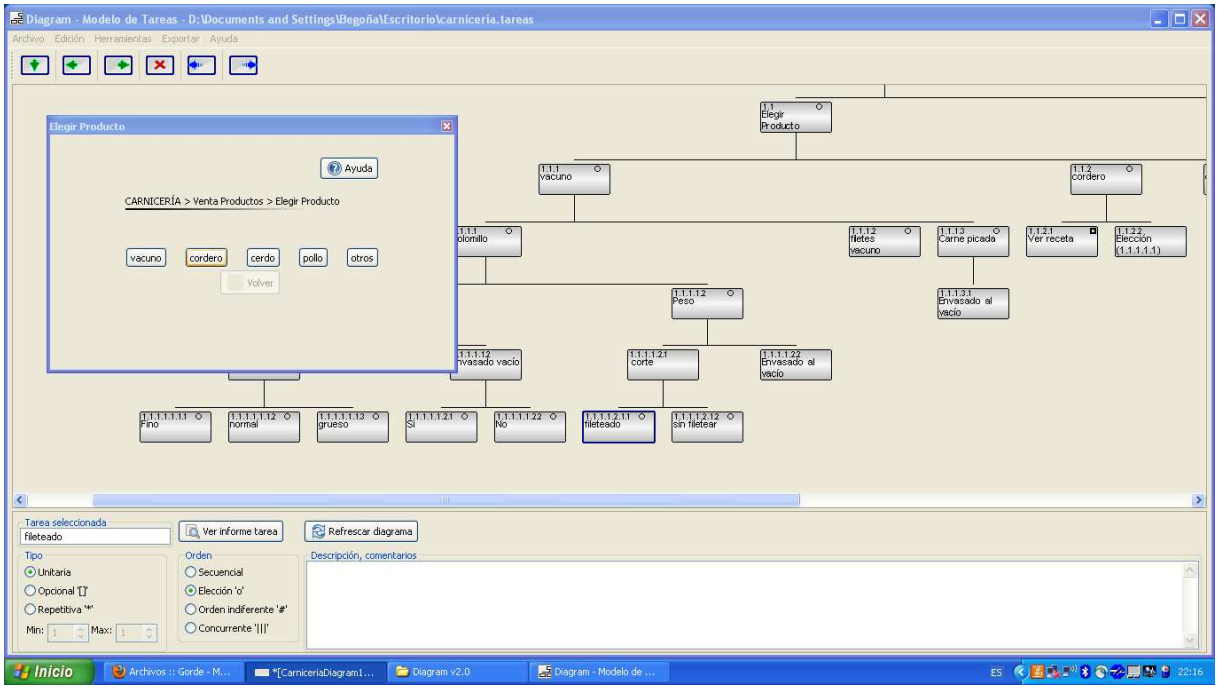


Figura A.14 Interfaz gráfica y Prototipo de Tareas de Usuario en Diagram 2.0

A.3.2 Diagram 3.0

En la versión tercera de Diagram se incluye la posibilidad de representar, guardar y evaluar el *Modelo del Diálogo* como una ampliación del *Modelo de Tareas de Usuario* de las versiones previas. Además se pueden representar parcialmente los Modelos de Prototipo y Comportamiento. Estos nuevos modelos evolucionarían más tarde hacia el *Modelo SE-HCI* descrito en el apartado 4.3.

A.3.2.1 Decisiones generales

En esta versión se amplia la herramienta con los siguientes UOs iniciales:

- UO1: Quiero crear el *Modelo de Diálogo*, como evolución del Modelo de Tareas existente, en el que se incorporarán las acciones del sistema como respuestas al usuario que le informan y pueden provocar un cambio en el flujo de navegación por la interfaz.

- UO2: Quiero crear un *Modelo de Prototipo* inicial que permitirá obtener una interfaz más real a la hora de simular el prototipo. Por ejemplo, se permitirá al diseñador elegir la representación de las tareas en una nueva ventana, en una sección diferente, o desplegable.
- UO3: Quiero crear un *Modelo de Comportamiento* inicial que indique la tarea siguiente de cada tarea.
- UO4: Quiero actualizar el simulador de la aplicación para que el prototipo obtenido se ajuste a las características de los modelos de Diálogo, Prototipo y Comportamiento.

A.3.2.2 Descripción de Diagram 3.0

Diagram 3.0 ayuda a representar el Modelo de Tareas de Usuario, de la misma manera que sus versiones previas. Además, describe algunos aspectos del Modelo de Diálogo, Modelo del Comportamiento y Modelo de Prototipo, y obtiene un prototipo que tiene en consideración las características escogidas de estos modelos.

Modelo de Diálogo

Para visualizar el Modelo de Diálogo se añade la opción de conversión de tarea sistema/usuario. Finalmente, la barra de acceso rápido presenta el aspecto mostrado en la Figura A.15.

Figura A.15 Botón de conversión de tarea Sistema-Usuario en Diagram 3.0

Las tareas del sistema se distinguen de las del usuario por un círculo circunscrito en un rectángulo azul que antecede a la etiqueta (Figura A.16).

Tarea de usuario	Tarea del sistema
	

Figura A.16. Diseño de la tarea del sistema en Diagram 3.0

Modelo de Comportamiento

El *Modelo de Comportamiento* muestra la navegación prevista según el orden y tipo de las tareas, así como la navegación imprevista provocada por saltos, indicados en las tareas hojas, sean éstas del sistema o de usuario, mediante la propiedad Ir a….

La representación utilizada en el diagrama incluye debajo de la tarea el texto "Ir a" seguido del número de la siguiente tarea a ejecutar.

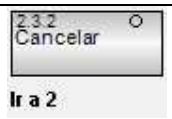
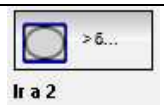
***Ir a…* en tarea de usuario**	***Ir a…* en tarea del sistema**
	

Figura A.17 Representación de Ir a… en una tarea, en Diagram 3.0

En la Figura A.18, se observa un ejemplo gráfico del *Modelo del Diálogo* y *Modelo del Comportamiento* en Diagram 3.0:

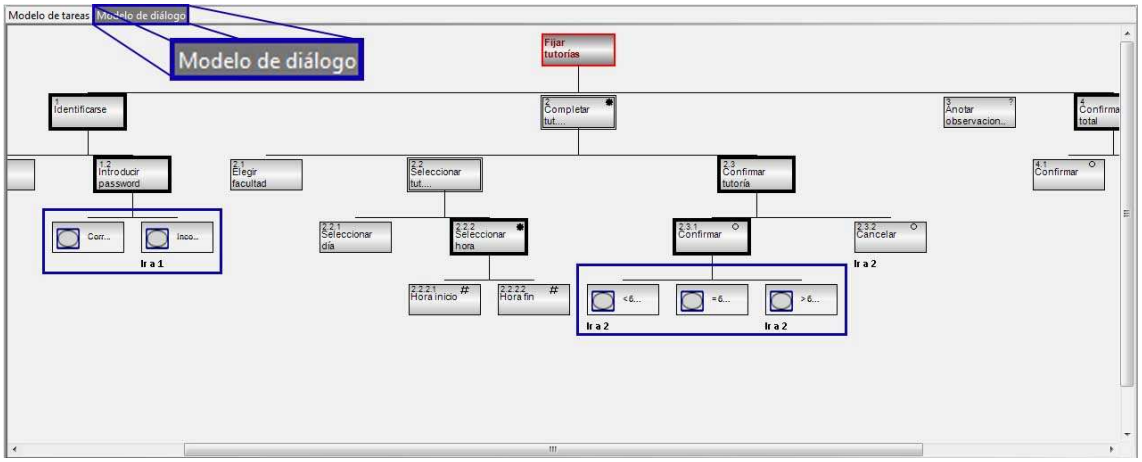


Figura A.18 Ejemplo gráfico de un modelo de Diálogo en Diagram 3.0

Modelo de Prototipo

Por defecto, cualquier tarea nueva diseñada representará sus hijos en una nueva ventana. Para indicar una opción de presentación diferente hay que acceder a la zona inferior de la interfaz principal donde están las opciones de presentación (Figura A.19).

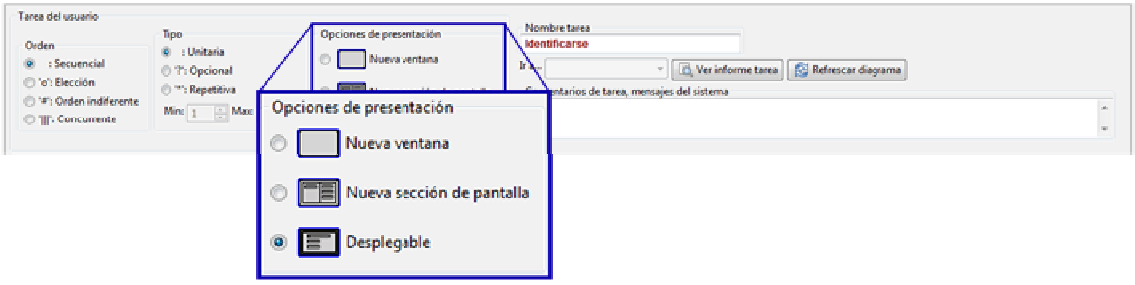


Figura A.19. Opciones de presentación en Diagram 3.0

En la Figura A.20 se puede observar la visualización gráfica de cada una de estas opciones en el diagrama.

Nueva ventana	Nueva sección	Desplegable
		

Figura A.20. Representación gráfica de opciones de presentación de tareas de usuario en Diagram 3.0

Visualización del prototipo

El prototipo se visualiza y se ejecutará en función de los modelos de Diálogo, Comportamiento y Prototipo confeccionados. En cuanto al *Modelo de Prototipo*, si la tarea seleccionada tiene como propiedad "desplegable", sus tareas hijas o subtareas aparecerán justo debajo de ella, como muestra la Figura A.21.

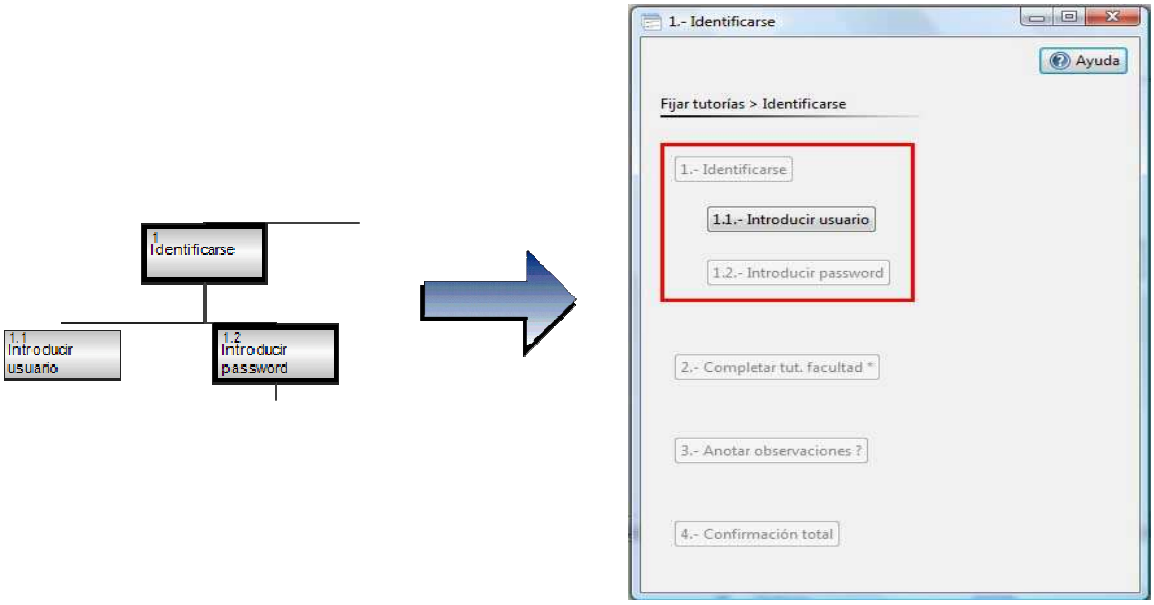


Figura A.21 Representación gráfica de tarea "desplegable" en el Prototipo de Diagram 3.0

En caso de la propiedad "Nueva Sección" de pantalla, las tareas hijas o subtareas de la tarea seleccionada se representarán en una nueva sección de la misma pantalla (Figura A.22).

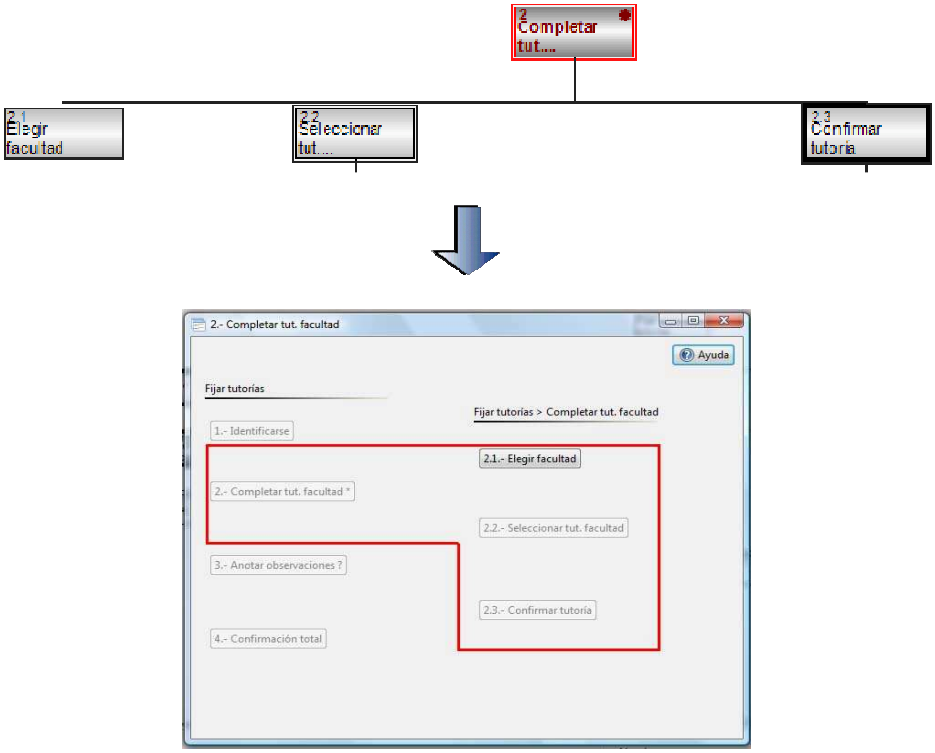


Figura A.22 Representación gráfica de tarea "Nueva Sección" en el Prototipo de Diagram 3.0

Finalmente, en caso de la propiedad "Nueva Ventana", las tareas hijas se mostrarán en una nueva ventana (Figura A.23).

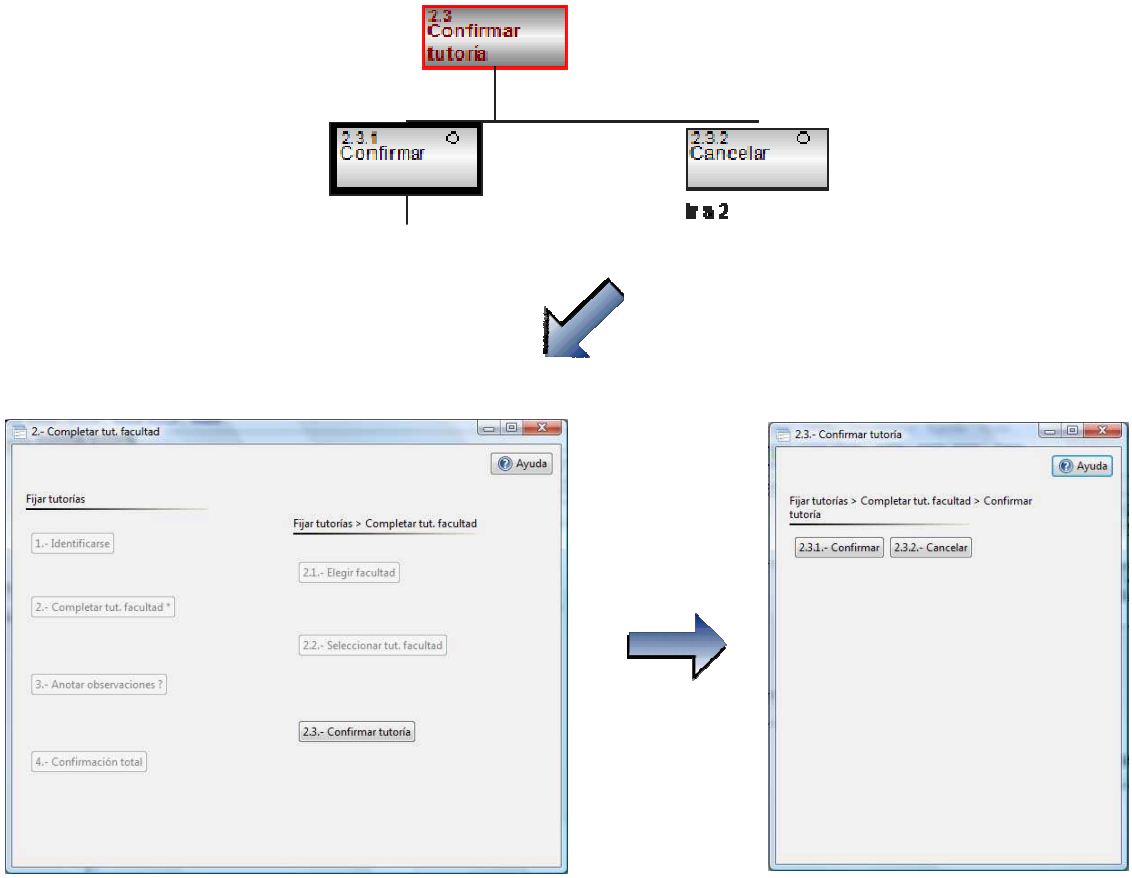


Figura A.23 Representación gráfica de tarea "Nueva Ventana" en el Prototipo de Diagram 3.0

Además, las tareas del Sistema se representan siempre en horizontal una al lado de la otra, ya que funcionarán como el tipo elección, y al pulsarlas aparecerá el comentario o mensaje que contenga dicha tarea. En caso de no contener ningún mensaje y estar ese campo vacío, el mensaje que se mostrará en pantalla será el nombre de la tarea (ver figura A.24).

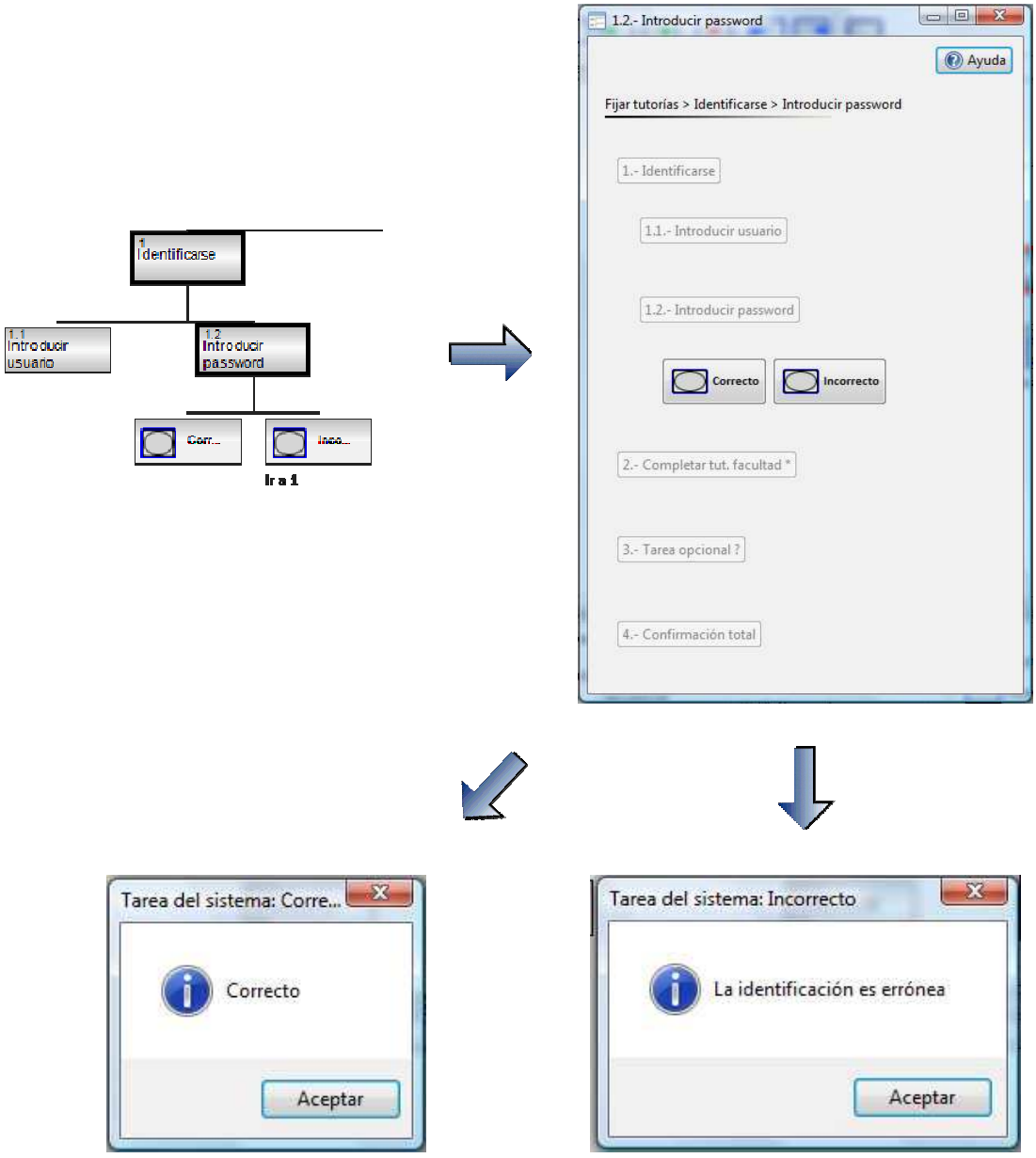


Figura A.24 Representación gráfica de las tareas del Sistema en el Prototipo de Diagram 3.0

A.3.3 Diagram 4.0: Ampliación de Diagram 3.0 con el Modelo de Usuario y Sistema

El proyecto de Diagram 4.0 es una ampliación de Diagram 3.0 que añade los modelos de Usuario y Sistema.

A.3.3.1 Decisiones generales

Los UOs iniciales de Diagram 4.0 serán:

- UO1: Quiero crear el *Modelo de Usuario* con cuatro modelos de usuario (básico, medio, experto y personalizado) . Cada uno con características configurables sobre el tamaño de botón, colores, tamaño de letra, máximo número de secciones, tareas, etc.
- UO2: Quiero crear el *Modelo de Sistema* para que permita indicar aspectos de seguridad que influirán en las respuestas del Sistema, y limitaciones físicas como el número máximo de secciones, de tareas, tamaño de la ventana y del botón, etc, que influirán en la navegación y presentación de los elementos del dominio.
- UO3: Quiero actualizar el simulador de la aplicación para que el prototipo obtenido se ajuste a las características del Modelo de Diálogo, Prototipo y Comportamiento, así como a las limitaciones establecidas por el *Modelo de Usuario* y el *Modelo del Sistema*.

A.3.3.2 Descripción de Diagram 4.0

Habrá cuatro pestañas cada una con su lienzo de trabajo: Modelo de sistema, Modelo de usuario, Modelo de tareas y Modelo de diálogo, como muestra la Figura A.25.

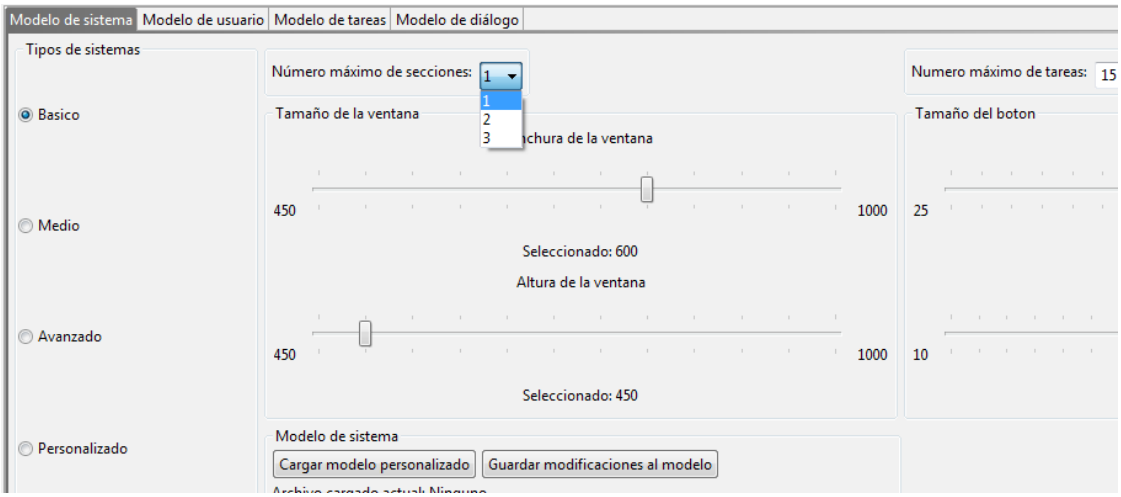


Figura A.25 Representación gráfica de Diagram 4.0

Los lienzos de los modelo de sistema y usuario permiten parametrizar el tipo de sistema y usuario de una manera sencilla, mediante unas configuraciones establecidas, como Básico, Medio, Avanzado, o bien de una forma personalizada.

A.3.4 WebDiagram 1.0

El objetivo de este proyecto es crear una versión web del programa Diagram, para que el usuario pueda trabajar con WebDiagram en su explorador de la misma manera que lo haría con la versión local de Diagram.

A.3.4.1 Decisiones generales

Los Objetivos de Usuario iniciales fueron:

- UO1: Quiero tener una versión reducida Diagram desde 0, con las funcionalidades asociadas al Modelo de Tareas de Usuario, que sea una herramienta web.
- UO2: Quiero equiparar las funcionalidades de esta herramienta web con las de Diagram 3.0.
- UO3: Quiero reparar y mejorar aspectos de la herramienta original.

Además , se tuvieron en cuenta los siguientes requisitos en los UOs anteriores:

- Su interfaz debe ser lo más parecida posible a la de Diagram.

- La herramienta debe ser ligera ya que se ejecutará a través de la web.

A.3.4.2 Descripción de WebDiagram 1.0

El aspecto gráfico de WebDiagram es igual que el de las herramientas Diagram predecesoras, como se observa en la Figura A.26.

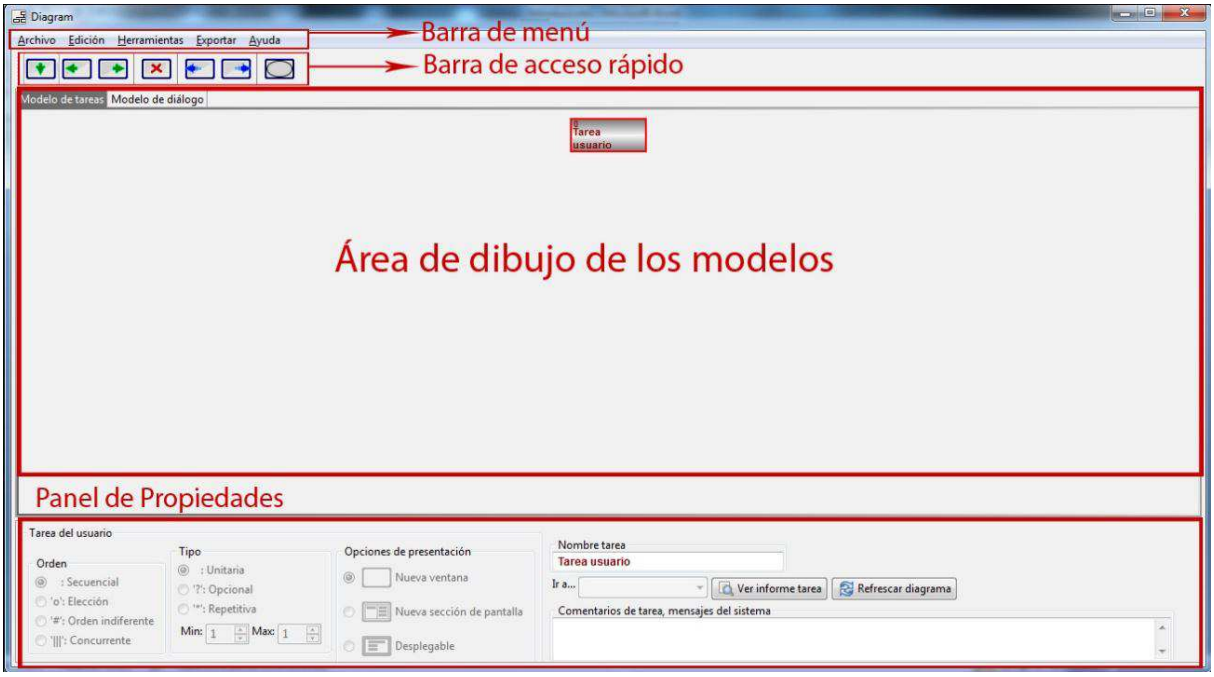


Figura A.26 Representación gráfica de WebDiagram 1.0

A.3.5 WebDiagramCMED 1.0

WebDiagramCMED es una herramienta web para el desarrollo de software interactivo con visualización mediante mapas conceptuales. Esto es, parte de Diagram 3.0 pero utiliza mapas conceptuales para representar gráficamente los modelos y trabajar de manera más flexible con ellos.

A.3.5.1 Decisiones generales

Los UOs iniciales de WebDiagramCMED 1.0 fueron:

- UO1: Quiero flexibilizar la realización de diagramas, de manera que se pueda trabajar con las tareas directamente en la zona de diseño.

- UO2: Quiero tener una aplicación on-line, con las funcionalidades de Diagram 3.0.

A.3.5.2 Descripción de WebDiagramCMED 1.0

WebDiagramCMED ha mantenido las funcionalidades de Diagram 3.0 y en gran medida su diseño, pero su aspecto gráfico es algo diferente. Las pestañas de cambio de Modelo de tareas/Modelo de diálogo, se representan ahora como botones en la parte superior izquierda (Figura A.27).

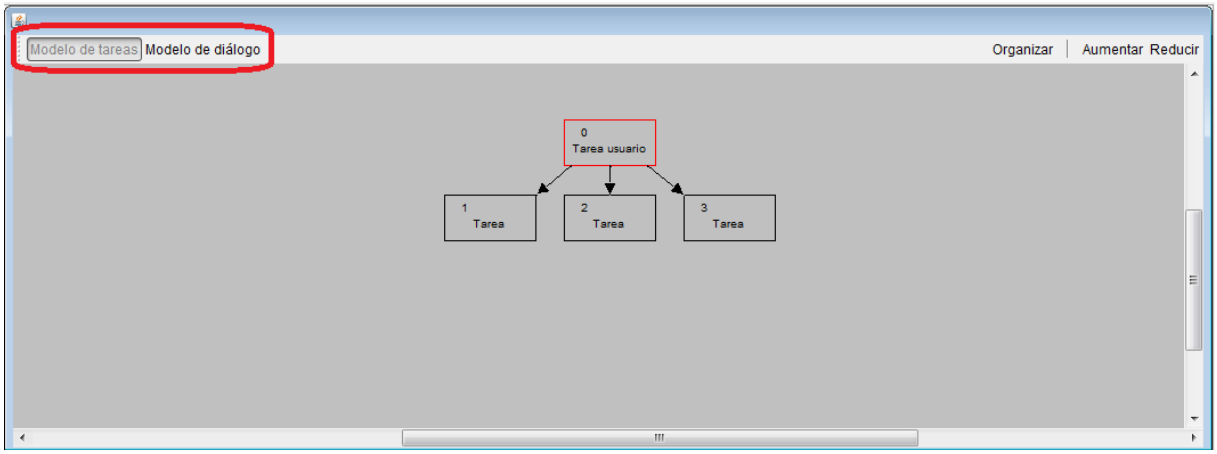


Figura A.27 Representación gráfica de WebDiagramCMED 1.0

Además, un pequeño menú en la parte superior derecha permite organizar el grafo y aumentar o disminuir su tamaño (Figura A.28).

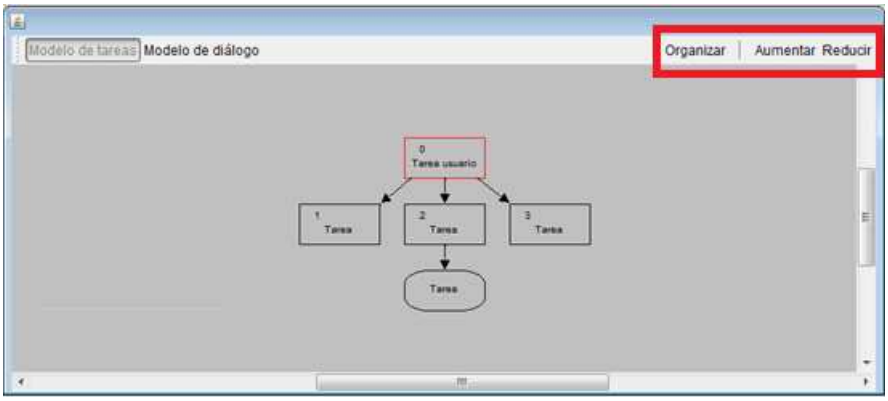


Figura A.28 Tratamiento del grafo en WebDiagramCMED 3.0

Un cambio sustancial en esta herramienta es que las acciones de organización, movimiento, borrado, creación, etc sobre las tareas se hacen directamente en el lienzo de dibujo.

A.3.6 WebDiagram 2.0: Tareas delegadas

Los modelos son habitualmente grandes. Los diseñadores expresan su deseo de trocearlos. La versión 2.0 de WebDiagram ofrece esta posibilidad.

A.3.6.1 Decisiones generales

Los objetivos de usuario iniciales fueron:

- UO1: Quiero tener la posibilidad de descomponer las tareas en subtareas delegadas.
- UO2: Quiero ver el Modelo de Diálogo para cada subtarea delegada de forma independiente.
- UO3: Quiero ver las subtareas delegadas, en forma de pestañas anidadas.

Otros requisitos a incluir en estos UOs fueron:

- Cada tarea delegada aparece en una pestaña nueva. En el árbol se muestra con el icono de tarea delegada. Desde este momento, no podrá tener hijos en el diagrama principal. Las tareas hijas deberán crearse en la pestaña de la tarea delegada.
- Cuando se desglosa una tarea, surge una pestaña cuyo nombre es el de la tarea junto con su numeración.
- Las propiedades de una tarea delegada serán las propiedades de la tarea de usuario de partida.
- El número de tareas delegadas es ilimitado. Sin embargo, en una tarea el número máximo de subtareas delegadas en profundidad es tres.
- Una tarea del sistema no puede delegarse. Se avisará de ello con un mensaje.
- Una tarea delegada mantendrá su numeración según su posición en el diagrama principal.
- Las pestañas se irán colocando a medida que se van delegando las tareas.
- Cada diagrama tendrá su prototipo, teniendo en cuenta sus tareas delegadas aunque estén en otras pestañas.

- Cada desglose tiene su modelo de tareas de usuario y modelo del diálogo.
- Una tarea externa con la etiqueta "ir a" no podrá ir a una tarea interna de una delegada.
- Una tarea marcada como delegada no podrá ser el objetivo de un "ir a" porque es una tarea raíz.
- Una tarea delegada podrá volver a su estado inicial pulsando el mismo botón de "desglose" en la aplicación.

A.3.6.2 Descripción de WebDiagram 2.0

El botón "desglose"  incorporado al menú superior permite indicar que una tarea será delegada y por tanto se visualizará en un diagrama nuevo, al que se accede por una pestaña de nombre el de la tarea delegada correspondiente. En la Figura A.29 por ejemplo, las tareas 1-A, 2-A y 3-A son delegadas y se visualizan en diagramas separados a los que se accede mediante las pestañas 1-A, 2-B y 3-C.

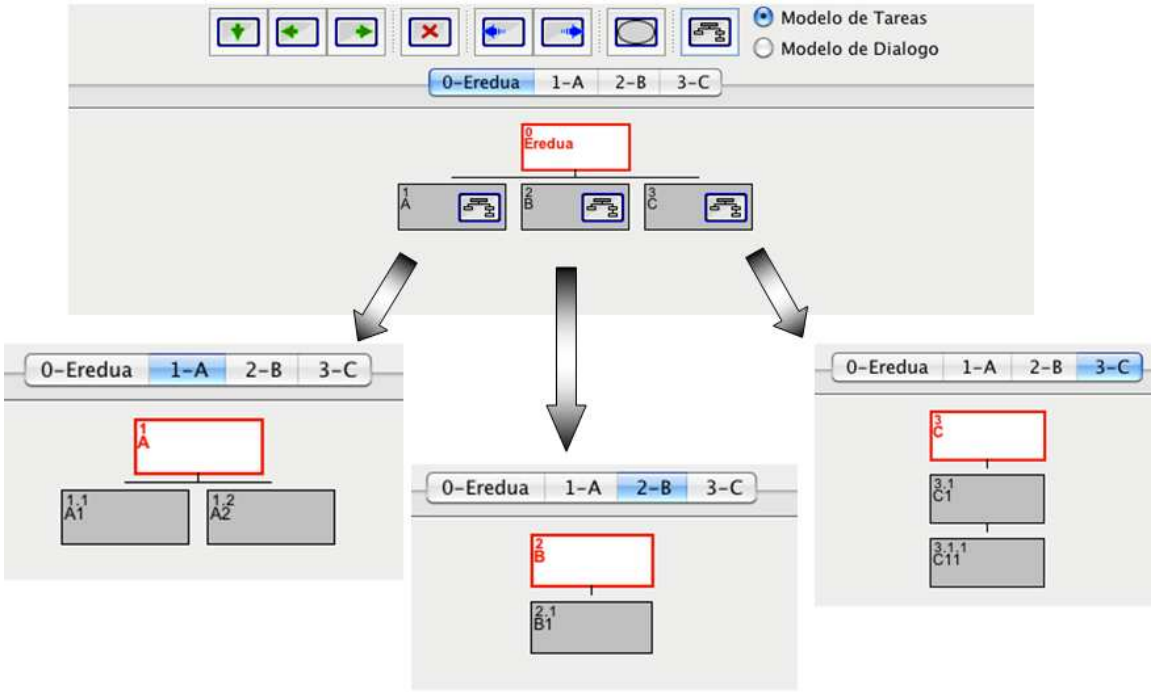


Figura A.29 Representación gráfica de las tareas delegadas en pestañas en WebDiagram 2.0

APÉNDICE B. Lenguaje UIML

B.1 Introducción

UIML ("OASIS User Interface Markup Language (UIML) TC," n.d.) es un lenguaje XML para la descripción de interfaces de usuario. Este lenguaje es capaz de definir los distintos elementos de interacción que se pueden encontrar en una interfaz de usuario. Esto significa que en un documento UIML se puede guardar información referida a menús, botones, listas y demás elementos que conforman una IU y además, especificar su colocación, tamaño o color. También ofrece la posibilidad de almacenar sus acciones, dando la opción de registrar los distintos eventos del prototipo.

En T-InterMod, sus herramientas se conectaban mediante el lenguaje UIML. Sin embargo, a partir de Diagram 3.0 estas herramientas se unificaron y no fue necesario utilizar este lenguaje de conexión, pero se conservó para ampliaciones futuras.

Estos son los elementos principales que se pueden encontrar en un documento UIML:

- <interface> Es el elemento principal en UIML. Contiene la descripción de la interfaz mediante estos cuatro elementos:

  - <structure> Contiene una enumeración del conjunto de partes de la interfaz. Cada parte tiene un nombre de instancia y un nombre de clase.
  - <content> Está compuesto por el texto, sonido e imágenes de la interfaz y puede declararse mediante el programa de aplicación.
  - <behaviour> Es la interacción con el usuario escrita mediante un conjunto de condiciones basadas en reglas y sus acciones relacionadas.
  - <style> Especifica el estilo de presentación, especifico de cada dispositivo para cada clase de las partes de la interfaz.
- <peers> Específica qué elementos en la plataforma destino y qué métodos o funciones en los scripts, programas u objetos en la aplicación están asociados con la interfaz de usuario. Toda la información del dispositivo se encuentra en este elemento.

Hay que aclarar que no todos estos elementos serán necesarios a la hora de exportar los prototipos software propuestos por InterMod y por esto, se escogen los elementos y propiedades necesarios para ello.

B.2 Exportación a UIML de los diagramas

En las siguientes líneas se exponen las características generales que deben cumplir los archivos UIML para que puedan ser importados por Evalgram:

- Se ha de utilizar como base el vocabulario genérico Generic_1.2_Harmonia_1.0.uiml desarrollado por Harmonia (UIML.org, 2002)
- Los identificadores IDs aportarán un nombre único (unívoco) y los caracteres que compongan cada identificador serán de tipo alfanumérico.
- La herramienta Evalgram requiere la existencia previa de un documento UIML, en el que se determinen los elementos y las interacciones existentes. No es necesario que este documento precise el aspecto estético del prototipo, sólo su contenido.
- La especificación previa de un documento UIML debe contener la información elemental que describa la interfaz de usuario. La herramienta Evalgram establece unas exigencias

mínimas, con el objeto de facilitar y agilizar la creación de documentos UIML de forma manual, ya que se centra en la evaluación temprana de interfaces de usuario, no en la creación y definición inicial de estos prototipos de interfaces.

- La descripción básica de la interfaz de usuario en un documento UIML debe constar de:
  - La definición de los diversos elementos de interacción existentes en el prototipo software descrito, que deben estar mínimamente especificados a través de las propiedades obligatorias designadas para cada caso en el vocabulario UIML propuesto y empleado por Evalgram.
  - Los eventos de interacción existentes entre las diferentes pantallas del prototipo descrito en el documento UIML.

En la Figura B.1 se describe la estructura exigida del documento UIML:

```
<uiml>
        <head>Prototipo</head>
        <interface>
                <structure>
                   //Pantallas del prototipo software, y en la descripción de cada pantalla,
                   //se determinan los elementos de interacción contenidos
                   ...
                </structure>
                <behavior>
                   // Eventos existentes (rules) en el prototipo software
                   ...
                 </behavior>
        </interface>
</uiml>
```

Figura B.1 Estructura básica UIML requerida

Cuando se exporte el prototipo de navegación generado por Diagram, dentro de la etiqueta <structure> se guardará la información necesaria concerniente a todas y cada una de las ventanas del prototipo y, dentro de ellas se guardarán los diferentes elementos de interacción, que en este caso serán los botones que representan a las tareas, junto con la ruta y el botón de retorno. Dentro de la etiqueta <behavior> se guardarán las acciones de los botones, es decir, qué pantalla debe ocultarse y cuál mostrarse una vez pinchado un botón de la ventana. Además, las propiedades tanto de las ventanas como de los elementos de interacción se determinan dentro de la definición del elemento <part>, mediante el apartado interno <style>, todo ello dentro de la etiqueta <structure>. Esto es, por cada ventana que tenga el prototipo se deberá crear un elemento <part> y definir sus propiedades con la etiqueta <style>. A su vez,

por cada elemento de interacción que contenga la ventana (botones, texto…) se creará una nueva etiqueta <part> dentro de la misma, como se observa en la Figura B.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <uiml>
    <!-- Exportado desde Diagram  -->
    <!-- Día: mié, 30 abr 2008  -->
    <head>Grabar Disco de Música</head>
  - <interface>
    - <structure>
      - <part class="G:TopContainer" id="P0">
        - <style>
            <property name="g:title" part-name="P0">Grabar Disco de Música</property>
          - <property name="g:size" part-name="P0">
              <width>600</width>
              <height>600</height>
            </property>
            <property name="description" part-name="P0">Grabar Disco de Música</property>
          </style>
        - <part class="G:Text" id="rut0">
          - <style>
            - <property name="g:size" part-name="rut0">
                <height>30</height>
                <width>400</width>
              </property>
            - <property name="g:location" part-name="rut0">
                <x>100</x>
                <y>70</y>
              </property>
              <property name="g:text" part-name="rut0">Grabar Disco de Música</property>
            </style>
          </part>
```

Figura B.2 Estructura detallada UIML requerida

En la Tabla B.1 se muestran los elementos de interacción que se van a utilizar en la exportación del prototipo a UIML. Estos cuatro elementos de interacción, serán suficientes para generar un documento UIML válido, que represente al prototipo de navegación de un Modelo de Tareas de Usuario y que permita la importación desde Evalgram.

Tabla B.1 Elementos de interacción UIML básicos escogidos

Elemento	Propósito
G:TopContainer	Las pantallas del prototipo
G:Button	Los botones que representan a las tareas y el retorno
G:Text	La ruta o migas de pan
G:Image	Imagen para separar la ruta de los botones

Además, las propiedades de los diferentes elementos de interacción que se utilizarán serán las mostradas en la Tabla B.2.

Tabla B.2 Propiedades UIML de los elementos de interacción, escogidas

Propiedad	**Descripción**
g:location	Especifica la posición del elemento
g:size	Especifica el tamaño del elemento
g:text	Especifica el texto que contendrá el elemento
g:title	Especifica el titulo del elemento de interacción
g:visible	Especifica la visibilidad de elemento (se utilizará para ocultar y mostrar las diferentes ventanas)
g:buttonType	Especifica el tipo de botón (en este caso siempre será push)
G:actionperformed	Determina la acción a realizar cuando el usuario pinche un botón

Una vez vistos los elementos y las propiedades a utilizar, queda describir la manera en la que se guardarán las acciones de los distintos botones, es decir, el contenido del elemento <behavior> mencionado anteriormente y que se muestra en la Figura B.3. Por cada botón del prototipo, se abrirá una nueva etiqueta <rule>, donde se definirán la condición y la acción que corresponda. Mediante la etiqueta <condition> se indicará a qué botón pertenece dicha regla, mientras que en <action> se definirá qué ventana se debe ocultar y cuál mostrar mediante la propiedad “g:visible”.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <uiml>
    <!-- Exportado desde Diagram  -->
    <!-- Día: mié, 30 abr 2008  -->
    <head>Grabar Disco de Música</head>
  - <interface>
    + <structure>
    - <behavior>
      - <rule>
        - <condition>
            <event name="g:actionperformed" part-name="bot1" />
          </condition>
        - <action>
            <property name="g:visible" part-name="P0">false</property>
            <property name="g:visible" part-name="P1">true</property>
          </action>
        </rule>
      - <rule>
        - <condition>
            <event name="g:actionperformed" part-name="bot2" />
          </condition>
        - <action>
            <property name="g:visible" part-name="P0">false</property>
            <property name="g:visible" part-name="P2">true</property>
          </action>
        </rule>
```

Figura B.3 Especificación de la propiedad <behavior> escogida en UIML

B.3 UIML extendido (en la versión Diagram 3.0)

En Diagram 3.0 se amplían las posibilidades de modelado con el *Modelo del Diálogo*. Para realizar la exportación al lenguaje UIML de su prototipo además de los elementos de interacción (Tabla B.1) y las propiedades de los diferentes elementos de interacción para el prototipo de tareas de usuario (Tabla B.2), es necesario incluir las propiedades de los elementos de interacción que se muestran en la Tabla B.3. Las propiedades g:typeTask y g:goTo son necesarias para representar el *Modelo de Comportamiento* y las etiquetas g:typePresentation y g:section representan al *Modelo del Prototipo*.

Tabla B.3 Propiedades UIML exclusivas del Prototipo del Diálogo

Propiedad	Descripción
g:typeTask	Indica el tipo de tarea
g:typePresentation	Indica el tipo de presentación de las hijas de una tarea
g:goTo	Indica cuál será la siguiente tarea a ejecutar
g:section	Indica en que sección está representada la tarea

A partir de Diagram 3.0, el contenido que admite cada propiedad es diferente dependiendo de lo que se quiera representar. Los tipos de datos de las nuevas etiquetas son los siguientes:

Tabla B.4 Contenido asociado a las propiedades UIML de los Modelos de Comportamiento y Prototipo

Propiedad	Contenido
g:typeTask	system | user
g:typePresentation	newWindow | newSection | dropDown
g:goTo	String
g:section	Integer (1..MAX_SECCIONES)

MAX_SECCIONES es una información prevista, que deriva del modelo de usuario/sistema y representa el número máximo de secciones que se permite para la representación en el prototipo del diálogo. En dichos modelos, están recogidas algunas restricciones dadas por el tipo de usuario o sistema que utilice la herramienta.

B.4 Tareas delegadas

En la versión WebDiagram 2.0 se crean las tareas delegadas. Para describirlas, se crea una etiqueta nueva "g:typeExtracted". En la Figura B.4 aparece un trozo de código con la exportación UIML de una parte de un Modelo SE-HCI. En él aparece la etiqueta de tarea

delegada “typeExtracted”. En este caso, como la tarea elegida no es delegada, aparece en modo “false”. Si fuera delegada aparecería en modo “true”.

```
<part class="G:Image" id="im0" />
∇<part class="G:Button" id="bot1">
    ∇<style>
        <property name="g:text" part-name="bot1">Tarea</property>
        <property name="g:typeTask" part-name="bot1">user</property>
        <property name="g:typePresentation" part-name="bot1">newWindow</property>
        <property name="g:typeExtracted" part-name="bot1">false</property>
        <property name="g:section" part-name="bot1">1</property>
        <property name="g:buttontype" part-name="bot1">push</property>
    </style>
</part>
```

Figura B.4 Exportación UIML de un modelo SE-HCI parcial con una tarea delegada

APÉNDICE C. WebDiagram 3.0: Aplicación del proceso InterMod con TDD

C.1 Introducción

La metodología ágil InterMod combinada con la técnica TDD se aplicó en el desarrollo de la versión 3.0 de WebDiagram. Esta herramienta acumulaba el potencial y la funcionalidad que todas sus versiones habían aportado, pero se habían detectado algunos errores semánticos de dificil solución. En la nueva versión debía asegurarse la robustez de su funcionalidad, especialmente en el prototipo del modelo SE-HCI. Por tanto, se utilizó InterMod con TDD para hacer frente a la compleja tarea de corrección de errores de versiones anteriores e integrar nueva semántica en esta versión, en base a la siguiente hipótesis : "*La integración de las técnicas de TDD en Intermod facilita y permite, poco a poco, descubrir y expresar la semántica de la aplicación, es decir, cómo la aplicación debe comportarse*".

En este apéndice se muestra la aplicación del proceso InterMod combinado con TDD que guió el desarrollo de WebDiagram 3.0. Este proceso siguió el esquema descrito en el capítulo 6.3.1.2. Primero, se indican las decisiones generales del proceso y después, se resumen los resultados del desarrollo efectuado en las iteraciones del proceso, que incluyen la Lista de UOs, el Diagrama de Creación de UOs, el conjunto de tests empleado y el análisis de los resultados en cantidad de tests y tiempos de desarrollo y evaluación.

C.2 Decisiones Generales

WebDiagram 3.0 tiene los precedentes de las versiones anteriores y por esta razón, se reutilizaron varios modelos ya evaluados. En particular, la descripción de algunos modelos M-1 fueron revisados, actualizados y evaluados por expertos en usabilidad por medio de prototipos de baja fidelidad. Los modelos M-2 de versiones anteriores fueron reutilizados en su totalidad, por lo que no hubo que realizar evaluaciones de usabilidad sobre ellos, y únicamente se comprobó, mediantes tests, que esta versión los respetaba. Los modelos M-2 nuevos fueron evaluados con expertos en usabilidad en los puntos de evaluación. Los modelos M-3 se evaluaron a través de tests, en los puntos de evaluación. El proceso de creación y evaluación de nuevos modelos se hizo iteración a iteración, siguiendo la propuesta de InterMod combinada con TDD. Por último, cuando se terminó el proceso, también se evaluaron aspectos de eficacia, eficiencia y satisfacción.

En el proceso InterMod combinado con TDD, los modelos M-1 (Modelos SE-HCI) son las pruebas a elaborar y validar paso a paso, y constituyen la semántica de la aplicación. Los modelos M-1 ya creados, se revisaron y acumularon en el conjunto de pruebas.

En este caso, el equipo de implicados fueron un desarrollador y dos expertos en usabilidad. Además, los expertos en usabilidad eran también los clientes de la aplicación. Dos equipos trabajaron en paralelo: un equipo consistía en un experto en usabilidad que creaba los modelos SE-HCI (nuevas pruebas) y el otro, formado por un desarrollador y dos expertos en usabilidad, desarrollaba y evaluaba los modelos. Se establecieron puntos de evaluación semanales, de duración flexible entre media hora y una hora. Con el objetivo de no retrasar los puntos de evaluación con la ejecución de tests, uno de los expertos en usabilidad fue responsable de ejecutar los tests previamente y seleccionar aquéllos que debían tratarse en la reunión, debido a su dificultad.

En la etapa inicial, se estudió una visión global de lo que ya se había realizado y que sería reutilizado con el nuevo material. Además, se tomaron decisiones comunes a todos los objetivos desarrollados, de presentación, navegación y comportamiento.

Presentación:

- El prototipo se lanzará en una ventana que podrá ajustarse en tamaño.
- Únicamente habrá una ventana activa en el prototipo, salvo cuando haya tareas concurrentes.
- El foco debe actualizarse en cada paso, y la activación de tareas debe señalar únicamente las posibles en cada momento.
- No se van a limitar el número de secciones y desplegables posibles, aunque sí se lanzará aviso al llegar a tres acumuladas.
- El diagrama irá señalando en rojo las tareas que se han ejecutado en el prototipo.

Navegación:

- Normalmente existen varias secuencias correctas posibles
- La navegación se considerará correcta si y solo si se efectúa una de las secuencias correctas consideradas en la especificación.

Comportamiento:

- En caso de simulación correcta (en presentación, navegación y comportamiento), el programa de simulación lanzará un aviso de finalización correcta. En caso contrario, el programa lanzará un aviso indicando que se ha cancelado el prototipo.
- Si es posible no realizar más tareas en la ventana activa, ésta podrá cerrarse.

El proyecto arrancó con un UO simple: UO1, y la escritura de los tests asociados con él.

UO1- *Prototipo de tareas unitarias y secuenciales*: Quiero simular un prototipo de un modelo SE_HCI que contenga únicamente tareas unitarias y secuenciales.

C.3 Resultados del proceso InterMod combinado con TDD en WebDiagram 3.0

A lo largo del proyecto, se desarrollaron, iteración a iteración, los modelos necesarios para obtener un prototipo robusto del modelo SE-HCI, que comprende los siguientes modelos (ver apartado 4.3): Modelo de tareas de Usuario (MT), Modelo de Prototipo (MP), Modelo de Comunicación (MC) y Modelo del Comportamiento (MCt).

Cada iteración comenzaba con el *Paso 1* y el *Punto deEvaluación* en el que se ejecutaban los tests seleccionados y se debatían los nuevos tests para los nuevos UOs que se incorporaban. Se continuaba con el *Paso 2* planificando la iteración, esto es, seleccionando los UOs y las actividades a desarrollar por los equipos, y planificando los Tests nuevos. Finalmente, en *el Paso 3* se creaban y evaluaban los modelos, realizando las actividades por equipos. Esto incluía escribir nuevos tests. La gestión de los tests era una actividad necesaria que se realizaba al final de cada iteración para reunir los tests escritos hasta el momento.

Se muestra a continuación la Lista de UOs desarrollados en WebDiagram 3.0, el Diagrama de Creación de UOs y la contabilidad en cada iteración de UOs, tests y tiempos.

C.3.1 Lista de UOs desarrollados en WebDiagram 3.0

En la Tabla C.1 aparece la lista de UOs desarrollados a lo largo de este proyecto. Se tomó la siguiente decisión en cuanto a la nomenclatura de UO: Su nombre identifica, de forma simplicada, a los modelos que desarrolla (parcialmente, en su mayoría) y el número de UO al que pertenece. Las descripciones indican el tipo de tareas que se quiere simular en un prototipo y aquéllas que están en negrita identifican a un modelo finalizado cuya prueba con usuarios finales debe ser especialmente tratada.

Tabla C.1 Lista de UOs del proyecto WebDiagram 3.0

UO1- *MT + MP-1*: Tareas Secuenciales unitarias presentadas en nuevas Ventanas
UO2- *MT+MP-2*: Tareas Secuenciales Opcionales presentadas en nuevas Ventanas

UO3- *MT+MP-3*: Tareas Secuenciales Repetitivas presentadas en nuevas Ventanas
UO4- *MT+MP-4*: Tareas Secuenciales presentadas en nuevas Ventanas
UO5- *MT+MP-5*: Tareas Elección presentadas en nuevas Ventanas
UO6- *MT+MP-6*: Tareas Secuenciales –Elección presentadas en nuevas Ventanas
UO7- *MT+MP-7*: Tareas Indiferentes presentadas en nuevas Ventanas
UO8- *MT+MP-8*: Tareas Secuenciales-Elección-Indiferentes presentadas en nuevas Ventanas.
UO9- *MT+MP-9*: Tareas Concurrentes presentadas en nuevas Ventanas.
UO10-*MT+MP-10*: **Modelo de Tareas de Usuario**: Secuenciales-Elección-Indiferentes-Concurrentes presentadas en nuevas Ventanas
UO11-*MP-11*: Presentación en Secciones
UO12- *MT+MP-12*: Tareas Secuenciales-Elección-Indiferentes presentadas en nuevas Ventanas y Secciones
UO13- *MT+MP-13*: Modelo de Tareas de Usuario y Modelo de Prototipo UO10 + UO12
UO14- *MC+MCt-14*: Tareas del Sistema y propiedad "Ir a"
UO15- *MT+MP+MC+MCt-15*: Modelo de Tareas de Usuario, Modelo del Prototipo, Modelo de Comunicación y Modelo de Comportamiento
UO16- *MT+MP+MC+MCt-16:* Tareas Secuenciales-Elección-Indiferentes con presentación en nuevas Ventanas y Secciones y Tareas del Sistema-Ir a
UO17- *MT+MP+MC+MCt-17*: Tareas Secuenciales-Elección-Indiferentes-Concurrentes con presentación en nuevas Ventanas y Tareas del Sistema-Ir a
UO18- *MP*: Tareas Desglosadas
UO19 - *SE-HCI:* **Modelo SE-HCI**

C.3.2 Diagrama de creación de UOs

Los UOs de este proyecto surgieron en las primeras 14 iteraciones, como muestra la Figura C.1. El desarrollo partió de UO1, recogiendo sus tests antes de la primera iteración y fue completándose mediante incrementos, fusiones y divisiones hasta llegar al UO Directo UO19. Los nombres que acompañan a la representación gráfica de un UO, como TSu o TSE, identifican a los grupos de tests para ese UO, que se muestran en el capítulo C.4.

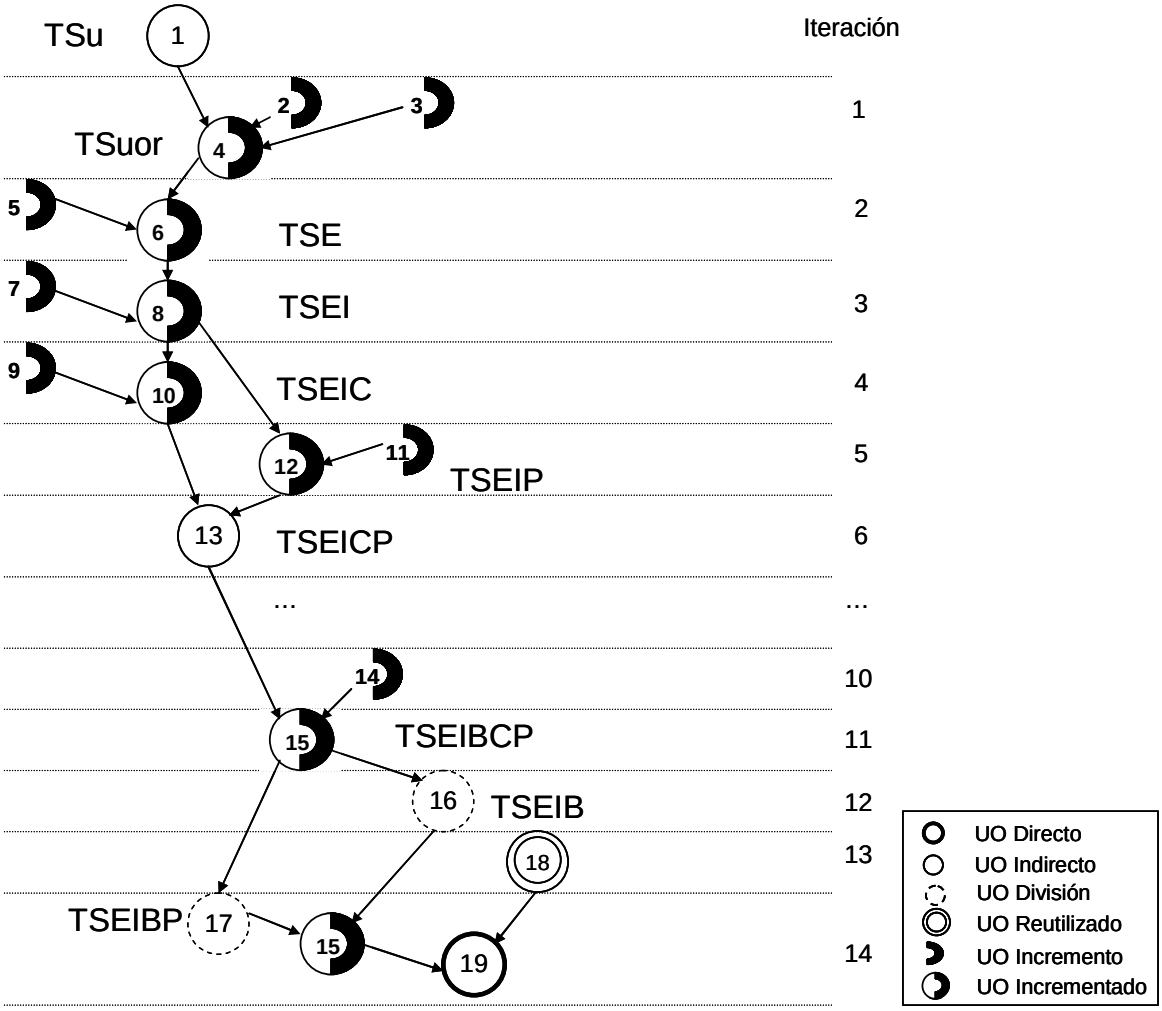


Figura C.1 Diagrama de Creación de UOs en el proyecto WebDiagram 3.0

C.3.3 Resultados del proceso

Con el fin de ilustrar el proceso seguido, la Tabla C.2 muestra la información del proceso en las 23 iteraciones del desarrollo de WebDiagram 3.0. Cada fila muestra un registro del progreso de la iteración: el número de iteración (**Ite**.), los nuevos UOs que surgen y se seleccionan, los tests pasados (**P**) y fallidos, distinguiéndo entre los tests nuevos fallidos (**NF**) y los tests de regresión fallidos (**RF**), los tiempos de evaluación de los tests en preEvaluación (**pET**) y en el Punto de Evaluación (**ET**), y los tiempos empleados en el desarrollo de los modelos de UOs (**M-1, M-2** y **M-3**), y finalmente, los tests nuevos escritos según la planificación (**NT**) y en el Punto de Evaluación (**NTe**).

Tabla C.2 Número de tests y Tiempos de ejecución y confección

	UOs		Pasan/Fallan			Tiempo (min)				Tests		
Ite.	**nuevos**	**seleccionados**	**P**	**NF**	**RF**	**pET**	**ET**	**M2&M3 DT**	**M1 DT**	**Nte**	**NT**	**TT**
	1	1							15	0	43	**43**
1	2,3,4	1,2,3,4	39	4	0	0	20	240	30	0	12	**55**
2	5,6	4,5,6	43	12	0	0	30	240	30	1	7	**63**
3	7,8	6,7,8	54	9	0	0	30	325	60	0	17	**80**
4	9,1	8,9,10	60	20	0	0	30	270	120	2	27	**109**
5	11,12	10,11,12	73	36	0	0	30	360	60	2	12	**123**
6	13	10,12,13	77	46	0	0	30	160	60	2	12	**137**
7		13	87	50	0	30	45	370	30	1	0	**138**
8		13	85	48	**5**	30	45	410	30	1	0	**139**
9		13	110	29	0	30	60	365	30	3	0	**142**
10	14	13,14	121	21	0	30	60	365	90	**7**	10	**159**
11	15	13,14,15	125	33	1	45	45	410	60	2	0	**161**
12	16,17	13,16,17	138	22	1	45	45	260	60	2	10	**173**
13	18	16,18	139	34	0	45	60	70	**30**	4	16	**193**
14		15,16,17,18,19	150	43	0	60	60	150	**120**	0	14	**207**
15		16,19	168	38	1	60	60	24	0	1	0	**208**
16		16	166	42	0	60	60	325	30	1	0	**209**
17		16	169	39	1	60	60	320	30	2	0	**211**
18		16,19	172	38	1	60	60	130	30	1	0	**212**

	UOs		Pasan/Fallan			Tiempo (min)				Tests		
Ite.	**nuevos**	**seleccionados**	**P**	**NF**	**RF**	**pET**	**ET**	**M2&M3 DT**	**M1 DT**	**Nte**	**NT**	**TT**
19		19	198	14	0	60	60	175	30	3	0	**215**
20		19	210	5	0	60	45	120	30	1	0	**216**
21		19	213	3	0	60	45	130	30	2	0	**218**
22		19	215	3	0	60	30	50	30	0	0	**218**
23		19	218	0	0	60	30	70	0	0	0	**218**
							1895	**5339**	**1020**			

La contabilidad de los tests en cada iteración (**TT**) suma los tests de la anterior iteración y los tests creados en esta iteración.

En las primeras iteraciones se reutilizaron muchos modelos de las versiones previas, por lo que el número de tests que pasan (**P**) era numeroso, esto se observa por ejemplo en la iteración 1 en la que **Pasan**: 39 y **Fallan**: 4. A medida que el proyecto avanza esta diferencia disminuye (iteración 5: **Pasan** :73 y **Fallan**:36) hasta llegar a las últimas iteraciones en las que el número de tests que pasan correctamente aumenta progresivamente.

Los Tests de regresión que fallaron son la clave del éxito de esta metodología. Este daño colateral es difícilmente detectable con un tratamiento de pruebas tradicional. Estos tests detectaron fallos en pruebas que habían pasado anteriormente con éxito, pero que fallaron posteriormente tras la incorporación de nueva semántica en la aplicación. Por ejemplo, en la Tabla C.2 se observan 5 fallos en tests de regresión (**RF**) en la iteración 9.

En cuanto al tiempo de evaluación se distinguen por un lado, el tiempo de PreEvaluación (**pET**) de tests que indican el tiempo que un evaluador tardó en ejecutar todos los tests antes de la reunión en el *Punto de Evaluación* y por otro lado, el tiempo de Evaluación (**ET**) de tests en el *Punto de Evaluación*. Se observa en la columna **pET** de la Tabla C.2 que en las primeras iteraciones no hubo tiempo de PreEvaluación porque había pocos tests. Sin embargo, a partir de la iteración 8 este tiempo fue de 30 minutos, aumentandose posteriormente a 45 minutos, y a 60 minutos al final. Este tiempo permitió aligerar las reuniones en el Punto de Evaluación, y utilizarlas para ejecutar e intentar solucionar aquellos tests que habían fallado.

La columna **ET,** que indica el tiempo de estas reuniones, oscila entre 20 y 60 minutos y refleja la dificultad de los tests fallidos y el debate para su solución.

El tiempo necesario para la creación de los modelos M-1, M-2 y M-3 fue variable dependiendo de su dificultad y su reutilización. Por ejemplo, en la iteración 13 la columna **M1 DT** es de 30 minutos, debido a la reutilización de UO18 mientras que en la iteración 14 es de 120 minutos, ocupados en la confección de tests para la fusión de UOs.

Finalmente, destacan los nuevos tests que surgen en el Punto de Evaluación (**Nte**), tras el debate entre los miembros del equipo, como por ejemplo, en la iteración 10 se realizan 7 tests en el Punto de Evaluación.

C.4 Conjunto de Tests TDD

Los tests para WebDiagram 3.0 tienen entradas y salidas gráficas. A partir de una entrada gráfica (un diagrama de tareas) se comprueban varios comportamientos y presentaciones en un prototipo de tipo gráfico (ventanas, botones, secciones, colocación, número y orden de accesos, etc). Por lo tanto, el proceso tradicional de tests empleado en TDD con herramientas tipo jUnit no fueron posibles. En esta versión, las unidades de test no fueron programadas. Se realizó un proceso similar al lanzamiento automático de tests pero provocando la ejecución manualmente, esto es, dada una figura, se comprobó si los efectos coincidían con la especificación. Se elaboraron dos tipos de tests para probar el desarrollo de WebDiagram 3.0:

- Tests para comprobar la visualización y ejecución correcta del diagrama, que incluyen aspectos visuales, activación de elementos y avisos.
- Tests para comprobar la visualización y ejecución correcta del prototipo.

C.4.1 Tests de visualización y ejecución del diagrama

Estos tests se encuadraron en cuatro tipos: una tarea inicial, una tarea hija sin hijas, una tarea con hija con hijas y una tarea concurrente. Todos salvo el último tipo, son tests que comprueban la consistencia de esta versión con las anteriores, esto es, la reutilización correcta de la visualización y ejecución del diagrama. Además, estos tests corrigen algunos errores de versiones anteriores.

C.4.1.1 T0.tareas: Una tarea inicial

Tabla C.3 Tests del diagrama para una tarea inicial en Webdiagram 3.0

OPCIONES	EJECUTABLE	MODO
Nombre, informe, comentario	S	
Crear hijas hacia abajo	S	
Crear tarea a la izda	N	**Aviso**: En la tarea ppal solo se pueden crear hijas hacia abajo
Crear hija a la dcha	N	**Aviso**: En la tarea ppal solo se pueden crear hijas hacia abajo
Mover tarea a la izda	N	**Aviso**: La tarea ppal no se puede mover a la izda
Mover tarea a la dcha	N	**Aviso**: La tarea ppal no se puede mover a la dcha
Borrar	N	**Aviso**: La tarea ppal no se puede borrar
Orden	N	**Desactivado**
Tipo	N	**Desactivado**
Presentación	N	**Desactivado**
Ir a	N	**Desactivado**
Conversión U/S	N	**Aviso**: La tarea ppal no puede ser una tarea del Sistema.
Glosar/Desglosar	N	**Aviso**: La tarea ppal no se puede desglosar

C.4.1.2 T1.tareas: una tarea hija sin hijas

Para la tarea 0, las opciones activas y no activas son las especificadas en T0.tareas.

Para la tarea 1:

Tabla C.4 Tests del diagrama para una tarea sin hijas en Webdiagram 3.0

OPCIONES	EJECUTABLE	MODO
Nombre, informe, comentario	S	
Crear hijas hacia abajo	S	
Crear tarea a la izda	S	
Crear hija a la dcha	S	
Mover tarea a la izda	N	**Aviso**: No hay más tareas a la izda
Mover tarea a la dcha	N	**Aviso**: No hay más tareas a la dcha
Borrar	S	
Orden	S	**Aviso**: No es posible orden elección-tipo opcional
Tipo	S	**Aviso**: No es posible un tarea hoja repetitiva. **Aviso**: No es posible orden elección-tipo opcional
Presentación	N	**Desactivado** en todas las tareas hojas
Ir a	S	
Conversión U/S	S	

OPCIONES	EJECUTABLE	MODO
Glosar/Desglosar	S	

C.4.1.3 T2.tareas: una tarea con hija con hijas

Para la tarea 0, las opciones activas y no activas son las especificadas en T0.tareas.

Para la tarea 1:

Tabla C.5 Tests del diagrama para una tarea con hija con hijas en Webdiagram 3.0

OPCIONES	EJECUTABLE	MODO
Nombre, informe, comentario	S	
Crear hijas hacia abajo	S	
Crear tarea a la izda	S	
Crear hija a la dcha	S	
Mover tarea a la izda	N	**Aviso**: No hay más tareas a la izda
Mover tarea a la dcha	S	
Borrar	S	**Aviso**: Las subtareas de esta tarea serán eliminadas. ¿Desea borrar la tarea?
Orden	S	

OPCIONES	EJECUTABLE	MODO
Tipo	S	**Aviso**: No es posible orden elección-tipo opcional **Aviso**: Repetitiva Min>Max o Min=Max=0 **Aviso**: No es posible orden elección–tipo concurrente **Aviso**: No es posible orden indiferente-tipo concurrente
Presentación	S	
Ir a	N	**Desactivado** en todos los nodos intermedios
Conversión U/S	N	**Aviso**: Una tarea con hijos no puede ser del sistema.
Glosar/Desglosar	S	**Aviso**: Al desglosar

C.4.1.4 TC_1.tareas: Una tarea concurrente

Para la tarea 0, las opciones activas y no activas son las especificadas en T0.tareas.

Tarea 1:

Tabla C.6 Tests del diagrama para una tarea concurrente en Webdiagram 3.0

OPCIONES	EJECUTABLE	MODO
Nombre, informe, comentario	S	
Crear hijas hacia abajo	S	

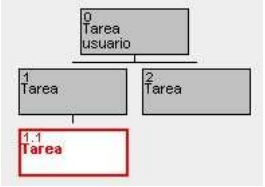
OPCIONES	EJECUTABLE	MODO
Crear tarea a la izda	S	
Crear hija a la dcha	S	
Mover tarea a la izda	N	**Aviso**: No hay más tareas a la izda.
Mover tarea a la dcha	N	**Aviso**: No hay más tareas a la dcha.
Borrar	S	
Orden	N	**Desactivado**: Las tareas concurrentes no son de ningún orden.
Tipo	N	**Desactivado**: Las tareas concurrentes no son de ningún tipo.
Presentación	N	**Desactivado** en todas las tareas hojas
Ir a	S	
Conversión U/S	S	Pierde la característica de concurrencia
Glosar/Desglosar	S	**Aviso**: Al desglosar

C.4.2 Tests de visualización y ejecución del prototipo

Estos tests se agrupan en unidades de tests asociadas a un UO, para probar la correcta visualización y ejecución del prototipo que genera su modelo SE-HCI. El nombre de la prueba identifica brevemente la descripción del UO y su numeración. El efecto de la prueba indica el error o aviso que produce, el aspecto del prototipo y la secuencia de acciones posibles para una ejecución correcta del prototipo.

C.4.2.1 Unidades de Test UO1: Tareas de orden secuencial-unitarias (TSu)

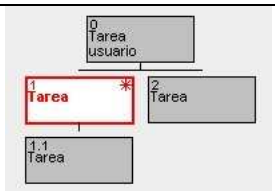
Tabla C.7 Tests del prototipo para TSu en Webdiagram 3.0

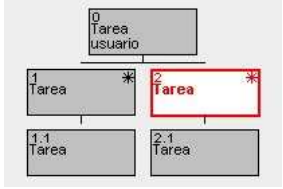
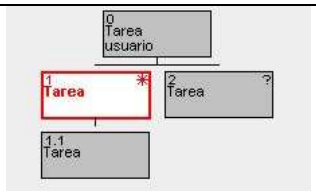
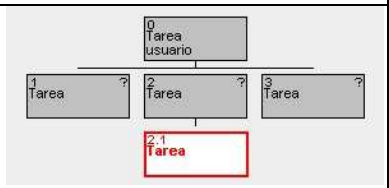
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSu_1		Una tarea principal	**AVISO**: No hay prototipo
TSu_2		Una tarea principal con una hija unitaria	Una ventana con un botón. Se pulsa y finaliza correctamente Secuencia: 1
TSu_3		Varias hijas secuenciales y unitarias	Una ventana con tres botones. Hay que pulsarlos secuencialmente para finalizar correctamente. Secuencia: 1-2-3
TSu_4		Varias hijas y nietas de orden secuencial y tipo unitario	Una ventana con dos botones a activar secuencialmente. El primero produce un botón que ha de activarse: 1-1.1-2

C.4.2.2 Unidades de Test UO4: TSu + Tareas de orden secuencial opcionales-repetitivas (TSuor)

Tabla C.8 Tests del prototipo para TSuor en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSuor_1		Una tarea principal con una hija de tipo opcional	Una ventana con un botón. Se pulsa o se cierra la ventana y finaliza correctamente Secuencias: 1, x

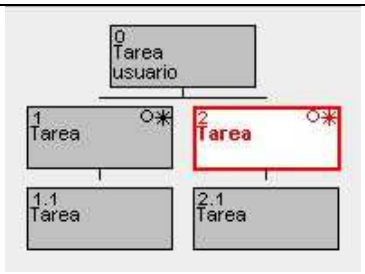
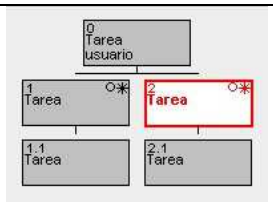
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSuor_2	AVISO: No es posible porque la tarea repetitiva no tiene hijas.	Una tarea principal con una hija de tipo repetitivo	
TSuor_3		Varias hijas secuenciales; la primera, opcional	Una ventana con dos botones. Finalizará correctamente si se pulsan secuencialmente o si se pulsa únicamente el segundo. Secuencias: 1-2, 2
TSuor_4		Varias hijas secuenciales, de tipo opcional	Una ventana con dos botones. Finalizará correctamente si se pulsan secuencialmente o si se pulsa uno de ellos o si se cierra la ventana. Secuencias: 1-2, 1, 2, x
TSuor_5		Varias hijas y nietas de orden secuencial, una hija opcional	
TSuor_6		Varias hijas y nietas de orden secuencial, una hija repetitiva Min=Max=0	AVISO: Las repetitivas deben poder efectuarse al menos una vez. La convierte en unitaria
TSuor_7		Varias hijas y nietas de orden secuencial, una hija repetitiva Min=Max=1	La convierte en unitaria Secuencia: 1-2
TSuor_8		Varias hijas y nietas de orden secuencial, una hija repetitiva Min=0, Max=999	Secuencias: (1-1.1)*- 2 2

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSuor_9		Todas las hijas repetitivas Min=0, Max=999	Secuencias correctas: (1-1.1)*- (2-2.1)* (1-1.1)* (2-2.1)* X
TSuor_10		Tareas de orden secuencial. Una hija repetitiva min=2, Max=4, la otra opcional	Secuencias correctas: (1-1.1)[2..4]- 2 (1-1.1)[2..4]
TSuor_11		Tareas de orden secuencial. Todas opcionales.	Secuencias correctas: 1-2-2.1-3 2-2.1-3 1-3 1 2 3 x
TSuor_12		Tarea secuencial, iterativa de 0 a 3 veces.	Secuencias correctas: (1-1.1)* máx 3 veces x

C.4.2.3 Unidades de Test UO6: TSu + TSuor + Tareas de orden elección en nueva ventana (TSE)

Tabla C.9 Tests del prototipo para TSE en Webdiagram 3.0

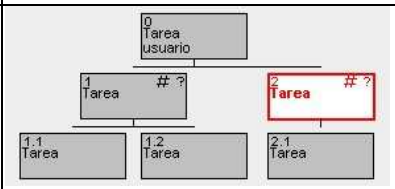
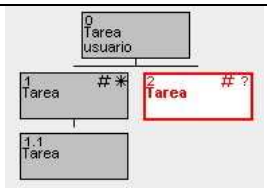
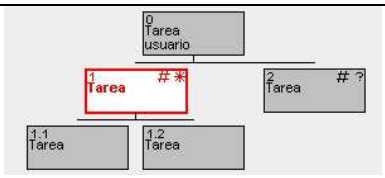
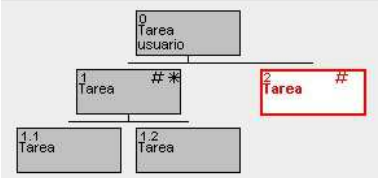
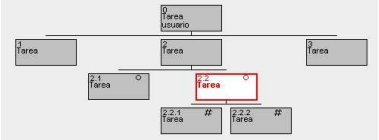
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSE_1		Una tarea principal con una hija de orden elección y tipo unitaria	Una ventana con un botón. Se pulsa y finaliza correctamente. Secuencia: 1

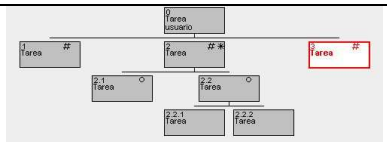
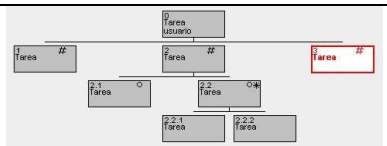
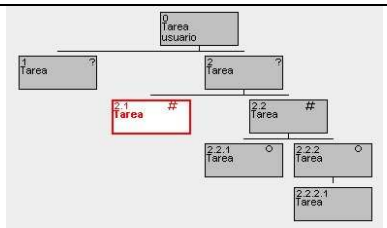
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSE_2	AVISO: No es posible una tarea de orden elección y tipo opcional	Una tarea principal con una hija de orden elección y tipo opcional	
TSE_3		Varias hijas de orden elección, unitarias	Una ventana con dos botones. Finalizará correctamente si se pulsan únicamente uno. Secuencias: 1, 2
TSE_4		Varias hijas y nietas. Las hijas de orden elección y repetitivo, Min=0, Max=999	AVISO: Las tareas repetitivas de orden elección, su Min debe ser mayor que 0.
TSE_5		Varias hijas y nietas. Las hijas de orden elección y repetitivo, Min=1, Max=999	Secuencias correctas: (1-1.1)* (2-2.1)* X
TSE_6		Varias hijas y nietas de orden elección y unitarias	Secuencias correctas: 1-1.1-1.2 2-2.1
TSE_7		Varias hijas y nietas de orden elección y unitarias	Secuencias correctas: 1-2-2.1-3 1-2-2.2-3

C.4.2.4 Unidades de Test UO8: TSu + TSuor + TSE + Tareas de orden indiferentes en nueva ventana (TSEI)

Tabla C.10 Tests del prototipo para TSEI en Webdiagram 3.0

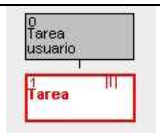
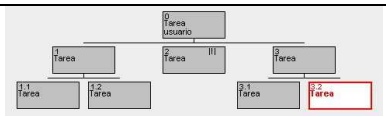
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEI_1		Una tarea principal con una hija de orden indiferente y tipo unitaria	Una ventana con un botón. Se pulsa y finaliza correctamente. Secuencia: 1
TSEI_2		Una tarea principal con una hija de orden indiferente y tipo opcional	Una ventana con un botón. Se pulsa o se cierra la ventana y finaliza correctamente. Secuencias: 1, x
TSEI_3		Varias hijas de orden indiferente, unitarias	Una ventana con dos botones. Finalizará correctamente si se pulsan los dos botones (en cualquier orden). Secuencias: 1-2, 2-1
TSEI_4		Varias hijas de orden indiferente, una opcional	Una ventana con dos botones. Finalizará correctamente si se pulsan los dos en cualquier orden o si se pulsa únicamente el segundo. Secuencias: 1-2, 2-1, 2
TSEI_5		Varias hijas de orden indiferente, todas opcionales	Una ventana con dos botones. Finalizará correctamente si se pulsan en cualquier orden o si se pulsa uno de ellos o si se cierra la ventana. Secuencias: 1-2, 2-1, 1, 2, x
TSEI_6		Varias hijas y nietas de orden indiferente y unitarias	Secuencias correctas: 1-1.1-1.2-2-2.1 2-2.1-1-1.1-1.2

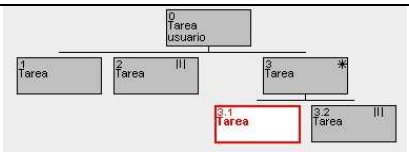
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEI_7		Varias hijas, y nietas, de orden indiferente y de tipo opcional y unitaria	Secuencias correctas: 1-1.1-1.2-2-2.1 2-2.1-1-1.1-1.2 2-2.1
TSEI_8		Varias hijas, y nietas, de orden indiferente y de tipo opcional	Secuencias correctas: 1-1.1-1.2-2-2.1 2-2.1-1-1.1-1.2 1-1.1-1.2 2-2.1 x
TSEI_9		Varias hijas, y nietas, de orden indiferente y de tipo repetitivo (Min=0, Max=999), y opcional	Secuencias correctas: (1-1.1-1.2)*-2-2.1 2-2.1-(1-1.1-1.2)* (1-1.1-1.2)* 2-2.1 x
TSEI_10		Hijas De orden indiferente. Una repetitiva min=2, Max=4, la otra opcional	Secuencias correctas: (1-1.1)[2..4] (1-1.1)[2..4] – 2 2 - (1-1.1)[2..4]
TSEI_11		Hijas de orden indiferente. Una repetitiva min=1. Max=999, la otra opcional	Secuencias correctas: (1-1.1-1.2)[1..999] (1-1.1-1.2)[1..999] – 2 2 - (1-1.1-1.2)[1..999]
TSEI_12		Orden indiferente. Iterativa min=2 Max=999	(1-1.1-1.2)*[2..999]-2 2-(1-1.1-1.2)*[2..999]
TSEI_13		Varias hijas, y nietas, de orden secuencial, elección e indiferente	Secuencias correctas: 1-2-2.1-3 1-2-2.2-2.2.1-2.2.2-3 1-2-2.2-2.2.2-2.2.1-3

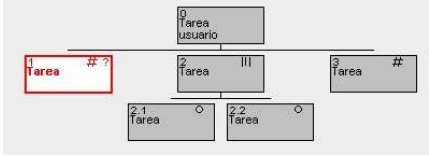
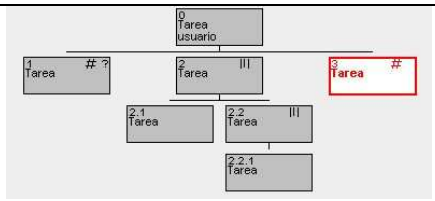
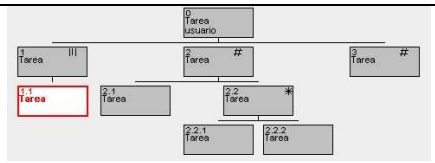
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEI_14		Varias hijas, y nietas, de orden secuencial, elección e indiferente	Secuencias correctas: 1-2-2.1-3 1-2-2.2-2.2.1-2.2.2-3 2-2.1-1-3 2-2.1-3-1 2-2.2-2.2.1-2.2.2-1-3 2-2.2-2.2.1-2.2.2-3-1 3-1-2-2.1 3-1-2-2.2-2.2.1-2.2.2
TSEI_15		Varias hijas, y nietas, de orden secuencial, elección e indiferente. Incluye tipo repetitivo con orden indiferente.	Secuencias correctas: 1-(2-2.1|2-2.2-2.2.1-2.2.2)*-3 (2-2.1|2-2.2-2.2.1-2.2.2)*-1-3 (2-2.1|2-2.2-2.2.1-2.2.2)*-3-1 3-1-(2-2.1|2-2.2-2.2.1-2.2.2)* 3-(2-2.1|2-2.2-2.2.1-2.2.2)*-1
TSEI_16		Varias hijas, y nietas, de orden secuencial, elección e indiferente. Incluye tipo repetitivo con orden elección.	Secuencias correctas: 1-2-(2.1|(2.2-2.2.1-2.2.2)*)-3 2-(2.1|(2.2-2.2.1-2.2.2)*)-1-3 2-(2.1|2-2.2-2.2.1-2.2.2)*)-3-1 3-1-2-(2.1|(2-2.2-2.2.1-2.2.2)*) 3-2-(2.1|2-2.2-2.2.1-2.2.2)*)-1
TSEI_17		Varias hijas, y nietas, de orden secuencial, elección e indiferente, con diferentes tipos.	

C.4.2.5 Unidades de Test UO10: TSu+TSuor+TSE+TSEI+tareas Concurrentes en ventanas (TSEIC)

Tabla C.11 Tests del prototipo para TSEIC en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIC_1		Una tarea principal con una hija concurrente	Una ventana con un botón. Se pulsa, aparecerá una ventana de información que se cierra y vuelve al estado inicial. Finaliza correctamente si se cierra la ventana Secuencias: 1-x, x, (1)*-x
TSEIC_2		Una tarea principal con una hija concurrente y tipo opcional	**Desactivado**: Si es concurrente no será posible otro tipo u orden.
TSEIC_3		Varias hijas, una o varias son concurrentes sin hijas	En la ventana actual aparecerán siempre activas las tareas concurrentes, que podrán hacerse tantas veces como se quiera y en el orden que se quiera, o no activarlas ninguna vez. Finaliza correctamente cerrando la ventana, si las tareas obligatorias se han efectuado en el orden y número correcto. En este caso, solo hay una ventana activa.
TSEIC_4		Varias hijas, una o varias son concurrentes con hijas	En este caso, hay tantas ventanas activas como tareas concurrentes se estén efectuando. La tarea concurrente (sus hijas) se ejecuta en una ventana en paralelo con la ventana principal.
TSEIC-5		Una hija concurrente, hija de una tarea opcional	(1-1.1*)?-2

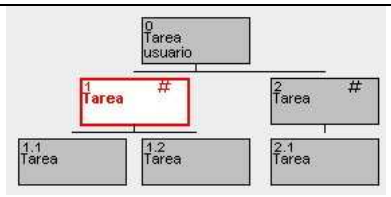
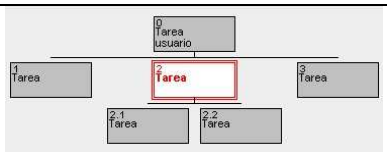
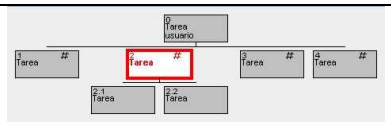
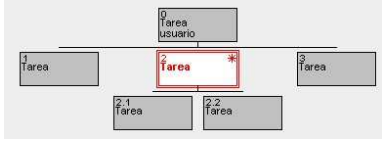
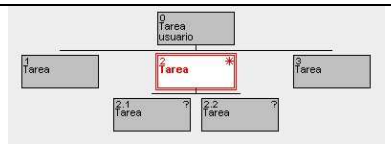
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIC-6		Una hija concurrente hija de una tarea repetitiva	En la ventana actual aparecerán siempre activas la tarea concurrente Secuencias correctas: (1-1.1*)*-2
TSEIC_7		Una tarea concurrente con hermanas de tipo elección	Aparece siempre activa la tarea concurrente. Se ha de cerrar la ventana. 3*-1-3* 3*-2-3*
TSEIC_8		Varias concurrentes en una ventana	Finaliza cerrando la ventana. (2*-3*)*-1-(2*-3*)*
TSEIC_9		Concurrentes en varias ventanas. Tareas repetitivas, hermanas de concurrentes.	Las ventanas que tienen concurrentes, cuando finalizan las tareas obligatorias, deben cerrarse si no desea ejecutarse más veces las concurrentes situadas en esa ventana. **Ventana 1**: 2*-1-2*-3* (ventana2)-2* **Ventana 2**: 3.2*-3.1-3.2*
TSEIC_10		Concurrentes con hijas en nueva ventana	Deben cerrarse **en orden** cuando se efectúan todas las tareas obligatorias de la ventana correspondiente. **Ventana 1**: 2*(ventana2)-1-2*(ventana2)-3-2*(ventana2) **Ventana 2**: 2.2*(ventana3)-2.1-2.2*(ventana3) **Ventana 3**: 2.2.1-2.2.2

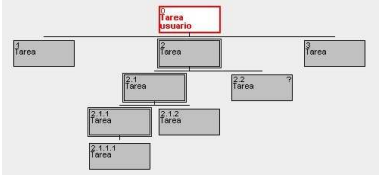
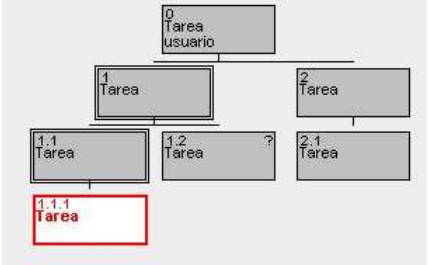
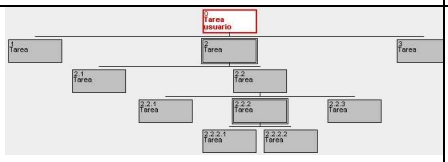
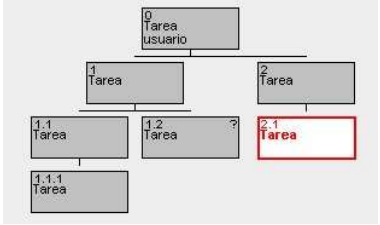
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIC_11		Una tarea concurrente con hermanas de tipo indiferente	Sólo una ventana. 1-2*-3 3-2*-1 Debe cerrarse la ventana para concluir el proceso.
TSEIC_12		Una tarea concurrente con hermanas de tipo indiferente. La concurrente tiene hijas.	Habrá tantas ventanas en paralelo como activaciones de la tarea concurrente. 1?-(2-(2.1||2.2))*-3 3-(2-(2.1||2.2))*-1?
TSEIC_13		Una tarea concurrente con hermanas de tipo indiferente. La concurrente tiene hijas concurrentes	Deben cerrarse en orden inverso: V3-V2-V1 V1: 1?-2*-3 V2(en paralelo): 2.1-2.2 V3(en paralelo): 2.2.1
TSEIC_14		Una tarea concurrente con hermanas de tipo indiferente.	V1:1*-(2-(2.1-2-2.2-2.2.1-2.2.2)*)-3||3-2-(2.1-2-2.2-2.2.1-2.2.2)*)) V2(en paralelo): 1.1

C.4.2.6 Unidades de Test UO12: TSu + TSuor + TSE + tSEI + presentación en ventanas y secciones (TSEIP)

Tabla C.12 Tests del prototipo para TSEIP en Webdiagram 3.0

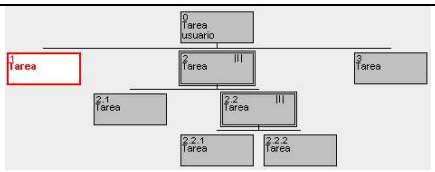
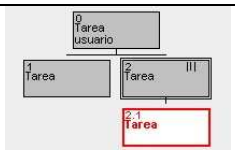
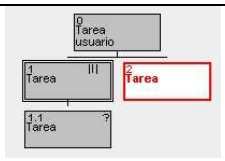
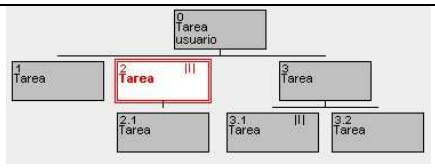
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIP_1		Una tarea principal	**Desactivado**: **Presentación**:. La tarea principal se mostrará siempre en una nueva ventana
TSEIP_2		Una tarea principal con una hija	**Desactivado**: **Presentación**. Una tarea sin hijas no tiene características de presentación para sus hijas.

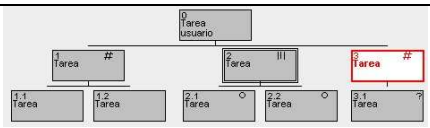
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIP_3		Una tarea con hijas tiene modo de presentación “Nueva Ventana”	Ventana 1: 1 Ventana 2: 1.1- 1.2 Ventana 1: 2 Ventana 3: 2.1
TSEIP_4		Una tarea con hijas con modo de presentación “Nueva sección”	Ventana 1: 1-2 Ventana 1, foco en nueva sección: 2.1-2.2 (desaparece sección) Ventana 1: 3
TSEIP_5		Una tarea con hijas con modo de presentación “Desplegable”	Ventana 1: 1-2 Ventana 1, desplegable: 2.1-2.2 (desaparece desplegable) Ventana 1: 3
TSEIP_6		Una tarea de orden indiferente y modo de presentación “Desplegable”	Ventana 1: 1-2-2.1-2.2 (desparecen 2.1,2.2)-3-4, Combinaciones de orden:1-2-3-4
TSEIP_7		Una tarea secuencial, iterativa y modo de presentación “Desplegable”o “Nueva sección”	Ventana 1: 1-(2-2.1-2.2(desaparecen 2.1,2.2))*-3
TSEIP_8		Una tarea iterativa, con modo de presentación “Desplegable” o “Nueva sección”, con dos hijas opcionales	Ventana 1: 1-(2-2.1?-2.2?)*-3(desaparecen 2.1,2.2)
TSEIP_9		Una tarea iterativa, con modo de presentación “Nueva ventana”, con dos hijas opcionales	Sec.: 1-(2-(2.1?-2.2?, x))*-3 Ventana 1: 1-2 Ventana 2: 2.1?-2.2?,X Ventana 1: 2 - 3

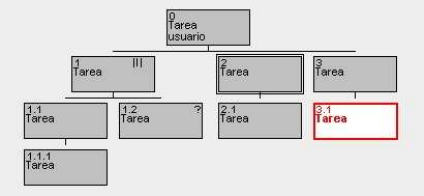
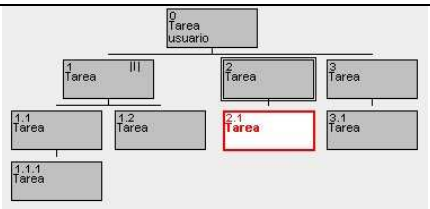
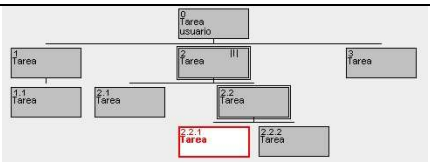
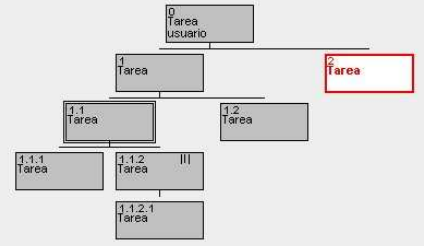
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIP_10		Enlaza tres “Nueva sección”.	Ventana 1, sección 1: 1-2 Sección 2: 2.1 Sección 3(se adapta a la resulución): 2.1.1 Sección 4: 2.1.1.1 Sección 3: 2.1.2 Sección 2: 2.2? Sección 1: 3
TSEIP_11		Enlaza nuevas secciones, hermanas opcionales	V1: 1 S2: 1.1 S3: 1.1.1 S2: 1.2?|| S1:2 V1: 2 V2: 2.1
TSEIP_12		Mezcla tipos de presentaciones.	El foco debe actualizarse en cada sección, y la activación de tareas debe señalar únicamente las posibles en cada momento. Ventana 1: 1,2 Sección 1: 2.1,2.2 Ventana 2: 2.3.1,2.3.2 Sección 2: 2.3.1.1, 2.3.2.2 Ventana 2: 2.3.3 Ventana 1: 3
TSEIP_13		Con presentación en nueva ventana.	V1: 1 V2: 1.1 V3: 1.1.1 V2: 1.2 || X V1: 2 V2: 2.1

C.4.2.7 Unidades de Test UO13: TSu + TSuor + TSE + tSEI + TSEIC + tSEIP + Pruebas de integración (TSEICP)

Tabla C.13 Tests del prototipo para TSEICP en Webdiagram 3.0

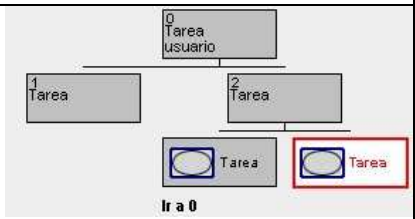
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEICP_1		Varias concurrentes encadenadas en secciones nuevas	Las secciones que tienen tareas concurrentes pendientes únicamente, desaparecen al pasar a otra activación de la sección anterior **Ventana 1**: 2(sección2)*-1-2(sección2)*-3-2(sección2)* **Sección 2**: 2.2(sección 3)*-2.1-2.2 (sección 3)* **Sección 3**: 2.2.1-2.2.2
TSEICP_2		Concurrente con hijas en nueva sección	Las hijas de la tarea concurrente aparecerán en una nueva sección. V1: [2(S2)]*-1-[2(S2)]* S2: 2.1 Permanence activa siempre la concurrente 2
TSEICP_3		Concurrentes con hijas opcionales en nueva sección	V1,S1: 1(S.2)*-2-1(S.2)* Se deshabilita 1.1 al hacer la secuencia 1-2.
TSEICP_4		Concurrentes con hijas en nueva sección, con hermanas no concurrentes	La tarea 2 se ejecutará en una nueva sección. Sin embargo, al hacer la tarea 3 solo estará activa la ventana principal. **Ventana 1**: 2(sección2)*-1-2(sección2)*-3(Ventana 2)-2(sección2)* **Sección 2**: 2.1 **Ventana 1:**: 3.1*-3.2-3.1*

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEICP_5		Concurrentes, con hijas opcionales en nueva sección, y con hermanas no concurrentes	La tarea 2 se ejecutará en una nueva sección. Sin embargo, al hacer la tarea 3 solo estará activa la ventana principal. **Ventana 1**: [2(sección2)]*-1-2(sección2)*-3(Ventana 2)-[2(sección2)]* **Sección 2**: 2.1? **Ventana 1:**: 3.1*-3.2-3.1*
TSEICP_6		Concurrentes, con hijas elección, y hermanas indiferentes.	La tarea 2 se ejecutará en una nueva sección. V1: [2(S2)]*-1-[2(S2)]*-3-[2(S2)]* || [2(S2)]*-3-[2(S2)]*-1-[2(S2)]* S2: 2.1|| 2.2
TSEICP_7		Concurrentes, con hijas elección y hermanas indiferentes. Las tareas indiferentes tienen hijas secuenciales y opcionales	La tarea 2 se ejecutará en una nueva sección. Las tareas 1 y 3 se ejecutarán en la ventana principal V1. V1: [2(S2)]*-1(V1')-[2(S2)]*-3(V1'')-[2(S2)]* || [2(S2)]*-3-[2(S2)]*-1-[2(S2)]* S2: 2.1|| 2.2 V1': 1.1-1.2 V1'':3.1-X
TSEICP_8		Tarea que se efectuará en nueva sección, con hermanas concurrentes	La tarea 2 se ejecutará en una nueva sección. En cualquier momento se puede activar la tarea 3. V1: 3*-1-3*-2(S2)-3* S2: 2.1-2.2

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEICP_9		Tarea que se efectuará en nueva sección, con hermanas concurrentes que tienen hijas opcionales	La tarea 2 se ejecutará en una nueva sección. En cualquier momento se puede activar la tarea 1. Se puede interrumpir en cualquier momento la ejecución de la concurrente 1 para realizar el resto de las tareas. Una vez hecha la tarea 1.1, se puede cerrar la ventana si no se desea hacer la tarea 1.2
TSEICP_10		Tarea que se efectuará en nueva sección, con hermanas concurrentes que tienen hijas secuenciales	La tarea 2 se ejecutará en una nueva sección. En cualquier momento se puede activar la tarea 1. Se puede interrumpir en cualquier momento la ejecución de la concurrente 1 para realizar el resto de las tareas.
TSEICP_11		Una concurrente con una hija en nueva sección	La tarea 2 se ejecutará en una nueva sección en paralelo con la realización en la ventana principal sección 1.
TSEICP_12		Una tarea concurrente en una nueva sección, con hijas.	La tarea 1.1 se ejecutará en una nueva sección. La nueva sección desaparece cuando se han realizado todas las tareas obligatorias de la sección y se pulsa 1.2 en la sección anterior.4

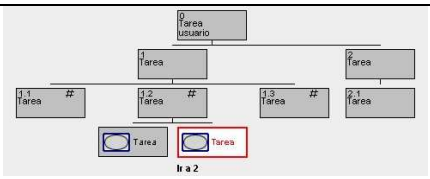
C.4.2.8 Unidades de Test UO14 sobre UO15: Tareas del Sistema con comportamiento (Behavior) (TB)

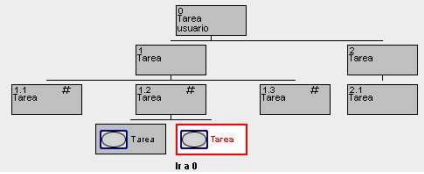
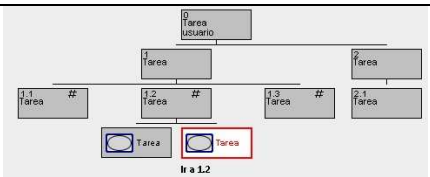
Tabla C.14 Tests del prototipo para TB en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TB_1		Tarea del Sistema	**Desactivado**: Orden , Tipo, Concurrente **Aviso**: No pueden tener hijas
TB_2		Tarea del Sistema con ir a..	• Sólo es posible en hojas • **Mensaje**: "Efectuando tarea X" • Se dirige a tareas objetivo (con hijas)
TB_3		Varias tareas del sistema	• Si una hermana es del Sistema, todas sus hermanas deben ser del Sistema. • Las tareas del Sistema sin etiqueta ir a.. van a la siguiente tarea a efectuar. Si es la última es "fin correcto"

C.4.2.9 Unidades de test UO16: TSu + TSuor + TSE + tSEI + tB (TSEIB)

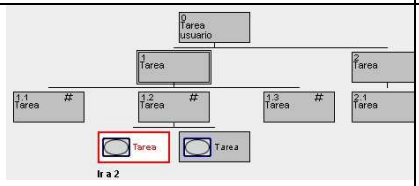
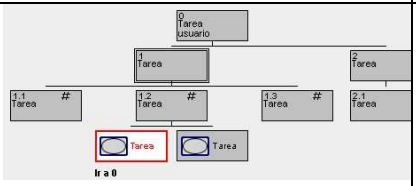
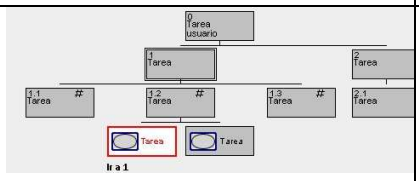
Tabla C.15 Tests del prototipo para TSEIB en Webdiagram 3.0

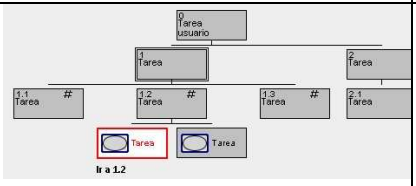
NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIB_1		Tarea del Sistema, hija de indiferente, con etiqueta "ir a 2"	Borra las acciones pendientes, saca mensaje "*Efectuando tarea 2*", y comienza desde la tarea 2

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIB_2		Tarea del Sistema, hija de indiferente, con etiqueta "ir a 0"	Borra las acciones pendientes(hacia abajo), saca mensaje "*Efectuando tarea 0*", y comienza desde la tarea 0
TSEIB_3		Tarea del Sistema, hija de indiferente, con etiqueta "ir a 1.2"	Borra las acciones pendientes, saca mensaje "*Efectuando tarea 1.2*", y comienza desde la tarea 1.2. Tendrá en cuenta el contexto (orden/tipo/concurrencia) de la tarea 1.2

C.4.2.10 Unidades de Test UO17: TSu + TSuor + TSE + TSEI + TSEIP + TB (TSEIBP)

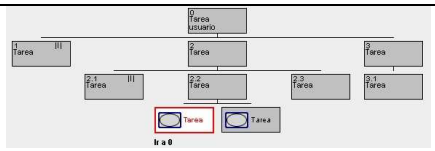
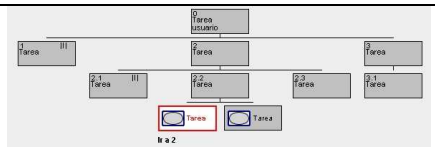
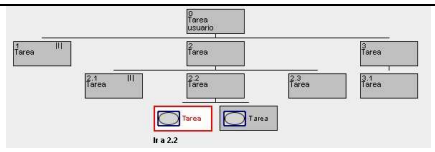
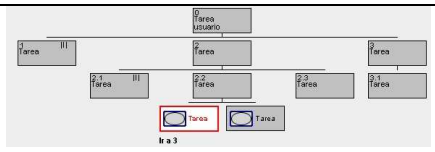
Tabla C.16 Tests del prototipo para TSEIBP en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIBP_1		Nueva sección Tarea del Sistema, hija de indiferente, con etiqueta "ir a 2"	Borra las acciones pendientes y la ventana con nueva sección, saca mensaje "*Efectuando tarea 2*", y comienza desde la tarea 2 en una nueva ventana.
TSEIBP_2		Nueva sección. Tarea del Sistema, hija de indiferente, con etiqueta "ir a 0"	Borra las acciones pendientes , y la ventana con nueva sección, saca el mensaje "*Efectuando tarea 0*", y comienza desde la tarea 0 en una nueva ventana.
TSEIBP_3		Nueva sección. Tarea del Sistema, hija de indiferente, con etiqueta "ir a 1"	Borra las acciones pendientes, saca mensaje "*Efectuando tarea 1*", y comienza desde la tarea 1, en una nueva sección.

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIBP_4		Nueva sección. Tarea del Sistema, hija de indiferente, con etiqueta "ir a 1.2"	Borra la última ventana, saca mensaje "*Efectuando tarea 1.2*", y comienza desde la tarea 1.2, en una nueva ventana. Tiene en cuenta el contexto de la tarea1.2 para saber cuál es la siguiente tarea.

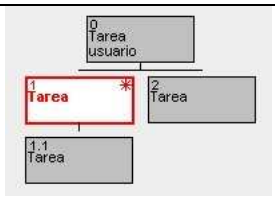
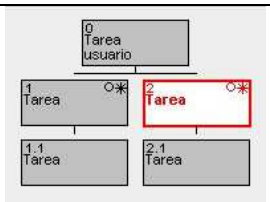
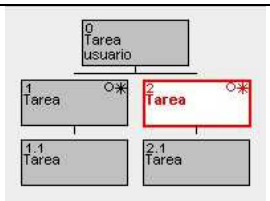
C.4.2.11 Unidades de Test UO15: TSu + TSuor + TSE + tSEI + TSEIC + TSEIP + TSEICP + Pruebas de integración (TSEIB + TSEIBP + Concurrentes) (TSEIBPC)

Tabla C.17 Tests del prototipo para TSEIBC en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TSEIBC_1		Tareas de distinto orden y concurrentes Tarea del Sistema, con etiqueta "ir a 0"	Borra las acciones (y ventanas) pendientes , saca mensaje "*Efectuando tarea0*", y comienza desde la tarea 0 en una nueva ventana.
TSEIBC_2		Tareas de distinto orden y concurrentes Tarea del Sistema, con etiqueta "ir a 2"	Borra acciones y ventanas pendientes desde la tarea 2. Saca el mensaje "*Efectuando tarea 2*", y comienza desde la tarea 2. Tendrá en cuenta el contexto de la tarea 2.
TSEIBC_3		Tareas de distinto orden y concurrentes Tarea del Sistema, con etiqueta "ir a 2.2"	Borra acciones y ventanas pendientes desde la tarea 2. Saca el mensaje "*Efectuando tarea 2.2*", y comienza desde la tarea 2.2. Tendrá en cuenta el contexto de la tarea 2.2.
TSEIBC_4		Tareas de distinto orden y concurrentes Tarea del Sistema, con etiqueta "ir a 3"	Borra las acciones (y ventanas) pendientes , saca mensaje "*Efectuando tarea 3*", y comienza desde la tarea 3 en una nueva ventana

C.4.2.12 Otras pruebas añadidas en los Puntos de Evaluación

Tabla C.18 Tests del prototipo para TF en Webdiagram 3.0

NOMBRE	FIGURA	DESCRIPCIÓN	EFECTO
TF_1		Una tarea repetitiva	Los límites Min y Max se actualizan en cuanto se modifican
TF_2		Orden elección-tipo repetitiva	Al pulsar tipo repetitiva, los límites Min y Max están a sus valores por defecto: Min=0, Max= 999 Luego sacará el aviso: "Min debe ser igual a 1" y pondrá Min= 1.
TF_3		Tipo repetitiva: Orden secuencial Cambia a orden elección	Al convertir una tarea repetitiva de orden secuencial a orden elección, deben cambiar el orden todas las hermanas.
TF_4		Queremos borrar las hijas de una tarea repetitiva	La tarea repetitiva, ya sin hijos, queda como unitaria.

C.5 Estudio de usabilidad de WebDiagram 3.0

Una vez terminada la implementación de WebDiagram 3.0, se realizó una prueba con usuarios para evaluar su usabilidad.

Objetivo

El objetivo de esta prueba fue determinar el grado de satisfacción del usuario final y la usabilidad de la nueva versión de WebDiagram en el desarrollo y prototipado del modelo SE-HCI.

Participantes

En esta evaluación participaron 17 estudiantes del grado de Ingeniería de Software con background en HCI y técnicas de evaluación. Ninguno de ellos había usado con anterioridad el software a evaluar ni ninguna de sus versiones previas.

Material e Instrumentos

Para conducir la evaluación, se utilizaron ordenadores con sistema operativo Windows XP y navegador Firefox, que disponían de Máquina Virtual Java y plugin de Java instalado para Firefox, necesarios para ejecutar la aplicación. La aplicación WebDiagram se instaló localmente, para evitar cualquier problema relacionado con la conexión a la red. Además, se proporcionó a los usuarios el manual de usuario de la aplicación.

Los usuarios tenían que realizar esta tarea: "Modelar un sistema para la compra online de libros". Esta tarea tenía 4 subtareas: identificación del usuario, compra del libro, pago y la visualización de la información de libros. La Tabla C.17 recoge la información que se les entregó con la tarea y subtareas a realizar.

Además. se definieron dos cuestionarios en GoogleDocs, uno para recoger información demográfica, y otro para evaluar la satisfacción en el uso mediante el cuestionario CSUQ (Lewis, 2001). También se anotó el tiempo que los usuarios necesitaron para realizar las tareas.

Tabla C.19 Tarea "Compra de libros on line" para el estudio de usabilidad de WebDiagram 3.0

Tarea:	**"Compra de libros on-line"**
Paso 1:	1. El usuario se identifica: 1.1 Introducir su identificación: Consiste en introducir en cualquier orden y en una nueva sección usuario y password. 1.2 Validar la identificación en una nueva sección: • El Sistema responde "Correcto" e irá al siguiente paso: "Compra de libros". • El Sistema responde "No correcto" e irá al paso 1 "Identificarse"
Paso 2:	2. El usuario realiza la compra de libros. Esta es una acción **repetitiva** todas las veces que desee: 2.1.Se puede **elegir** "Por autor": 2.1.1. Se indica el nombre del autor: • El Sistema responderá "Autor encontrado" e irá tarea 2.1.2. • El Sistema responderá "Autor no encontrado" e irá a la tarea 2. 2.1.2. Se indica el título del libro: • El Sistema responderá "Título encontrado" e irá a "Pago" (tarea 3) • El Sistema responderá "Título no encontrado" e irá a la tarea 2. 2.2. o "Por título": 2.2.1. Se indica el título del libro: • El Sistema responderá "Título encontrado" e irá a "Pago" (tarea 3) • El Sistema responderá "Título no encontrado" e irá a la tarea 2.
Paso 3:	3. Proceder al pago con tarjeta: 3.1. Introducir tipo de tarjeta 3.2. Introducir fecha de caducidad 3.3. Introducir código de seguridad: • El sistema indica "Tarjeta Válida" e irá al inicio • El Sistema indica "Tarjeta No válida" e irá a la tarea 3.

Tarea:	**"Compra de libros on-line"**
Paso 4:	4. De forma concurrente al proceso anterior es posible visualizar los libros que se desee: 4.1. Indicar el título: • El Sistema responderá "Título encontrado" • El Sistema responderá "Título no encontrado"

Procedimiento

Los usuarios se sentaron y encendieron sus ordeenadores. Entonces, los evaluadores les explicaron brevemente (15 minutos) la aplicación WebDiagram y la tarea que debían realizar con ella. La fotografía de la Figura C.2 refleja este momento.

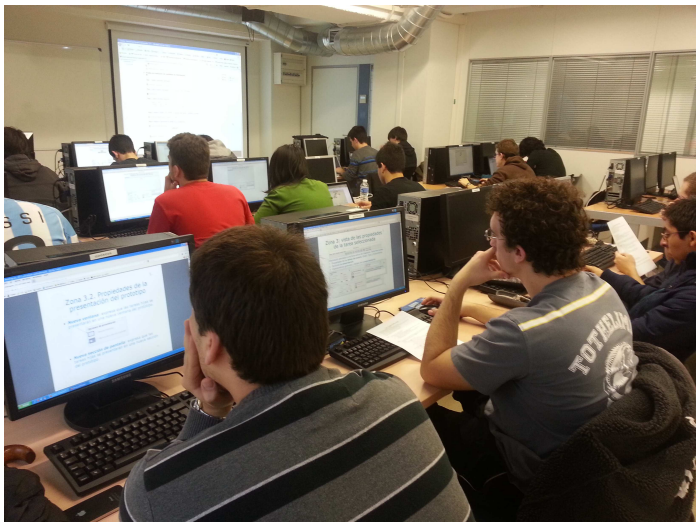


Figura C.2 Momento inicial de la evaluación de usabilidad de WebDiagram 3.0

Después, rellenaron el cuestionario demográfico. Tras esto, recibieron una hoja de papel con instrucciones detalladas sobre la tarea a realizar (Tabla C.19) y comenzaron a realizarla. Este trabajo implicaba desarrollar el modelo SE-HCI de la tarea, visualizar el prototipo generado automáticamente por la herramienta a partir del modelo construído, y verificar que la ejecución del prototipo cumplía todos los aspectos a modelar. Cuando finalizaron la tarea, rellenaron el cuestionario de satisfacción. Finalmente, se llevó a cabo una pequeña entrevista

con los usuarios, para que pudieran expresar su opinión sobre WebDiagram y las dificultades que encontraron.

Resultados

Se midió la usabilidad mediante resultados de efectividad, eficiencia y satisfacción. La eficacia se calculó mediante datos cuantitativos sobre la corrección de los modelos desarrollados por los usuarios. La Tabla C.20 muestra los porcentajes de eficacia para cada subtarea.

Tabla C.20. Porcentajes de eficacia por subtareas en la evaluación de WebDiagram 3.0

Subtarea	**1**	**2**	**3**	**4**	**General**
Porcentaje General	54,41	79,41	79,41	88,24	71,12

USR	**MT 1.1**	**MP 1.1**	**MP 1.2**	**MC 1.2**	**MT 2**	**MT 2.1/2.2**	**MT 2.1.2**	**MC 2**	**MC 3**	**SE-HCI 3.3**	**SE-HCI 4.1**
1	1	0	0	1	0	1	0	1	0	1	1
2	0	0	0	1	1	1	0	1	1	1	1
3	1	0	0	1	1	1	1	1	1	1	1
4	0	1	0	1	1	1	1	1	1	1	1
5	1	0	0	0	1	0	0	0	0	0	0
6	1	1	1	1	1	1	1	1	1	1	1
7	1	0	0	1	1	1	1	1	1	1	1
8	1	0	0	1	1	1	1	1	1	1	1
9	1	1	1	1	1	0	0	1	1	1	1
10	1	0	0	1	1	1	1	1	1	1	1
11	1	0	1	1	1	1	1	1	1	1	1
12	1	0	1	0	1	1	0	0	0	1	1
13	1	0	0	1	1	1	1	1	1	1	1
14	1	0	0	0	0	0	0	0	0	0	0
15	1	0	0	1	1	1	1	1	0	1	1
16	1	0	1	1	1	1	1	1	1	1	1
17	1	0	0	1	1	1	1	1	1	1	1
	88,24	**17,65**	**29,41**	**82,35**	**88,24**	**82,35**	**64,71**	**82,35**	**70,59**	**88,24**	**88,24**

Los resultados se obtuvieron verificando si los usuarios (filas) desarrollaron correctamente todos los aspectos solicitados para cada subtarea. Había un total de 11 aspectos (columnas) que formaban parte del modelo SE-HCI de la tarea general: (Modelo de Tareas de usuario-

MT, Modelo del Prototipo-MP y Modelo de Comportamiento-MC). Por ejemplo, las celdas correspondientes a la columna MT 1.1 indican 1 o 0, si el usuario realizó correctamente o no el Modelo de Tareas de usuario de la subtarea 1.1.

Los resultados de eficacia para la subtarea 1 fueron bastantes bajos: 54,41%, debido a que solo el 17,65% y el 29,41% de los usuarios habían desarrollado correctamente los modelos del prototipo para las subtareas 1.1 y 1.2 respectivamente. En las entrevistas, los usuarios indicaron que no habían entendido el requisito de prototipado en nuevas secciones, por lo que se concluyó que estos resultados bajos no fueron directamente causados por la interfaz de WebDiagram. Si no hubiera sido por esto, los resultados habrían sido ostensiblemente mejores ya que los otros dos modelos para la tarea 1 se completaron en un 88,24% y un 82,35%, respectivamente.

La eficiencia (Tabla B.21) se calculó mediante el tiempo (en MiNutos - MN) que los usuarios necesitaron para realizar la tarea completa. Los usuarios necesitaron de media 32,88 minutos para ello, con una desviación estándar de 8,87 minutos, un mínimo de 17 y un máximo de 52 minutos. Otra medida de eficiencia estaba relacionada con la cantidad de nodos (Número de Nodos – NN) o subtareas que tenían los modelos desarrollados por los usuarios. En este caso, se determinó en 30 el número óptimo de nodos incluído en los modelos. La media fue de 29,35 y la desviación estándar 2,95. También se consideraron como medidas de eficiencia, los porcentajes de los aspectos considerados y alcanzados (Aspectos Alcanzados-AA) en los diferentes tipos de modelos, logrando de media un 88,24 % en el desarrollo de los aspectos solicitados.

Tabla B.21 Porcentajes de eficiencia en la evaluación de WebDiagram 3.0

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**media**
MN	23	25	28	28	32	28	29	26	17	38	39	40	44	40	40	52	30	**32,88**
NN	27	25	29	31	21	30	30	30	30	32	30	30	35	30	30	30	29	**29,35**
AA	6	7	9	9	2	11	9	9	9	9	10	6	9	1	8	10	9	**88,24**

En cuanto a la satisfacción de usuario, el cuestionario CSUQ fue el mostrado en la Tabla B.22. Como se observa en la Tabla C.23, los resultados de satisfacción fueron positivos.

Tabla C.22 Cuestionario CSUQ empleado en la evaluación de satisfacción de WebDiagram 3.0

1. En general, estoy satisfecho con la facilidad de uso de este sistema.
2. El sistema era fácil de usar.
3. He podido completar las tareas y escenarios de forma efectiva usando este sistema.
4. He podido completar las tareas y escenarios de manera rápida usando este sistema.
5. He podido completar eficientemente las tareas y escenarios usando este sistema.
6. Me siento cómodo usando este sistema.
7. Ha sido fácil Aprender a usar este sistema.
8. Creo que podría llegar a ser productivo rápidamente usando este sistema.
9. El sistema ha dado mensajes de error que me indicaron claramente cómo solucionar los problemas.
10. Siempre que he cometido un error usando el sistema, he podido recuperar de manera fácil y rápida.
11. La información (como ayuda en línea, mensajes en pantalla y otros documentación) provista por este sistema resultaba clara.
12. Ha sido sencillo encontrar la información que necesitaba.
13. La información proporcionada por el sistema era sencilla de entender.
14. La información fue efectiva para ayudarme a completar las tareas y escenarios.
15. La organización de la información en las pantallas del sistema era clara.
16. La interfaz de este sistema era agradable.
17. Me ha gustado usar la interfaz de este sistema.
18. Este sistema cuenta con todas las funciones y capacidades que esperaba que tuviera.
19. En general, estoy satisfecho con el sistema.

La escala de las puntuaciones a las sentencias del cuestionario tuvieron un rango de 1 (más positivo) a 7 (más negativo). La media de las respuestas fue de 2,29, con una desviación estándar de 1,21. Hubo cuatro respuestas con un rango inferior a 2; concretamente, las relacionadas con conseguir rápidamente productividad con el sistema (8), la facilidad para encontrar la información necesaria (12), la claridad de la organización de la información en la interfaz (15) y, la disponibilidad de las funciones y capacidades que se esperaba (18).

Únicamente hubo 2 sentencias que recibieron una puntuación superior a 3, y estaban relacionadas con el agrado por la interfaz (16) y la satisfacción de uso (17). En las entrevistas algunos usuarios comentaron que la interfaz de usuario estándar del applet Java no era de su gusto, y sin embargo sí lo era la interfaz de la aplicación. Las demás sentencias fueron puntuadas de un 2 a un 3. La sentencia 19, relacionada con la satisfacción general con el sistema recibió una media de 2 sobre 7.

Tabla C.23 Resultados del cuestionario de Satisfacción empleado en WebDiagram 3.0

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\overline{x}$	2,41	2,00	2,00	2,35	2,41	2,29	2,18	**1,88**	2,80	2,38
s	**0,87**	1,06	1,00	1,37	1,12	1,31	1,29	**0,99**	1,84	**0,99**
σ^2	**0,76**	1,13	1,00	1,87	1,26	1,72	1,65	**0,99**	**3,40**	**0,98**

	11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**MEDIA**
$\overline{x}$	2,13	**1,94**	2,18	2,41	**1,88**	3,29	3,06	**1,94**	2,00	**2,29**
s	1,32	1,09	1,33	1,00	1,05	1,86	1,20	1,30	1,00	**1,21**
σ^2	1,73	1,18	1,78	1,01	1,11	3,47	1,43	1,68	1,00	**1,53**

Los comentarios de los usuarios incluídos en el cuestionario se completaron con entrevistas cortas, y permitieron recoger información interesante, como que algunas restricciones del dominio de los modelos no era conocido por los usuarios, aunque se explicaba en el manual de usuario.

ACRÓNIMOS

AAIML	Alternate Abstract Interface Markup Language
AUI	Abstract User Interface-Interfaz de Usuario Abstracta
AUIML	Abstract User Interface Markup Language
AMDD	Agile Model-Driven Development-Desarrollo Dirigido por Modelos Ágiles
CBSE	Component-Based Software Engineering-Ing. de Sw Basada en Componentes
CSUQ	Computer System Usability Questionnaire
CTT	Concur Task Trees
CUI:	Concret User Interface-Interfaz de Usuario Concreta
DCU	Diseño Centrado en el Usuario
DII	Desarrollo iterativo e incremental
DA	Developmental Activity-Actividad de Desarrollo
DA-k(i)	Developmental Activity-Actividad de Desarrollo k (1<=k<=3) para un UOi
DDM	Desarrollo de software Dirigido por Modelos - Model Driven Development
DSML	Domain-specific modeling languages-Lenguajes de modelado específicos del dominio
EBP	Elementary Business Processes-Procesos de Negocio Elementares.
EMF	Eclipse Modeling Framework
FSM	Finite State Machine-Máquina de Estados Finita
GME	Generic Modeling Environment
HCI	Human-Computer Interaction-Interacción Persona-Computador

IA	Integration Activity - Actividad de Integración
IA	Incremental Integration Activity - Actividad de Integración Incremental
IA-k(i)	Integration Activity-Actividad de Integración k (1<=k<=3) para un UOi
IA-k(i)	Incremental Integration Activity - Actividad de Integración Incremental k (1<=k<=3) para un UOi
IS	Ingeniería de Software
IU	Interfaz de Usuario
IngUS	Ingeniería de la Usabilidad
M-k	Modelo i creado directa o indirectamente por la actividad k (1<=k<=3)
M-k(i)	Modelo i creado directa o indirectamente por la actividad k (1<=k<=3) para el UOi
MA	Metodologías Ágiles
MDA	Model Driven Architecture-Arquitectura Dirigida por Modelos
MDE	Model Driven Engineering
MPIu+a	Modelo de Proceso de la Ingeniería de la usabilidad y de la accesibilidad
PIM	Platform-Independent Model-Modelo Independiente de la Plataforma
PSM	Platform-Specific Model-Modelo Específico para la Plataforma
RDA	Regla de Distribución de Actividades
RCI	Regla de Creación Indirecta
RDI	Regla de Derivación Indirecta
RGD	Regla de Gestión Directa
RGM	Regla de Gestión Manual
RPA	Regla de Planificación de Actividades
SE-HCI	Semantically Enriched Human-Computer Interaction (model) - (Modelo) de Interacción Hombre-Computador Enriquecido Semánticamente

SRS Severity Ranking Scale

TDD Test-Driven Development

UIML User Interface Markup Language

UML Unified Modeling Language- Lenguaje Unificado de Modelado

UsIXML User Interface eXtended Markup Language

UO User Objective-Objetivo de usuario

XIML eXtensible Interface Markup Language

XML eXtensible Markup Language

XUL XML-based User Interface Language

DEFINICIONES

Big up-front design: Se refiere a aquellas metodologías que requieren realizar un esfuerzo de diseño (y de especificación de los requisitos) total, antes de proceder a la implementación.

Casos de Uso: Son modelos de sistemas, esto es, secuencias de acciones, incluyendo secuencias alternativas y secuencias de error, que un sistema, subsistema o clase puede realizar interactuando con actores externos (Rumbaugh et al., 2004)

Desarrollador: Es un profesional del software implicado en la creación del producto.

Diagram: Es una herramienta, local y de escritorio, creada para ayudar a los analistas y diseñadores a registrar y evaluar el modelo SE-HCI de las aplicaciones en desarrollo, de una manera sencilla.

Diseño centrado en el usuario : Es un proceso de desarrollo fuertemente estructurado, en el que el foco está en entender las necesidades y los objetivos del usuario del producto.

Diseño centrado en el uso: Es un enfoque sistemática, dirigida por modelos, para mejorar la usabilidad del producto, en el que el foco está en el uso del producto (Constantine and Lockwood, 1999).

Escenario: Es una historia sobre gente que realiza una actividad en una aplicación (Rosson and Carroll, 2001).

Escenario de UO: Es una historia hipotética diseñada por el evaluador a realizar por el usuario final a través de la interfaz propuesta para evaluar un UO a través de esta situación dada (Losada et al., 2013b)**.**

Entrevista estructurada: Una entrevista en la que las preguntas están predeterminadas y no hay margen para explorar las opiniones individuales (Preece et al., 1994).

Estándar: Es un requisito, regla o recomendación basado en principios probados y en la práctica, acordado por un grupo de profesionales oficialmente autorizados en un ámbito local, nacional o internacional (Smith, 1996)

Feature: Unidad de progreso en la metodología Feature-Driven Development (Palmer and Felsing, 2002)

Heavyweight: Califican a un conjunto de metodologías basadas en el desarrollo en series secuenciales de pasos tales como la definición de requisitos, el diseño, la implementación y las pruebas; estos métodos se basan, por lo tanto, en una fuerte docuementación.

High-Fidelity prototype: Prototipo cercano al producto final, con muchos detalles y funcionalidad, lo que permite examinar cuestiones de usabilidad y deducir su comportamiento ("Usability First: Usability in Website and Software Design," 2002-13).

Ingeniería de Software: Aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software; es decir, la aplicación de ingeniería al software.

Ingeniería de la Usabilidad: Disciplina que provee métodos estructurados para conseguir usabilidad en el diseño de la interfaz de usuario durante el desarrollo de un producto (Mayhew, 1999).

Interacción persona-ordenador - Human-computer interaction: Disciplina relacionada con el diseño, evaluación e implementación de sistema informáticos interactivos para el uso de seres humanos, y con el estudio de los fenómenos más importantes con los cuales está relacionada (Granollers i Saltiveri et al., 2005)**.**

Interfaz de usuario: Conjunto de aspectos del sistema con los que el usuario entra en contaco, físicamente, perceptivamente o conceptualmente (Granollers i Saltiveri et al., 2005).

InterMod: Metodología ágil y centrada en el usuario, basada en modelos para el desarrollo de aplicaciones interactivas.

Iteración (en una metodología ágil): Es un mini-proyecto autocontenido que se compone de actividades como análisis de requisitos, diseño, programación y test. El objetivo al final de

una iteración es una "iteration release". Esto es, un sistema parcialmente completo estable, integrado y probado (Larman, 2003).

Lightweight Califican a un conjunto de metodologías basadas en los principios ágiles

Low-fidelity prototype: Prototipo incompleto que tiene algunas casracterísticas del producto final pero es simple, normalmente para realizar evaluaciones con poco detalle y generar el producto rápidamente ("Usability First: Usability in Website and Software Design," 2002-13).

Modelo: Es una abstracción sobre algún producto software; por ejemplo, la especificación de requisitos, diseño, código, test, grafo de flujo (Hailpern and Tarr, 2006)

Patrón de diseño: Representa una solución a un problema enun contexto dado.

Plan-driven methods: Se refiere a aquéllos métodos que se dirigen por un plan que requiere en primer lugar una definición de los requisitos, una fuerte documentación y planes detallados de trabajo.

Prototipo: Implementación incompleta y experimental, utilizada para probar las ideas de diseño (Preece et al., 1994).

Sprint: Unidad de trabajo en Scrum, necesaria para alcanzar un requisito.

Sistema Es el producto, que incluye a menudo el software, bajo desarrollo (Ambler, 2008).

Stakeholder: Es cualquier persona implicada en la creación u operación del producto (Ambler, 2008).

Storyboard: Técnica de prototipado de baja fidelidad consistente en una secuencia de ilustraciones acompañadas con narraciones del escenario, tomada del cómic o el cine (Cooper et al., 2007).

T-test: Técnica de evaluación que consiste en comparar la diferencia en las mediciones estadísticas entre varios grupos (o tareas en el caso de mediciones de éxito en la consecución de tareas de usuario) (Tullis and Albert, 2008).

Test de aceptación: Técnica de validación cuyo objetivo es determinar si un sistema satisface sus criterios de aceptación, y permitir a los implicados determinar si se acepta el producto (Ambler, 2008).

Test de usabilidad Es un método por el que a los usuarios de un sistema se les pide realizar ciertas tareas para medir la facilidad de uso del sistema, el tiempo por tarea, y la percepción del usuario en la experiencia (Ambler, 2008).

Test de usuario : Actividades de validación, que incluyen los tests de aceptación y usabilidad, en el que intervienen los implicados (Ambler, 2008).

Thinking aloud protocol Protocolo oral en el que un usuario comenta en voz alta lo que piensa mientras realiza una tarea.(Preece et al., 1994).

Time-boxed: Se refiere a las iteraciones que se fijan en duración. Si no puede efectuarse lo previsto en ese tiempo, se reduce el alcance (se colocan las peticiones menos prioritarias para el final). El sistema parcial siempre debe finalizar en un estado estable y evaluable en la iteración planificada (Larman, 2003).

User story: Declaración breve de intenciones, que describe algo que el sistema debe hacer para el usuario (Leffingwell, 2011).

Usuario o usuario final: Persona que trabajará con el producto que se está construyendo (Ambler, 2008).

WebDiagram: Herramienta web creada para ayudar a los analistas y diseñadores a registrar y evaluar el modelo SE-HCI de las aplicaciones en desarrollo, de una manera sencilla.

REFERENCIAS

Abbas, N., Gravell, A.M., Wills, G.B., 2008. Historical Roots of Agile Methods: Where Did "Agile Thinking" Come From? In: Abrahamsson, P., Baskerville, R., Conboy, K., Fitzgerald, B., Morgan, L., Wang, X. (Eds.), Agile Processes in Software Engineering and Extreme Programming, Lecture Notes on Business Information Processing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 94–103.

Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J., 2003. New directions on agile methods: a comparative analysis. In: Proceedings of the 25th International Conference on Software Engineering, ICSE '03. IEEE Computer Society, Washington, DC, USA, pp. 244–254.

Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E., 1999. UIML: an appliance-independent XML user interface language. Comput. Netw. 31, 1695–1708.

Agile Alliance: The Twelve Principles of Agile Software [WWW Document], 2001. URL http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/ (accedido el 14.12.13).

Ambler, S., 2002. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, 1st ed. Wiley.

Ambler, S., n.d. Introduction to Test Driven Development (TDD) [WWW Document]. URL http://www.agiledata.org/essays/tdd.html (accedido el 8.1.14).

Ambler, S.W., 2004. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press.

Ambler, S.W., 2008. Tailoring Usability into Agile Software Development Projects. In: Law, E.L.-C., Hvannberg, E.T., FRSA, G.C.M., PGCE,PhD, CITP, FBCS (Eds.), Maturing Usability, Human-Computer Interaction Series. Springer London, pp. 75–95.

Ambler, S.W., n.d. An Introduction to Agile Modeling [WWW Document]. URL http://www.agilemodeling.com/essays/introductionToAM.htm (accedido el 8.1.14).

Annett, J., Duncan, K.D., 1967. Task Analysis and Training Design. Occup. Psych. 12, 211–221.

Anton, A.I., 1996. Goal-based requirements analysis. In: , Proceedings of the Second International Conference on Requirements Engineering, 1996. Presented at the , Proceedings of the Second International Conference on Requirements Engineering, 1996, pp. 136–144.

Basili, V.R., Turner, A.J., 1975. Iterative Enhancement: A Practical Technique for Software Development. IEEE Trans. Software Eng. 1, 390–396.

Beck, K., 1999. Extreme Programming Explained: Embrace Change, US ed. ed. Addison Wesley.

Beck, K., 2002. Test Driven Development: By Example, 1st ed. Addison-Wesley Professional.

Beck, K., Andres, C., 2004. Extreme Programming Explained: Embrace Change, 2nd ed. Addison-Wesley Professional.

Bell, A.E., 2004. Death by UML Fever. Queue 2, 72–80.

Berti, S., Correani, F., Mori, G., Paternò, F., Santoro, C., 2004a. TERESA: a transformation-based environment for designing and developing multi-device interfaces. In: CHI '04 Extended Abstracts on Human Factors in Computing Systems, CHI EA '04. ACM, New York, NY, USA, pp. 793–794.

Berti, S., Correani, F., Paternò, F., Santoro, C., 2004b. The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction. In: Leveles, Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages. pp. 103–110.

Bertini, E., Catarci, T., Dix, A., Gabrielli, S., Kimani, S., Santucci, G., 2009. Appropriating Heuristic Evaluation for Mobile Computing. International Journal of Mobile Human Computer Interaction 1, 20–41.

Bias, R.G., 1994. Usability inspection methods. In: Nielsen, J., Mack, R.L. (Eds.), John Wiley & Sons, Inc., New York, NY, USA, pp. 63–76.

Blomkvist, S., 2005. Towards a Model for Bridging Agile Development and User-Centered Design. In: Seffah, A., Gulliksen, J., Desmarais, M.C. (Eds.), Human-Centered Software Engineering — Integrating Usability in the Software Development Lifecycle, Human-Computer Interaction Series. Springer Netherlands, pp. 219–244.

Boehm, B.W., 1988. A spiral model of software development and enhancement. Computer 21, 61–72.

Boehm, B.W., Turner, R., 2003. Balancing agility and discipline: a guide for the perplexed. Addison-Wesley Professional.

Booch, G., 2006. El lenguaje unificado de modelado 2/e, 2nd ed. ADDISON WESLEY.

Bradac, M.G., Perry, D.E., Votta, L.G., 1993. Prototyping a process monitoring experiment. In: Proceedings of the 15th International Conference on Software Engineering, ICSE '93. Presented at the ICSE 93, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 155–165.

Brooke, J., 1996. SUS: A quick and dirty usability scale. Usability evaluation in industry.

Brooks,Jr., F.P., 1995. The Mythical Man Month and Other Essays on Software Engineering, 2nd ed. Addison Wesley.

Chow, T., Cao, D.-B., 2008. A survey study of critical success factors in agile software projects. Journal of Systems and Software 81, 961–971.

Coad, P., Luca, J. de, Lefebvre, E., 1999. Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall PTR.

Cockburn, A., 2004. Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams, 1st ed. Addison-Wesley Professional.

Cockburn, A., Highsmith, J., 2001. Agile Software Development: The People Factor. Software development magazine 131–133.

Cockburn, A, 1997. Structuring Use Cases with Goals Journal of Object-Oriented Programming, 35.40, 56–62.

Connell, J.L., Shafer, L., 1989. Structured Rapid Prototyping: An Evolutionary Approach to Software Development. Yourdon.

Constantine, L.L., 1995. Essential modeling: use cases for user interfaces. interactions 2, 34–46.

Constantine, L.L., 2002. Beyond User-Centered Design and User Experience: Designing for User Performance. Cutter IT Journal 17.

Constantine, L.L., Lockwood, L.A.D., 1999. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design, 1st ed. Addison-Wesley Professional.

Constantine, L.L., Lockwood, L.A.D., 2002. Usage-centered engineering for Web applications. IEEE Software 19, 42 –50.

Cooper, A., Reimann, R., Cronin, D., 2007. About Face 3: The Essentials of Interaction Design, 3rd ed. Wiley.

Cotterman, W.W., Professionals, I. for C. of C., Society, I.C., Machinery, A. for C., 1981. Systems analysis and design: a foundation for the 1980's. North-Holland.

Dayton, D., Barnum, C., 2009. The Impact of Agile on User-centered Design: Technical Communication 56, 219–234.

Debrauwer, L., Heyde, F.V. der, 2005. UML 2: iniciación, ejemplos y ejercicios corregidos. Ediciones ENI.

Dijkstra, E., 1979. Go to statement considered harmful. Yourdon Press, Upper Saddle River, NJ, USA, pp. 27–33.

Dix, A., Finlay, J., Abowd, G.D., Beale, R., 2003. Human Computer Interaction, 3rd ed. Prentice Hall.

Dumas, J.S., Redish, J., 1993. A practical guide to usability testing. Ablex Pub. Corp.

Dwayne, P., 1997. How to Make a V-Shaped Spiral -- Vice Versa. American Programmer 10, 23–26.

Eckstein, J., 2004. Agile Software Development in the Large: Diving Into the Deep. Dorset House.

Extreme Programming vs. Interaction Design [WWW Document], 2003.. Extreme Programming vs. Interaction Design. URL http://37signals.com/svn/archives2/extreme_programming_vs_interaction_design.php (accedido el 9.1.14).

Federoff, M., Villamor, C., Miller, L., Patton, J., Rosenstein, A., Baxter, K., Kelkar, K., 2008. Extreme usability: adapting research approaches for agile development. In: CHI '08 Extended Abstracts on Human Factors in Computing Systems, CHI EA '08. ACM, New York, NY, USA, pp. 2269–2272.

Ferre, X., Juristo, N., Moreno, A., 2004. Improving software engineering practice with HCI aspects. In: Software Engineering Research and Applications. pp. 349–363.

Finstad, K., 2010. The Usability Metric for User Experience. Interacting with Computers 22, 323–327.

Freiberg, M., Baumeister, J., 2008. A Survey on Usability Evaluation Techniques and an Analysis of their actual Application (No. 450), Research Report Series. Institute of Computer Science, University of Würzburg, Germany.

Ghezzi, C., Jazayeri, M., Mandrioli, D., 2002. Fundamentals of Software Engineering, 2nd ed. Prentice Hall.

Gilb, T., 1977. Software metrics. Winthrop Publishers.

Gilb, T., 1985. Evolutionary Delivery versus the "waterfall model". SIGSOFT Softw. Eng. Notes 10, 49–61.

Gilb, T., Finzi, S., 1988. Principles of software engineering management. Addison-Wesley Pub. Co.

Gould, J.D., Lewis, C., 1985. Designing for usability: key principles and what designers think. Commun. ACM 28, 300–311.

Granollers i Saltiveri, T., Lorés Vidal, J., Cañas Delgado, J.J., 2005. Diseño de sistemas interactivos centrados en el usuario. Editorial UOC.

Greenberg, S., 1996. Teaching human computer interaction to programmers. interactions 3, 62–76.

Greenfield, J., 2004. Software factories: assembling applications with patterns, models, frameworks, and tools. Wiley Pub., Indianapolis, IN.

Hailpern, B., Tarr, P., 2006. Model-driven development: the good, the bad, and the ugly. IBM Syst. J. 45, 451–461.

Hanna, M., 1995. Farewell to waterfalls? Software Magazine 15, 38–ff.

Harmelen, M.V., 2001. Object Modeling and User Interface Design: Designing Interactive Systems, 1st ed. Addison-Wesley.

Hartson, H.R., Siochi, A.C., Hix, D., 1990. The UAN: a user-oriented representation for direct manipulation interface designs. ACM Trans. Inf. Syst. 8, 181–203.

Hellmann, T.D., Hosseini-Khayat, A., Maurer, F., 2010. Agile Interaction Design and Test-Driven Development of User Interfaces – A Literature Review. In: Dingsøyr, T., Dybå, T., Moe, N.B. (Eds.), Agile Software Development. Springer Berlin Heidelberg, pp. 185–201.

Hix, D., Hartson, H.R., 1993. Developing User Interfaces: Ensuring Usability Through Product & Process, 1st ed. Wiley.

Hussain, Z., Slany, W., Holzinger, A., 2009. Current State of Agile User-Centered Design: A Survey. In: Holzinger, A., Miesenberger, K. (Eds.), HCI and Usability for E-Inclusion. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 416–427.

IEEE SA - 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology [WWW Document], 1990. URL http://standards.ieee.org/findstds/standard/610.12-1990.html (accedido el 9.1.14).

InfoQ: Craig Larman on the Challenges of Scaling Scrum to Large Organizations [WWW Document], 2011. URL http://www.infoq.com/interviews/larman-scrum-large-organizations (accedido el 9.1.14).

ISO 13407:1999 - Human-centred design processes for interactive systems [WWW Document], 1999. URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=21197 (accedido el 9.1.14).

ISO 9000:2005 - Quality management systems -- Fundamentals and vocabulary [WWW Document], 2005. URL http://www.iso.org/iso/catalogue_detail?csnumber=42180 (accedido el 9.1.14).

ISO/IEC 25000:2005 - Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE [WWW Document], 2005 URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35683 (accedido el 9.1.14).

ISO/IEC 25010:2011 [WWW Document], 2011. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35733 (accedido el 9.1.14).

Jackson, M.A., 1975. Principles of Program Design. Acad. Press.

Jacobson, I., 1992. Object Oriented Software Engineering: A Use Case Driven Approach, Revised Printing. ed. Addison-Wesley Professional.

Jarzombek, S.J., 1999. Proceedings of the 5th Annual Joint Aerospace Weapons Systems Suppor, Sensors, and Simulation Symposium. Presented at the JAWS S3, Gov't Printing Office Press.

Jiang, L., Eberlein, A., 2009. An analysis of the history of classical software development and agile development. In: Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics. Presented at the International Conference on Systems, Man, and Cybernetics, IEEE, San Antonio, TX, USA, pp. 3733–3738.

Johnson, H., Johnson, P., 1991. Task knowledge structures: Psychological basis and integration into system design. Acta Psychologica 78, 3–26.

Johnson, J., 2002. Keynote speech at XP 2002. Presented at the XP 2002, Sardinia, Italia.

Kaikkonen, A., Kekalainen, A., Cankar, M., Kallio, T., Kankainen, A., 2005. Usability Testing of Mobile Applications: A Comparison between Laboratory and Field Testing. JUS 1, 4–17.

Kangas, E., Kinnunen, T., 2005. Applying user-centered design to mobile application development. Commun. ACM 48, 55–59.

Kleppe, A.G., Warmer, J.B., Bast, W., 2003. Mda Explained, the Model Driven Architecture: Practice and Promise. Addison-Wesley Professional.

Larman, C., 2003. Agile and Iterative Development: A Manager's Guide, 1st ed. Addison-Wesley Professional.

Larman, C., Basili, V.R., 2003. Iterative and Incremental Development: A Brief History. Computer 36, 47–56.

Larman, C., Vodde, B., 2008. Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, 1st ed. Addison-Wesley Professional.

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P., 2001. The Generic Modeling Environment. In: Workshop on Intelligent Signal Processing.

Lederer, A.L., Prasad, J., 1992. Nine management guidelines for better cost estimating. Commun. ACM 35, 51–59.

Leffingwell, D., 2011. Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Addison-Wesley, Upper Saddle River, NJ.

Lewis, C., 1982. Using the "thinking Aloud" Method in Cognitive Interface Design. IBM T.J. Watson Research Center.

Lewis, C., 1982. Using the thinking-aloud method in cognitive interface design. IBM Research Report RC 9265, Yorktown Heights, NY.

Lewis, C., Polson, P.G., Wharton, C., Rieman, J., 1990. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90. ACM, New York, NY, USA, pp. 235–242.

Lewis, J.R., 2001. Psychometric Evaluation of the CSUQ Using Data from Five Years of Usability Studies (TR 29.3418), IBM Voice Systems. IBM, West Palm Beach, Florida.

Lewis, J.R., 2002. Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies. International Journal of Human-Computer Interaction 14, 463–488.

Lewis, J.R., n.d. IBM computer usability satisfaction questionnaires:Psychonometric evaluation and instructions for use. International Journal of Human-Computer Interaction 7, 57–78.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V., 2004. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. Springer-Verlag, pp. 11–13.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V., 2005. USIXML: A Language Supporting Multi-path Development of User Interfaces. In: Bastide, R., Palanque, P., Roth, J. (Eds.), Engineering Human Computer Interaction and Interactive Systems, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 200–220.

limitedwipsociety [WWW Document], 2013. URL http://limitedwipsociety.ning.com/ (accedido el 9.1.14).

Losada, B., Urretavizcaya, M., Fernández De Castro, I., 2009a. Efficient Building of Interactive Applications Guided by Requirements Models. In: Proceedings of the 9th International Conference on Web Engineering, ICWE '9. Springer-Verlag, Berlin, Heidelberg, pp. 481–484.

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2009b. The InterMod Methodology: An Interface Engineering Process Linked with Software Engineering Stages. In: New Trends on Human-Computer Interaction: Research, Development, New Tools . Springer, pp. 53–63.

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2009c. Requirements analysis as a guide for the process of organising and developing an interactive application. Presented at the Int. Ass.Development of the Information Society, Barcelona, pp. 412–416.

Losada, B., Urretavizcaya, M., Fernández de Castro, I., 2010. N_InterMod: Una Propuesta de notación de Diálogo enriquecida para el desarrollo ágil de aplicaciones interactivas. Presented at the Interacción 2010.

Losada, B., Urretavizcaya, M., de Castro, I.F., 2011a. An integrated approach to develop interactive software. In: Proceedings of the 13th IFIP TC 13 International Conference on Human-Computer Interaction - Volume Part IV, INTERACT'11. Springer-Verlag, Berlin, Heidelberg, pp. 470–474.

Losada, B., Urretavizcaya, M., Fernández de Castro, I., 2011b. Agile Development of Interactive Software by means of User Objectives. Presented at the ICSEA 2011, The Sixth International Conference on Software Engineering Advances, pp. 539–545.

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2011c. User Objectives as a guide to develop an interactive application. In: Actas Del XII Congreso Internacional de Interacción Persona-Ordenador, INTERACCION '11. Presented at the Interacción 2011, Garceta, Lisboa, pp. 77–86.

Losada, B., Urretavizcaya, M., López-Gil, J.-M., Fernández-Castro, I., 2012. Combining InterMod agile methodology with usability engineering in a mobile application development. In: Proceedings of the 13th International Conference on Interaccion Persona-Ordenador, INTERACCION '12. ACM, New York, NY, USA, pp. 39:1–39:8.

Losada, B., Urretavizcaya, M., Fernández-Castro, I., 2013a. A guide to agile development of interactive software with a "User Objectives"-driven methodology. Science of Computer Programming 78, 2268–2281.

Losada, B., Urretavizcaya, M., López-Gil, J.-M., Fernández De Castro, I., 2013b. Applying Usability Engineering in InterMod Agile DevelopmentMethodology. A Case Study in a Mobile Application. Journal of Universal Computer Science 19, 1046–1065.

Luna, E.R., Panach, J.I., Grigera, J., Rossi, G., PASTOR, O., 2010. Incorporating Usability Requirements in a Test/Model-driven Web Engineering Approach. J. Web Eng. 9, 132–156.

MacCormack, A., 2001. Product Development Practices that Work: How Internet Companies Build Software. MIT Sloan Management Review 75–84.

Maciaszek, L., Liong, B.L., 2004. Practical Software Engineering: A Case-Study Approach. Addison Wesley.

Madden, W.A., Rone, K.Y., 1984. Design, development, integration: space shuttle primary flight software system. Commun. ACM 27, 914–925.

Manchón, E., 2002. Resultados encuesta perfil profesional AI y Usabilidad Iberoaméricanos: España, Portugal y Latinoamérica. [WWW Document]. URL http://www.oocities.org/es/foro_perfil/paginas/ii/iiaiw03.html (accedido el 9.1.14).

Manifesto for Agile Software Development [WWW Document], 2001. URL http://agilemanifesto.org/ (accedido el 9.1.14).

Markopoulos, P., Marijnissen, P., 2000. UML as a representation for Interaction Design. In: IN PROCEEDINGS OF OZCHI 2000. pp. 240–249.

Mayhew, D.J., 1999. The Usability Engineering Lifecycle: A Practioner's Handbook for User Interface Design. Morgan Kaufmann.

McConnell, S., 1996. Rapid Development: Taming Wild Software Schedules, 1st ed. Microsoft Press.

McCracken, D.D., Jackson, M.A., 1982. Life cycle concept considered harmful. SIGSOFT Softw. Eng. Notes 7, 29–32.

MDA [WWW Document], n.d. URL http://www.omg.org/mda/ (accedido el 9.1.14).

Memmel, T., Gundelsweiler, F., Reiterer, H., 2007. Agile human-centered software engineering. In: Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but Not as We Know It - Volume 1, BCS-HCI '07. British Computer Society, Swinton, UK, UK, pp. 167–175.

MetaCase - Domain-Specific Modeling with MetaEdit+ [WWW Document], n.d. URL http://www.metacase.com/ (accedido el 9.1.14).

Michael Gellner, P.F., 2004. Extreme Evaluations -- Lightweight Evaluations for Software.

Miller, L., 2006. Interaction Designers and Agile Development: A Partnership. In: Procedings of UPA 2006. Denver/Broomfield: Usability Professionals'Association.

Mills, H.D., 1971. Top-Down Programming in Large Systems. In: Debugging Techniques in Large Systems. Prentice-Hall, pp. 41–55.

Mills, H.D., 1976. Software Development. IEEE Transactions on Software Engineering 2, 265–273.

Mills, H.D., Dyer, M., Linger, R.C., 1987. Cleanroom Software Engineering. IEEE Software 19–25.

Myers, B.A., Rosson, M.B., 1992. Survey on user interface programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92. ACM, New York, NY, USA, pp. 195–202.

Nielsen, J., 1993. Usability Engineering, 1st ed. Morgan Kaufmann.

Nielsen, J., 2011. Mobile Website and Application Usability | Nielsen Norman Group Report [WWW Document]. URL http://www.nngroup.com/reports/mobile-website-and-application-usability/ (accedido el 9.1.14).

Nielsen, J., Mack, R.L., 1994. Usability inspection methods. Wiley.

Nielsen, J., Molich, R., 1990. Heuristic evaluation of user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '90. ACM, New York, NY, USA, pp. 249–256.

Norman, D.A., 2006. Logic versus usage: the case for activity-centered design. interactions 13, 45–ff.

Nunes, N.J., Cunha, J.F. e, 2001. Wisdom: a UML based architecture for interactive systems. In: Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems, DSV-IS'00. Springer-Verlag, Berlin, Heidelberg, pp. 191–205.

Nunes, N.J., e Cunha, J.F., 2000. Towards a UML profile for interaction design: the wisdom approach. In: Proceedings of the 3rd International Conference on The Unified Modeling Language: Advancing the Standard, UML'00. Springer-Verlag, Berlin, Heidelberg, pp. 101–116.

Nunes, N.J., e Cunha, J.F., 2001. Object modeling and user interface design. In: Van Harmelen, M. (Ed.), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, pp. 197–243.

OASIS User Interface Markup Language (UIML) TC [WWW Document], n.d. URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml (accedido el 9.1.14).

Object Management Group - UML [WWW Document], n.d. URL http://www.uml.org/ (accedido el 9.1.14).

Obrenovic, Z., Starcevic, D., Selic, B., 2004. A model-driven approach to content repurposing. IEEE MultiMedia 11, 62–71.

Olayemi Olagbegi, H.H., 2008. Agile Development: Do advantages outweigh short comings? 46–52.

Palmer, S.R., Felsing, J.M., 2002. A Practical Guide to Feature-Driven Development, 1st ed. Prentice Hall.

Parnas, D.L., Clements, P.C., 1986. A rational design process: How and why to fake it. IEEE Trans. Softw. Eng. 12, 251–257.

Paterno, F., 2000. Model-Based Design and Evaluation of Interactive Applications, 1st Edition. ed. Springer.

Paternò, F., Russino, A., Santoro, C., 2007. Remote evaluation of mobile applications. In: Proceedings of the 6th International Conference on Task Models and Diagrams for User Interface Design, TAMODIA'07. Springer-Verlag, Berlin, Heidelberg, pp. 155–169.

Paterno, F., Santoro, C., Mantyjarvi, J., Mori, G., Sansone, S., 2008. Authoring pervasive multimodal user interfaces. Int. J. Web Eng. Technol. 4, 235–261.

Paternò, F., Santoro, C., Spano, L.D., 2011. Engineering the authoring of usable service front ends. Journal of Systems and Software 84, 1806–1822.

Paterno', F., Santoro, C., Spano, L.D., 2009. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16, 19:1–19:30.

Pihneiro da Silva, P., 2002. OBJECT MODELLING OF INTERACTIVE SYSTEMS: THE UMLi APPROACH (PhD thesis).

Pinheiro da Silva, P., Paton, N.W., 2003. User Interface Modeling in UMLi. IEEE Softw. 20, 62–69.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T., 1994. Human-Computer Interaction, 1st ed. Addison Wesley.

Pressman, R.S., 2010. Ingeniería del software: un enfoque práctico. MacGraw-Hill.

Propp, S., Buchholz, G., Forbrig, P., 2008. Task Model-Based Usability Evaluation for Smart Environments. In: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams, HCSE-TAMODIA '08. Springer-Verlag, Berlin, Heidelberg, pp. 29–40.

Puerta, A., Eisenstein, J., 2002. XIML: a common representation for interaction data. In: Proceedings of the 7th International Conference on Intelligent User Interfaces, IUI '02. ACM, New York, NY, USA, pp. 214–215.

Puerta, A.R., 1997. A Model-Based Interface Development Environment. IEEE Softw. 14, 40–47.

Robles Luna, E., Grigera, J., Rossi, G., 2009. Bridging Test and Model-Driven Approaches in Web Engineering. In: Proceedings of the 9th International Conference on Web Engineering, ICWE '9. Springer-Verlag, Berlin, Heidelberg, pp. 136–150.

Rogers, Y., Sharp, H., Preece, J., 2011. Interaction Design: Beyond Human - Computer Interaction, 3rd ed. Wiley.

Rosson, M.B., Carroll, J.M., 2001. Usability Engineering: Scenario-Based Development of Human-Computer Interaction, 1st ed. Morgan Kaufmann.

Royce W.W., 1970. Managing the Development of large Software Systems: Concepts and Techniques. In: Proceeding of the IEEE WESTCON. Los Angeles, California.

Rumbaugh, J., Jacobson, I., Booch, G., 2004. The Unified Modeling Language Reference Manual, (paperback), 2nd ed. Addison-Wesley Professional.

Schmidt, D.C., 2006. Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer 39, 25–31.

Schwaber, K., 2001. Agile Processes and Self-Organization.

Schwaber, K., Beedle, M., 2001. Agile Software Development with Scrum, 1st ed. Prentice Hall.

Scrum Guides | Scrum.org - The home of Scrum [WWW Document], n.d. URL http://www.scrum.org/Scrum-Guides (accedido el 9.1.14).

Selic, B., 2003. The Pragmatics of Model-Driven Development. IEEE Softw. 20, 19–25.

Sendall, S., Kozaczynski, W., 2003. Model transformation: the heart and soul of model-driven software development. IEEE Software 20, 42–45.

Smith, W.J., 1996. ISO and ANSI ergonomic standards for computer products: a guide to implementation and compliance. Prentice Hall PTR, Upper Saddle River, N.J.

Snyder, C., 2003. Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces, 1st ed. Morgan Kaufmann.

Sommerville, I., 2006. Software Engineering: (Update), 8th ed. Addison Wesley.

Stapleton, J., Consortium, D., 2003. DSDM: business focused development. Pearson Education.

Steinberg, D., Budinsky, F., Paternostro, M., Merks, E., 2009. EMF: Eclipse Modeling Framework 2.0, 2nd ed. Addison-Wesley Professional.

Stone, D., Jarrett, C., Woodroffe, M., Minocha, S., 2005. User Interface Design and Evaluation. Morgan Kaufmann.

Sy, D., 2007. Adapting Usability Investigations for Agile User-Centered Design. Journal of Usability Studies 2, 112–132.

SysML Open Source Specification Project | SysML.org [WWW Document], n.d. URL http://www.sysml.org/ (accedido el 9.1.14).

Szekely, P.A., Sukaviriya, P.N., Castells, P., Muthukumarasamy, J., Salcher, E., 1996. Declarative interface models for user interface construction tools: the MASTERMIND approach. In: Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction. Chapman & Hall, Ltd., London, UK, UK, pp. 120–150.

Szyperski, C., Gruntz, D., Murer, S., 2002. Component software: beyond object-oriented programming. Pearson Education.

T23E-T10E STANDISH GROUP REPORT [WWW Document], 1995. URL http://www.spinroot.com/spin/Doc/course/Standish_Survey.htm (accedido el 9.1.14).

Taylor, A., 2000. IT projects: sink or swim. The Computer Bulletin 42, 24–26.

The Object Management Group, 2003. MDA Guide Version 1.0.1. Object Management Group.

Thomas, D., Barry, B.M., 2003. Model driven development: the case for domain oriented programming. In: Companion of the 18 Th OOPSLA, ACM Press. ACM Press, pp. 2–7.

Tullis, T., Albert, W., 2008. Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Morgan Kaufmann Publishers In.

Usability First: Usability in Website and Software Design [WWW Document], 2002. . Usability First:Usability in Website and Software Design. URL http://www.usabilityfirst.com/ (accedido el 9.1.14).

UsiXML - USer Interface eXtended Markup Language [WWW Document], n.d. URL http://www.usixml.org/en/home.html?IDC=221 (accedido el 9.1.14).

Van Lamsweerde, A., 2001. Goal-oriented requirements engineering: a guided tour. In: Fifth IEEE International Symposium on Requirements Engineering, 2001. Proceedings, pp. 249–262.

Van Lamsweerde, A., 2004. Goal-oriented requirements enginering: a roundtrip from research to practice [enginering read engineering]. In: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, pp. 4 – 7.

Vanderdonckt, J., 2005. A MDA-compliant Environment for Developing User Interfaces of Information Systems. In: Proceedings of the 17th International Conference on Advanced Information Systems Engineering, CAiSE'05. Springer-Verlag, Berlin, Heidelberg, pp. 16–31.

Wharton, C., Rieman, J., Lewis, C., Polson, P., 1994. Usability inspection methods. In: Nielsen, J., Mack, R.L. (Eds.), John Wiley & Sons, Inc., New York, NY, USA, pp. 105–140.

Wiegers, K., 2003. Software Requirements 2, 2nd ed. ed. Microsoft Press.

Wong, C., 1984. A Successful Software Development. IEEE Transactions on Software Engineering SE-10, 714–727.

XIML.org [WWW Document], n.d. URL http://www.ximl.org/ (accedido el 9.1.14).

XUL [WWW Document], n.d. Mozilla Developer Network. URL https://developer.mozilla.org/en-US/docs/XUL (accedido el 9.1.14).