Una Contribución al Mantenimiento de Modelos en Investigación Operativa desde la Ingeniería del Software

Conference Paper · November 2017

CITATIONS
0

3 authors:

Melina Vidoni
National Scientific and Technical Research Council
23 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Laura Cunico
Ingeniería Argentina (INGAR-UTN-CONICET), Argentina, Santa Fe
9 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Aldo Vecchietti
National Scientific and Technical Research Council
87 PUBLICATIONS **678** CITATIONS
SEE PROFILE

Una Contribución al Mantenimiento de Modelos en Investigación Operativa desde la Ingeniería del Software

Melina Vidoni, María Laura Cúnico, Aldo Vecchietti
Instituto de Desarrollo y Diseño, INGAR CONICET-UTN
Santa Fe, Argentina
{melinavidoni, laura-cunico, aldovec}@santafe-conicet.gov.ar

Resumen

Enfocar la Investigación de Operaciones (IO) desde una perspectiva de Ingeniería en Sistemas ha producido múltiples ventajas y técnicas, mayormente conocidas como soft-systems, *las cuales han cambiado la perspectiva del análisis y diseño del problema a modelar. Sin embargo, la literatura académica continúa denotando al mantenimiento y usabilidad de los mismos como un área que presenta grandes desafíos. Por su parte, la Ingeniería del Software (IS) ha reconocido la importancia de estos aspectos, además de su influencia al momento de permitir que el sistema cambie y se adapte a medida que la organización que lo implementa crece y evoluciona. Por lo tanto, este trabajo toma la perspectiva de Sistemas y propone Atributos de Calidad para los modelos matemáticos de IO, define tácticas que contribuyen a la mantenibilidad y usabilidad, y plantea posibles puntos de medición.*

1. Introducción

En la última década, la perspectiva del pensamiento sistémico y la Ingeniería del Software (IS) han realizado variadas contribuciones a la Investigación Operativa (IO) [1], incluyendo técnicas de elicitación y análisis de la problemática a modelar [2], metodologías para incluir stakeholders de forma dinámica en los proyectos [3], y el establecimiento de un concepto de ciclo de vida para proyectos de IO [4]. No obstante, en la actualidad, IS aún ofrece posibilidades para continuar contribuyendo a la Investigación Operativa [1]. Entre ellas, se destaca la importancia que tiene el matenimiento de los modelos de IO en su ciclo de vida [4, 5], lo que permite que éstos cambien y se adapten conforme evolucionan las problemáticas que modelan [6], y que se simplifique su lectura y usabilidad [7]. Sin embargo, no se encuentran propuestas específicas que aborden estos desafíos.

Ya en 1994, Coleman y colaboradores [8] definían a la mantenibilidad como un desafío central para la Ingeniería del Software. En la actualidad, se ha demostrado que la misma tiene un impacto directo en cuán sencillo resulta modificar y cambiar un artefacto [9]. A su vez, se ha establecido que la misma debe ser considerada tanto en el diseño como el desarrollo del proyecto, con el objetivo de reducir costos innecesarios y facilitar la adaptación del proyecto [10].

A raíz de lo mencionado previamente, este artículo propone trabajar sobre el desafío de mantenibilidad de los modelos de IO desde una perspectiva de sistemas. Esta decisión se fundamenta no sólo en las amplias contribuciones de la IS sobre la IO [6], incluidas en las propuestas denominadas *soft-systems*, sino también en la intrínseca relación entre modelos y software como facilitadores en los procesos de toma de decisiones [11]. Trabajos previos en el área de la IS han probado la adaptabilidad de algunos conceptos propios de esta disciplina al área de la IO [12], reforzando el alcance e impacto de las propuestas desarrolladas, así como también abriendo puertas para una mayor contribución de la IS en la IO.

No obstante, avanzar en esta área implica establecer qué es la calidad para la IO. En IS, la calidad ha tomado mayor relevancia con el paso de los años, y actualmente se considera como un elemento esencial para el éxito del negocio [13]. Además, se ha demostrado que puede controlarse y mejorarse, logrando un incremento en la productividad de los desarrolladores y en la satisfacción de los usuarios [14]. Trasladar estas características a la IO resulta clave para mejorar las transferencias a la industria, así como facilitar su adaptación constante a las situaciones modeladas [6].

Por esto mismo, primero se presentan Atributos de Calidad para los modelos de IO y luego se proponen tácticas que asisten a las decisiones de diseño para beneficiar el alcance de una respuesta positiva en dichos atributos. Estos conceptos son desarrollados exclusivamente para IO, partiendo desde estándares vigentes en la IS, tales como el ISO/IEC 25010:2011 [15] y las propuestas de tácticas generadas en el SEI (*Software Engineering Institute*) [16]. Finalmente, se plantean las áreas que permiten crear métricas para analizar el impacto de la aplicación de dichas tácticas.

Este trabajo se organiza del siguiente modo. La Sección 2 introduce los conceptos base que se utilizan a

lo largo del artículo. Luego, la Sección 3 presenta las definiciones de Atributos de Calidad propuestos, mientras que la Sección 4 discute el impacto de las modificaciones en los diversos elementos de los modelos de IO. A continuación, la Sección 5 elabora el concepto de tácticas, las decisiones propuestas en este trabajo, y la Sección 6 los compromisos y trade-offs que generan entre los Atributos. Relacionado a esto, la Sección 7 presenta los conceptos iniciales para la construcción de métricas e indicadores. Finalmente, la Sección 8 presenta discusiones, trabajos futuros y conclusiones al artículo.

2. Conceptos Iniciales

Esta propuesta impulsa la integración de dos disciplinas de alto impacto para el sector empresarial, las cuales pueden reforzar su contribución a través del trabajo conjunto. Por un lado, la IO a partir del desarrollo de *modelos matemáticos* de programación para abordar distintas problemáticas y requerimientos de la industria; y, por el otro lado, la IS brindando un eficiente soporte en las distintas etapas del ciclo de vida de esos proyectos, junto con sus técnicas para la gestión de la calidad. En cualquier caso, la intencionalidad final gira entorno a intensificar el aprovechamiento de los recursos materiales, virtuales y humanos de los que disponen las entidades involucradas.

Para dar inicio a esta propuesta, resulta funcional disponer de una noción global e integradora de los conceptos y elementos que serán ampliamente utilizados a lo largo del escrito.

Dentro del ámbito de la IO, se entiende por *modelo matemático* a una abstracción lógica-simbólica de una porción de la realidad. La captura y representación de las distintas entidades físicas reconocidas se realiza mediante la definición de variables y datos de entrada (valor fijo). La interconexión (vínculo) entre estos elementos se describe a través de formulismos y artilugios matemáticos, que conducen a la construcción de un sistema de proposiciones lógicas (como ecuaciones e inecuaciones), responsables de recrear lo más fidedignamente posible la realidad percibida por el modelador. Este grupo de proposiciones definen además lo que se conoce como *espacio de búsqueda*, es decir, el conjunto de todas las soluciones factibles del problema (aquellas que, sin ser necesariamente óptimas, satisfacen todos los requerimientos o relaciones planteadas).

El objetivo de este tipo de herramientas, dentro del área de la IO, resulta ser la determinación o mejora de una solución de índole pragmática. Esto conlleva un incremento en la complejidad de resolución, proporcional al volumen de datos procesados y al grado de realismo pretendido. Es por ello que para alcanzar una respuesta conveniente, se recurre a la utilización de entornos de optimización que requieren de una transcripción del modelo planteado a un concreto *lenguaje de programación*. Estos entornos emplean distintos *resolvedores*, cada uno de ellos especializado (a partir de alguna heurística) para la búsqueda de una solución, en principio factible.

En consecuencia, desde la perspectiva de la IS y en función de las definiciones previas, las tácticas propuestas se aplicarán sobre *modelos* caracterizados o construidos a partir de los siguientes elementos:

- Variable/s: representan entidades del problema, como ser número de equipos y personal, depósitos, plantas, etc. cuyos valores requieren ser especificados.
- Datos de entrada: grupo de valores conocidos o predictivos que condicionan el problema, sus variables y relaciones. Por ejemplo: períodos de planificación, precios de materias primas, pronósticos de demanda, etc.
- Respuesta: valores a determinar de las variables definidas, los cuales supeditan el resultado global.
- Objetivo: corresponde al sentido de búsqueda (optimización) de una solución, determinado por al menos una función objetivo. Ésta función condiciona la naturaleza del modelo (lineal, no lineal, mixto-entero, etc.)
- Restricción/es: proposición/es lógica/s (conjunto de ecuaciones e inecuaciones) que delimitan el espacio de búsqueda. Son también las responsables de vincular datos de entrada y variables; además de influir en la clasificación del modelo en lineal, no lineal, entero, mixto-entero, disyuntivo, etc.
- Lenguaje: refiere al software de programación específicamente elegido para transcribir y dar solución al modelo planteado. Esta elección depende principalmente de las habilidades del modelador y de las herramientas que dispone el cliente[1].
- Resolvedor: su selección está directamente ligada a la clasificación del modelo construido y al lenguaje de programación utilizado.

A su vez, es posible reconocer distintas etapas que hacen a su concepción o ciclo de vida. Siguiendo la propuesta de Ormerod [4], estas fases son denominadas de aquí en adelante como: *elicitación, diseño/intervención y desarrollo*; estas denominaciones mantienen una relación directa con las etapas de análisis, diseño y desarrollo reconocidas como fundamentales en los ciclos de vida del software [14, 15, 16].

Resumidamente, la etapa de elicitación se refiere al período de análisis y contextualización del problema, y la abstracción de los requerimientos del cliente, a través de una interacción coordinada. La fase de diseño o

[1] Se entiende por cliente a la entidad que encarga el modelo, y también los individuos responsables de su posterior uso.

intervención engloba transcribir los requerimientos y percepciones de funcionamiento a un bosquejo inicial del proyecto; en este punto se reconoce una fuerte relación de compromiso entre el grado de realismo deseado, es decir la representatividad del modelo, y la complejidad admitida en el mismo, limitada por el conocimiento de los usuarios, y la disponibilidad de hardware y software del cliente. Por último, la etapa de desarrollo corresponde a la instancia de transcripción del diseño a un lenguaje de programación seleccionado, su implementación y posterior extracción y análisis de resultados.

Finalmente, resulta importante destacar que todas estas etapas incluyen actividades de verificación y validación del modelo y sus requerimientos. No obstante, estos procesos no constituyen el eje central de este trabajo.

3. Atributos de Calidad

La calidad es el grado de excelencia de un ítem respecto a un aspecto en particular, y en IS esto se conoce como Atributos de Calidad o requerimientos no-funcionales. El estándar ISO/IEC 24765:2010 [17] los define como *"[…] un requisito que especifica el grado de una propiedad que afecta la calidad [aptitud] que el sistema o software debe poseer"*.

No obstante, partiendo de la definición propuesta en la Sección 2, se puede afirmar que un modelo también es un sistema: un conjunto ordenado de normas y procedimientos que regulan el funcionamiento de un grupo. En consecuencia, los modelos también son afectados por los Atributos de Calidad.

Sin embargo, generar Atributos de Calidad para la IO (AC-IO) no es una tarea sencilla, ya que se presentan dos dificultades principales. Primero, no existe un estándar del área que presente estos conceptos, y por lo tanto este trabajo es una propuesta innovadora con un alto potencial de desarrollo. Segundo, la relación de un modelo de IO con sus cualidades resulta más compleja que la existente entre las definiciones tradicionales de Atributos de Calidad y el software; esto se debe, principalmente, a que un AC-IO puede afectar no sólo al modelo sino también a la elección del lenguaje o resolvedor.

En consecuencia, la propuesta de AC-IO se genera en base a las clasificaciones y definiciones existentes en IS, las cuales se encuentran estandarizadas en el ISO/IEC 25010:2011[15], también conocido como modelo SQuaRE. La Tabla 1 describe la propuesta de AC-IO, junto con sus definiciones.

A partir de estas definiciones resulta necesario profundizar algunos conceptos.

- Los nombres de los AC-IO son los mismos que los Atributos de Calidad del Software (AC-S). Esta decisión permite favorecer la integración entre ambas áreas, y trazar una relación directa con el origen sistémico de los términos. A su vez, los nombres se trabajan en idioma inglés, dado que no existe una traducción de amplia aceptación.
- No todos los AC de Software pueden convertirse en un AC-IO. Esto sucede debido a que exceden el alcance de la IO, o porque no forman parte de su objetivo. Un ejemplo de esto es *Security*: un modelo o lenguaje no puede proveer medios para favorecer la autenticidad o autorización de sus usuarios, sino que esto debe realizarse en los sistemas de software que integren dichos modelos.
- Los AC-IO deben ser considerados durante todo el ciclo de vida de un proyecto de IO. Dado que también son requisitos, deben ser explicitados durante la fase de elicitación. Por esto mismo, su elicitación y obtención se adapta a técnicas existentes como *facilitated modeling* [3] y métodos de estructuración del problema (PSM, por sus siglas en inglés) [1].
- Relacionado con lo anterior, los AC-IO deben plantearse e iniciarse en la fase de diseño, para ser posteriormente implementados en el desarrollo. Esto último implica tomar decisiones estructurales que tengan un impacto directo en uno, o varios, AC-IO. En IS, esto se conoce como tácticas arquitectónicas.

Si bien todos los AC-IO tienen una amplia importancia e impacto en los modelos generados, resulta primordial acotar el ámbito de trabajo al momento de generar tácticas y, posteriormente, métricas que los evalúen. Por esto mismo, y debido a la necesidad de generar modelos que se adapten fácilmente al cambio y evolucionen junto con las situaciones que modelan [5, 6], se selecciona la familia de AC-IO de *Maintainability*, junto con sus sub-atributos.

4. Costo del Cambio

Así como ocurre en la arquitectura del software, el impacto de un Atributo de Calidad se relaciona con el *costo del cambio* [21]. Éste último refiere a la facilidad de implementación del cambio propuesto y plantea cuatro cuestionamientos comunes: quién es el responsable de realizar el cambio, en qué etapa resulta admisible, qué elementos son susceptibles, y cómo se mide el costo de cambiar [16]. En este caso en particular, el 'qué' y 'quién' son preguntas ligadas a la etapa del ciclo de vida en que se encuentra el modelo, es decir, al 'cuándo'. Respecto al 'cómo', los puntos de medición son propuestos en la Sección 7 de este trabajo.

En la Tabla 2 se resumen (no exhaustivamente) estas consideraciones por etapa, y se evalúa brevemente el costo del cambio en cada caso. Nótese también que allí se discrimina entre *modelador* y *desarrollador*, pudiéndose tratar o no de una misma persona o grupo de personas.

Tabla 1. Definiciones de Atributos de Calidad para modelos de Investigación Operativa.

Atributo	Definición
Compatibility	Modelo: Grado en el que los modelos pueden ser ejecutados mientras comparten un ambiente de hardware o software.
Interoperability	Modelo: Grado en que los modelos pueden obtener y usar información de otros sistemas, y almacenar sus resultados en sistemas externos (incluyendo bases de datos).
Coexistence	Modelo: Grado en que un modelo puede llevar a cabo sus funciones eficientemente mientras comparte un ambiente común (software) y recursos (hardware) con otros sistemas, sin un impacto negativo.
Portability	Modelo: Grado en que un modelo puede ser efectiva y eficientemente transferido de un ambiente operacional de hardware o software (incluyendo el resolvedor) hacia otro.
Adaptability	Modelo: Grado en que un modelo puede ser efectiva y eficientemente adaptado mientras el problema modelado cambia o evoluciona.
Reliability	Modelo: Grado en el que un modelo se ejecuta de una forma predeterminada, bajo condiciones específicas, por un período de tiempo dado. Se centra en condiciones de estrés en el hardware que afectan la resolución.
Availability	Resolvedor: Proporción del tiempo total en el cual el resolvedor está disponible y es capaz de ejecutar el modelo adecuadamente.
Fault Tolerance	Modelo: Grado en el que un modelo es capaz de sobrellevar situaciones infactibles, bloqueos e inconsistencias. Resolvedor: Grado en el que un resolvedor puede ejecutar los modelos como se planificaron más allá de la presencia de fallas de software y/o hardware.
Performance Efficiency	Modelo: Performance relativa a la cantidad de recursos usados bajo condiciones específicas (tipo de modelo, tipo de restricción, número de variables, disponibilidad de hardware, etc.).
Time Behavior	Modelo: Grado en el cual el tiempo de procesamiento, respuesta, y ejecución del modelo mientras corre (con un tipo específico de objetivo y parámetros) satisface un requisito de demora temporal.
Resource Utilization	Modelo: Grado en el cual la cantidad y tipo de recursos utilizados satisface los requerimientos.
Functional Stability	Modelo: Grado en el que el objetivo y restricciones del modelo cumplen las necesidades y la realidad mencionada por el cliente.
Correctness	Modelo: Grado en el que el modelo provee resultados correctos para la optimización, con el grado de precisión adecuado.
Appropriateness	Modelo: Grado en el cual el uso del modelo facilita alcanzar la generación de un plan de producción optimizado.
Maintainability	Modelo: Grado de efectividad y eficiencia con la que el modelo puede ser modificado.
Modularity	Modelo: Grado en el que un modelo puede reemplazar a sus componentes (objetivo, restricciones, parámetros) con un impacto mínimo en otros modelos y el sistema.
Modifiability	Modelo: Grado en el que un modelo puede ser efectiva y eficientemente modificado sin introducir defectos, inconsistencias, infactibilidad y/o degradar la calidad existente.
Testability	Modelo: Grado de efectividad y eficiencia por el cual puede establecerse un criterio de *test* para un modelo. Los *tests* pueden llevarse a cabo para determinar si esos criterios fueron alcanzados.
Usability	Modelo: Grado en que un modelo puede ser fácilmente comprendido, aprendido y atractivo para el modelador y el usuario, cuando se usa bajo condiciones específicas.
Learnability	Modelo: Grado en el cual un modelo puede ser usado por usuarios específicos para alcanzar metas de aprendizaje del modelo en un contexto especifico.
Operability	Modelo: Grado en el cual el modelo tiene atributos que hacen más fácil leerlo y usarlo. Ejemplos de esto son: comentarios, nombres de variables significativos, convenciones de nombres, etc.
Accessibility	Modelo: Grado en el que el modelo puede ser usado por personas con el más amplio rango de características y capacidades para alcanzar una meta particular en un contexto específico.

Además se emplea el término *cliente* en referencia a una persona o equipo de trabajo, especializado o no en técnicas de modelado.

Algunas cuestiones que se destacan examinando esta última tabla mencionada son:

- El bajo impacto en la primera etapa del ciclo de vida. Las variaciones en esta instancia del proyecto no representan un costo significativo para ninguna de las partes interesadas. Por esto mismo, se considera que es una etapa ideal para realizar cambios y ajustes.
- Un impacto medio-bajo en la etapa de diseño o intervención, el cual influencia principalmente las responsabilidades del modelador. Cabe destacar que

Tabla 2. Costo de cambio discriminado por etapa y elemento.

¿Cuándo? **Fase/Etapa**	¿Qué? **Elemento**	¿Quién? **Responsable**	**Costo de Cambiar**
Elicitar	Objetivo Restricciones Variables	Cliente/ Modelador	Bajo impacto. En esta instancia los elementos están sujetos a análisis, sin especificar, motivo por el cual los cambios no son críticos.
Diseño/ Interven-ción	Objetivo Restricciones Variables	Modelador	Impacto medio-bajo. Depende del problema a resolver, del tamaño y tipo de cambio.
	Resultados	Cliente	Impacto medio-alto. Un cambio de especificación en el tipo de resultado esperado puede afectar no sólo las restricciones, sino también el diseño del modelo.
	Lenguaje	Cliente/ Modelador	Impacto medio-bajo. Una mudanza de lenguaje puede significar un reto para el modelador (inclusive de aprendizaje) en el planteamiento del modelo.
Desarrollo	Objetivo	Cliente	Impacto muy alto. Puede comprometer un cambio en la perspectiva del problema, en los resultados obtenidos y sus respectivos análisis. Puede requerir incremento o eliminación de variables, parámetros y restricciones, afectando las definiciones e interacciones entre ellos. Puede considerarse un nuevo modelado.
	Restricciones	Cliente/ Desarrollador	Impacto medio. No implica esfuerzos significativos para el desarrollador, aunque puede requerir modificaciones en las definiciones de variables o parámetros.
	Variables	Desarrollador	Impacto medio. Al agregar/modificar una variable se requiere un controlar global de las restricciones y/o objetivo, y modificarlas caso sea necesario. Puede introducir errores en el modelo.
	Datos Entrada	Cliente/ Desarrollador	Impacto medio. La mayor influencia recae sobre los resultados y en la validez de los mismos. Puede requerir de estimaciones y/o proyecciones para suplir los datos faltantes.
	Resultados	Cliente	Bajo impacto si se trata exclusivamente de cambios de visualización. Los agregados o modificaciones requeridas corresponden sólo a líneas de lenguaje, sin que esto impacte en la esencia del modelo. Alto impacto al agregar/modificar resultados. Pueden requerirse cambios en la estructura del modelo (objetivo, restricciones, variables, datos de entrada), y la introducción de nuevos datos de entrada, puede generar errores en el modelo. Además, existe una posibilidad de impacto sobre la empresa, en aquellos caso en que trabajen con archivos 'en crudo' para generar ellos mismos la visualización.
	Resolvedor	Desarrollador	Impacto medio. Es posible que cambien los resultados (porque se modifican las estrategias de búsqueda y el grado de convergencia a la solución) y, por ende, la validez de los mismos. También impacta sobre la consumición de recursos de hardware y, consecuentemente, afectan la velocidad de resolución, los recursos requeridos, etc.
	Lenguaje	Cliente/ Desarrollador	Muy alto impacto. Interviene en el costo de la empresa (compra de licencias de software, capacitación de su personal, enlace con los sistemas vigentes, etc.). Además, cambiar lenguaje puede representar un cambio de paradigma para los modeladores (en caso de no posee experiencia en el nuevo lenguaje, influenciando el personal responsable del diseño del proyecto. Por lo tanto, puede comprometer tiempos de ejecución (requeridos para la capacitación) y derivar en un rediseño. Por otro lado, la complejidad de modelado puede variar ampliamente de un lenguaje a otro.

se asume que el encargado del diseño es una persona idónea en el área.

- Alto impacto en la fase de desarrollo. En esta etapa, pequeñas variaciones en los requerimientos pueden traducirse y comprometer severamente una parte del modelado, o inclusive derivar en un rediseño. Todo esto implica una fuerte repercusión en las etapas posteriores al proyecto, en lo que

refiere al proceso de actualización y adaptabilidad de los modelos matemáticos, en respuesta a la dinámica de evolución de los sistemas modelados. Por otro lado, en esta misma fase, es de vital importancia considerar las herramientas de trabajo vigentes del cliente. Una mudanza de lenguaje puede acarrear elevados costos tanto a nivel monetario, ante la necesidad de adquisición de licencias, así como sobre el factor humano, requiriendo de capacitación y actualización del personal.

Todo esto conduce a profundizar y aunar esfuerzos en la gestación de tácticas que auxilien, principalmente, a la mantenibilidad como Atributos de Calidad de los modelos.

5. Tácticas para Mantenibilidad

En la definición sistémica original, las tácticas son decisiones que influencian las respuestas de los AC-S, impactando consecuentemente en el sistema, para un estímulo determinado [22].

En la IS, las tácticas asociadas a la mantenibilidad afectan a los elementos basados en dos conceptos principales: cohesión y acoplamiento [16]. La cohesión es el grado en que se relacionan las funciones realizadas por cada sub-componente, mientras que el acoplamiento es cuánto interactúan entre sí [23]. Estos conceptos también existen entre los elementos de los modelos, y resultan vitales para el mantenimiento de los mismos. De esta forma, por ejemplo, la relación entre restricciones y variables puede medirse en términos de cohesión y acoplamiento, pudiendo limitar la utilización de éstas últimas a sólo las esenciales.

Por otro lado, y como se ha mencionado en la sección previa, el costo del cambio también afecta a la generación de tácticas de mantenibilidad, dado que éste se refiere tanto al impacto económico, como al esfuerzo requerido para realizar cada modificación [21]. En consecuencia, las tácticas desarrolladas deben ser capaces de abordar esta situación para reducir el costo existente.

Bajo estas consideraciones, y partiendo desde las tácticas de modificabilidad para software elaboradas en el SEI [16], se proponen diez tácticas para IO que se presentan más detalladamente en las siguientes subsecciones.

5.1. Táctica: Abstracción de Términos Comunes

La táctica original para el software, implica la refactorización del código para examinar las responsabilidades existentes y extraer como 'factor común' aquellas similares [16].

En relación a los modelos, a menudo un grupo de proposiciones o restricciones pueden repetir ciertas estructuras algebraicas (o términos); esto implica que, ante un cambio, sea necesario modificar todas sus ocurrencias, incrementando no sólo el esfuerzo requerido, sino también la posibilidad de incurrir en un error. Estas consecuencias son similares a las que existen en el desarrollo del software [24].

Por lo tanto, la propuesta de esta táctica implica extraer el término matemático que se repite, y definirlo en una nueva ecuación, denominada *fragmento de restricción*. Ésta última debe contener una nueva variable de definición, que se da a conocer como *variable resumen*, cuya única función es reemplazar al término en cuestión en cada una de las restricciones que lo invoque.

A continuación se muestra un ejemplo en lenguaje GAMS. El conjunto de ecuaciones (1) describe la escritura tradicional.

```
e11(i,t)..
LS(i,t) =E=
(peso1(i,t)*d_inf(i,t)+peso2(i,t)*d_sup(i,t))-
SUM(s, SS(i,s,t));

e12(i,t)..
LS(i,t) =L=
(peso1(i,t)*d_inf(i,t)+peso2(i,t)*d_sup(i,t));
```
(1)

Luego, el conjunto de ecuaciones (2) muestra las mismas restricciones, utilizando la abstracción de términos. Esto implica adicionar una variable resumen y un fragmento de ecuación.

```
ponderar(i,t)..
sum_Ponderar(i,t) =E=
peso1(i,t)*d_inf(i,t)+peso2(i,t)*d_sup(i,t);

e11(i,t).. LS(i,t) =E= sum_Ponderar(i,t) -
SUM(s, SS(i,s,t));

e12(i,t).. LS(i,t) =L= sum_Ponderar(i,t);
```
(2)

5.2. Táctica: Convención de Nombres

En programación, una convención de nombres es un conjunto de reglas para seleccionar la secuencia de caracteres usadas por los identificadores que nominan a las entidades en el código fuente y en la documentación [25]. Su utilización simplifica la comprensión del mismo y reduce el esfuerzo necesario, permite concentrarse en la lógica y no en el formato, facilitando el análisis del código [26]. No obstante, cada lenguaje suele promover una convención diferente.

Debido a las ventajas que presenta la utilización de estas reglas, las mismas se proponen como una táctica adicional. Para esto, la Tabla 3 resume algunos ejemplos de nomenclaturas, no determinantes.

Hay que destacar que los nombres deben ser significativos, pero tampoco excesivamente largos, que no puedan recordarse. A su vez, esta táctica ofrece sugerencias de convenciones: cada modelador puede

adaptarlas a su lenguaje o costumbres, pero siempre manteniendo coherencia y consistencia en su aplicación.

Tabla 3. Sugerencias de convención de nombres.

Elemento	Convención	Ejemplo
Variables	PacalCasing. Sustantivos	`ExpectedSales SupplyProducs`
Variables (Binarias)	Pascal Casing. Verbos.	`isPrinterEnable`
Restricciones	Minúsculas. Carácter: Guión	`mass_balance production_level`
Tablas (Datos Entrada)	CamelCasing Indicando subordinados.	`periodicDemandOf( products,periods)`

5.3. Táctica: Dividir Responsabilidad

En la táctica homónima propuesta para el SEI, es posible reducir el costo de modificar una responsabilidad simple al dividirlas en responsabilidades más pequeñas y ubicarlas en módulos separados [16].

Para trasladar esto a los modelos de IO, se propone dividir su implementación en múltiples archivos. Esta es una característica soportada por varios lenguajes de amplia utilización, tales como GAMS y MatLab. No obstante, no existen guías para su estructuración.

En consecuencia, esta táctica busca crear módulos al agrupar el contenido en directorios, cada uno limitado a un elemento diferente. A su vez, se propone separar la declaración de la inicialización de las variables y parámetros: de esta forma, los datos concretos quedan aislados de la definición de tipo y nombre.

De este modo, en primer lugar el proyecto debe agruparse dentro de una carpeta raíz que posea el nombre del proyecto. Dentro de ésta, la separación propuesta se constituye de la siguiente forma: una carpeta para cada elemento (datos de entrada, variables, objetivos, restricción y resultados), y una carpeta adicional que contenga los escenarios disponibles. Dentro de cada una, se trabaja con *archivos de definición*, aquellos que contienen declaraciones concretas, y *archivos de inclusión*, que sólo contienen sentencias de inclusión de los primeros.

Este procedimiento guarda una relación con el encapsulamiento empleado en el desarrollo del software, ya que no sólo facilita la reutilización sino que también limita el impacto de los cambios [27]. A su vez, contribuye a la separación semántica de los elementos, permitiendo que aquellos que deben modificarse a menudo -los datos de entrada, los resultados y los escenarios- estén agrupados de forma relacionada.

5.4. Táctica: Comentarios

En IS, un comentario es una explicación legible por el programador y adicionada al código fuente, con el propósito de simplificar el entendimiento del mismo, y que es ignorado por compiladores e intérpretes [28]. Aunque varios autores han discutido qué porcentaje del código debe estar comentado [29, 30], su correcta utilización permite clarificar la intención del código [29]. Por esto mismo, en IS varios lenguajes ofrecen lineamientos sobre la aplicación de los comentarios.

Por lo tanto, y dado que la programación matemática guarda una estrecha relación y similitud con la programación de software, ésta táctica propone utilizar la capacidad de comentarios de los lenguajes existentes para simplificar la lectura del modelo. De esta forma, se sugieren tres tipos de comentarios:

- Por Archivo: al inicio de cada archivo, si se emplea la táctica 'Dividir Responsabilidad', el comentario debe incluir autor, ubicación, año, y una breve descripción del contenido del archivo.
- Por Definición: dentro de cada archivo que define elementos, arriba de cada definición debe incluirse un breve comentario que describa el elemento.
- Por inclusión: en cada archivo donde se realice una inclusión de otro archivo, arriba de esta línea se sugiere colocar un comentario que describa qué se incluye.

El siguiente extracto de código presenta un ejemplo de comentario de tipo inclusión en lenguaje GAMS.

```
*
* This file introduces the definition of the
* parameter that lists the number of units in
* stock at time zero for each item of raw
* material.
*
PARAMETER initialStock(products) /
$INCLUDE
C:\..\..\..\initialStock(products).txt
/ ;
```

5.5. Táctica: Agrupar Restricciones

Como se presentó en la Sección 2, cada modelo se constituye por proposiciones que intentan reproducir el funcionamiento de un sistema físico o real. A menudo, un subconjunto de éstas se relaciona teóricamente, modelando distintos aspectos o áreas dentro del problema abordado. Por ejemplo, un grupo de restricciones es responsable de modelar los requerimientos de materias primas; otro define y controla los niveles de inventario; otro grupo determina los flujos productivos; etc.

Por lo tanto, resulta conveniente una estructura de análisis y diseño ordenada, donde se distingan aquellos grupos de restricciones interrelacionados por función. Mediante esta táctica, el agrupamiento puede combinarse con 'Dividir Responsabilidad' para generar archivos individuales para cada grupo, o separarse mediante

comentarios siguiendo la propuesta de 'Utilización de Comentarios'.

Esta propuesta deriva de varias tácticas existentes, como ser 'mantener la coherencia semántica, que implica separar la lógica en porciones relacionadas, y utilizar encapsulación, que reduce la probabilidad de que un cambio afecte módulos que no debería [16].

En IS, se ha demostrado que esta combinación no sólo simplifica la reutilización de código [31], sino que también simplifica el aprendizaje y comprensión del mismo[22]. En consecuencia, su aplicación en IO resulta clave para favorecer tanto la mantenibilidad como la usabilidad del código.

5.6. Tácticas: Agrupar Escenarios y Limitar Uso de Componentes

En un modelo de IO, se entiende por escenario a un subconjunto de variables, parámetros, restricciones y objetivos, que generan resultados respecto a una configuración fijada o limitada. Su propósito es evaluar el modelo y las respuestas bajo condiciones particulares.

A partir de esto, surgen dos tácticas altamente relacionadas entre sí:

- Agrupar Escenarios: propone generar múltiples escenarios, compuestos por el subconjunto de elementos con el que trabajan.
- Limitar Uso de Componentes: cada escenario sólo puede incluir aquellos elementos que efectivamente utiliza, y sólo puede escribir sus resultados en un directorio específico. En particular, esto se combina con 'Dividir Responsabilidades'.

Así, este par se relaciona con las tácticas de IS denominadas *utilización de envoltorios*, la cual consiste en emplear una interfaz que controla la información del módulo, y *restringir caminos de comunicación*, que implica que no haya conexiones innecesarias entre elementos [16].

De esta forma, se espera no sólo simplificar la evaluación de distintas posibilidades de la situación modelada, sino también reducir la cantidad de cambios que deben realizarse al controlar estas posibilidades mediante escenarios concretos. A su vez, la utilización de escenarios permite aplicar otra táctica: 'Tests Unitarios'.

5.7. Táctica: Test Unitarios

Los test unitarios son un concepto de la IS, en el cual un programa se descompone en las unidades más pequeñas posibles, y cada una es evaluada al generar entradas específicas con salidas concretas y conocidas[32]. Detectar y corregir defectos cerca de cuando se originan, es clave para lograr un desarrollo rápido y eficiente en costos [33].

Para los modelos de IO, las unidades comprobables más pequeñas son restricciones y ecuaciones de objetivos. El proceso para aplicar esta táctica requiere de tres pasos:

1. Definir las unidades a testear.
2. Para cada una, generar datos de prueba (ya sea extrayendo los datos existentes, o creando datos manualmente) que tengan resultados conocidos, sean correctos o incorrectos.
3. Crear escenarios, aislados de los escenarios finales, que sólo se limiten a utilizar estos datos y evaluar cada combinación.

A modo de ejemplo, para testear p para la ecuación eq_A, se establecen dos conjuntos de datos de prueba: cp_1 y cp_2. Si al ejecutar el escenario de test todo funciona adecuadamente, cp_1 debe satisfacer la ecuación y cp_2 no debe satisfacerla; de la misma forma, si en un test cp_2 satisface la ecuación, entonces se ha detectado un error.

5.8. Táctica: Generar Documentación

Para la IS, la documentación de un sistema de software es un reporte textual que incluye diagramas y figuras, y que explica cómo opera un sistema particular, y cuáles son sus funcionalidades principales. Desde hace varios años, la generación de una documentación adecuada, legible y actualizable es considerada como una práctica prioritaria para mejorar el mantenimiento [34]. No obstante, para que la documentación sea relevante y efectiva, la misma debe ser fácil de actualizar, concreta y no ser excesiva [35].

Por lo tanto, hay que destacar que en esta táctica no se espera realizar un *big design up front*, es decir completar la documentación antes de desarrollar el modelo, sino que el mismo sea creado de forma iterativa e incremental a medida que se avanza en el ciclo de vida del proyecto.

Así, ésta documentación debe ofrecer una mirada general sobre el problema modelado y explicar decisiones de diseño básicas, tales como la selección del lenguaje y resolvedor, hardware requerido, entre otros. Además, debe clarificar aspectos del modelado, similares a los grupos de restricciones, los test unitarios y los escenarios incluidos. Sin embargo, el reporte generado debe ser fácil de cambiar, y no contener información innecesaria.

Cabe destacar que como parte de este trabajo se ha generado una plantilla de documentación sugerida, basada en las recomendaciones de la IEEE y del SEI para las especificaciones de requerimientos del software y documentación de arquitectura, respectivamente [36, 37]. A pesar de esto, transcribir dicha plantilla se encuentra fuera del alcance de este artículo.

6. Compromisos y Trade-offs

Al igual que en la IS, existen compromisos (o *trade-*

offs) entre los Atributos de Calidad, esto implica que al favorecer alguno/s de ellos se afecte negativamente a otros [32]. En el caso de los AC-IO, este impacto sucede no sólo con otros atributos, sino también en los elementos del modelo que afectan.

En ambas comparaciones, la simbología empleada fue adaptada dela propuesta de Harrison y Avgeriou [33], adicionando símbolos para casos específicos. De esta forma, la Tabla 4 resume la notación completa.

Tabla4. Resumen de la notación empleada en la evaluación del impacto.

Símbolo	Efecto (Elemento / Atributo)
++	Favorece muy positivamente, y es clave aplicarlo.
+	Favorece positivamente.
	No tiene efectos ni positivos ni negativos.
-	Afecta negativamente.
--	Tiene un impacto muy dañino, y puede llegar a prevenir su realización.
~	Puede afectar o no. En caso de hacerlo, el efecto sólo es positivo.
¬	Inversa del anterior, sólo efecto negativo.
±	Puede afectar tanto negativa como positivamente, dependiendo de las circunstancias.

6.1. Impacto en Elementos

En la Tabla 5 que se presenta a continuación, se evalúa el impacto de las distintas tácticas seleccionadas sobre los respectivos elementos de los modelos de IO, discriminados por etapa. Esta es una evaluación ideal, derivada de los elementos que se ven afectados por cada una de las tácticas propuestas.

Las tácticas detalladas en la Sección 5 evidencian un balance positivo en lo que refiere a su impacto en las distintas etapas del ciclo de vida de los modelos de IO. Este comportamiento se intensifica especialmente en la etapa de desarrollo de los mismos, contribuyendo de manera esencial en la implementación y transferencia de estas herramientas.

Esto se condice con la intencionalidad de la propuesta de extensión de las tácticas características de la IS al ámbito de la IO. La adaptación de estas estrategias, de eficiencia probada para el desarrollo del software, beneficia distintos atributos como la facilidad de aprendizaje, la modificabilidad y el mantenimiento de los modelos; atributos ampliamente valorados en el sector.

Como contraste, el mayor impacto o costo negativo del cambio percibido en la Tabla 5, responde a un incremento en los requerimientos de hardware, en lo que refiere a la compilación y ejecución de los modelos planteados. Esto ocurre, por ejemplo, con el aumento del número de variables y restricciones, debido a la implementación de la táctica de "Abstracción de términos comunes". Sin embargo, este conflicto puede ser eventualmente resuelto sin requerir de un esfuerzo adicional significativo.

6.2. Trade-off De Atributos

En la Tabla 6 se resumen las interacciones de las

Tabla 5. Impacto de las tácticas propuestas sobre los elemento de los modelos de IO.

	Elemento	Conv. Nombres	Div. Resp.	Coment.	Agrupar Restric.	Agrup. Escen.	Limit. Uso Componentes	Abs.Térm. Comunes	Generar Documen.
Elicitar	Objetivo					++			++
	Restric.								++
	Variables								++
Diseño/ Interv.	Objetivo			+		+			
	Restric.			+	++	+			
	Variables					~			
	Resultados					++			
	Lenguaje								
Desarrollo	Objetivo	++	+	+			~		++
	Restric.	++	++	++	++		~	±	++
	Variables	++	++	++			~	-	+
	D. Entrada	++	++	+			~		+
	Resultados	+	++	++			++	-	++
	Resolvedor								~
	Lenguaje	~	-	~					

Tabla 6. Compromisos y trade-offs de las tácticas en los AC-IO.

AC-IO	Conv. Nom.	Div. Resp.	Coment.	Agrup. Rest.	Agrup. Escen.	Limit. Uso Comp.	Abs.Térm. Comunes	Generar Doc.	Test Unit.
Compatibili-ty									
Interopera-bility		++				+			
Coexistence						~			
Portability									
Adaptability	+	+		+	~		+	++	+
Reliability									
Availability						~			
Fault Tolerance		±		~	±	+	+		++
Performanc e Efficiency		±		¬	~	±	--		
Time Behavior		-		¬	~	-	--		
Resource Utilization		-		¬	~	-	--		
Functional Stability					+			~	
Correctness							-		+
Appropriate-ness					+				
Usability	++	±	++	++	+		¬	++	±
Learnability	++	-	++	++	+		¬	++	±
Operability	++	±	++	++	~		¬	++	+
Accessibility	~	¬	~					~	

distintas tácticas propuestas sobre los AC-IO. Al igual que antes, las relaciones introducidas son ideales, y derivadas de las relaciones existentes entre AC-S y las tácticas arquitectónicas más comúnmente aplicadas para la mantenibilidad.

Una diferencia a remarcar entre la IO y la IS es la relación directa entre mantenibilidad y usabilidad visible en la primera de estas disciplinas. Por un lado, el software es compilado y/o interpretado sin que el usuario vea las líneas de código, siendo el ambiente de trabajo del usuario el resultado de este proceso. En cambio, en IO el modelador/usuario trabaja directamente con el código fuente, lo que implica que, al mejorar la mantenibilidad y la comprensión del código, también se mejora la usabilidad del modelo. En este caso, ambos grupos de AC-IO están directamente relacionados y manifiestan un impacto preponderantemente positivo con la implementación de las tácticas propuestas.

Un punto crítico a destacar es el trade-off entre *Performance Efficiency* y *Maintainability*. El primero de éstos sufre un impacto generalmente negativo al incrementar los niveles de satisfacción de los atributos de *Usability*, *Learnability* y *Operability*. Este comportamiento muchas veces responde a una necesidad de aumento de detalle o separabilidad que deriva en una exigencia mayor sobre los recursos computacionales. Nuevamente la táctica de 'Abstracción de Términos Comunes' supone un incremento en el número de variables y restricciones que componen el modelo. No obstante, éste resulta ser usualmente despreciable (con un uso adecuado), de modo tal que, priman las ventajas que manifiesta sobre los atributos de *Adaptability* y *Fault Tolerance*. Algo semejante ocurre con la táctica de 'Dividir Responsabilidad': en este caso, el punto de inflexión entre la generación de un impacto positivo o uno negativo, se encuentra condicionado por el buen uso de la estrategia. Un exceso en la creación de archivos puede implicar dificultades tanto de compilación como de comprensión del modelo.

7. Puntos de Medición

El verdadero nivel de satisfacción de una cualidad y su impacto en el modelo, sólo puede determinarse

específicamente cuando se puede medir. Para esto, la IS emplea métricas: funciones que miden el grado en el cual un sistema o proceso posee una propiedad específica [20]. Estas métricas están asociadas a los AC-S, y se encuentran definidas en el estándar internacional ISO/IEC 25023:2016 [38].

La utilización de métricas no resulta ajena a la IO. Por ejemplo, el criterio de parada de los métodos de búsqueda a través de la determinación del gap que estima el error entre la solución hallada y la óptima, puede ser considerado como tal. Sin embargo, este uso es diferente al empleado en IS.

Por esto mismo, se considera viable la generación de métricas e indicadores, valores que permitan interpretar los resultados arrojados por las primeras [20], para evaluar el grado de aplicación de los AC-IO propuestos. El primer paso de este análisis está relacionado con la generación de preguntas que permitan dirigir la atención hacia elementos medibles relacionados con los AC-IO *Mantainability*, y sus sub-atributos. Algunas de las preguntas son:

- ¿Cuánto se utilizan los escenarios?
- ¿Puede un usuario distinguir qué se requiere como dato de entrada y qué se provee como resultado?
- ¿Cuánto le lleva a un usuario comprender cómo se representa la situación real en el modelo?
- ¿Cuántos elementos del modelo pueden comprenderse a partir del código?
- ¿Los directorios son altamente cohesivos?
- ¿Pueden usarse los tests unitarios para determinar si el modelo está listo para operar, o no?

A partir de aquí, el desafío más importante y complejo a afrontar es la generación de funciones o métricas, que den respuestas a estas preguntas y, posteriormente, proponer rangos de valores que permitan interpretarlas adecuadamente. Sin embargo, la discusión y concepción de tales métricas, así como su análisis extendido se reserva como línea de trabajo futuro.

8. Conclusiones

La Ingeniería del Software (IS) ha realizado múltiples y valiosas contribuciones a la Investigación Operativa (IO), generando un grupo de técnicas y métodos conocidos como *Soft-OR* o *soft-systems*. No obstante, aún existen muchas vías de integración sobre las cuales avanzar, y otras por profundizar. Una de ellas, la que concentra la atención en este trabajo, es la posibilidad de mejorar el mantenimiento de los modelos de IO.

Ante el rápido cambio que viven las organizaciones y la sociedad en la actualidad, resulta esencial que los modelos construidos para representar su accionar, sean capaces de adaptarse y evolucionar paralelamente a un bajo costo, tanto económico como de esfuerzo profesional. Sin embargo, las propuestas en esta área son escasas o poco efectivas. A menudo, la baja practicidad de los modelos para ser modificados o mantenidos y la dependencia de personal capacitado, representan una preocupación que deriva en el desánimo de su desarrollo. Esto promueve el abandono de las ventajas competitivas que brindan este tipo de herramientas y la adopción de soluciones menos rentables y de mayor riesgo.

Partiendo de estas premisas, el trabajo presentado propone la extensión de técnicas propias de la IS, con el objeto de mejorar la mantenibilidad en modelos de IO. Para esto, inicialmente, se genera una definición de Atributos de Calidad exclusiva para IO (AC-IO), y se analiza el costo de realizar cambios en las distintas etapas del ciclo de vida de los modelos. Luego, se presentan tácticas que, de aplicarse adecuadamente, pueden tener un impacto significativamente positivo en la mantenibilidad. No obstante, al igual que ocurre en la IS, los AC-IO presentan ciertos *trade-offs*, o relaciones de compromiso, que también son analizadas y discutidas a lo largo de esta propuesta.

Si bien lo expuesto a este punto es un desarrollo mayormente teórico, el mismo ofrece un marco de trabajo de gran potencial sobre el mantenimiento de los modelos de IO.

Por un lado, la propuesta se destaca por proveer una contribución efectiva sobre una problemática que evidencia un vacío de abordaje importante, como se menciona anteriormente. Más aún, el carácter innovador de la propuesta se refuerza al no hallarse, hasta el momento, evidencia de la existencia de una definición formal de AC-IO (aun cuando algunos de ellos son considerados de forma implícita en el desarrollo). Esta axiomatización concreta, permite elicitarlos y considerarlos expresamente durante el desarrollo. Por otra parte, el análisis exhaustivo y crítico de los *trade-offs* se basa en los realizados para Atributos de Calidad en el Software, y certifica un impacto positivo de las tácticas propuestas, no sólo en la fase de desarrollo de los modelos, sino también con respecto a los otros AC-IO.

Finalmente, se presentan varios trabajos futuros potenciales. Por un lado, la elaboración de las funciones de métrica y establecimiento de indicadores como medio de evaluación concreto del grado de aplicación de cada AC-IO. Por otra parte, la necesidad de realizar un estudio de caso, para evaluar el grado de aceptación de ésta propuesta con expertos y/o académicos. Finalmente, queda abierta la posibilidad de generar tácticas y métricas para los otros AC-IO planteados al inicio de este trabajo.

9. Referencias

[1] J. Mingers y L. White, «A review of the recent contribution of systems thinking to operational research and management science,» *EJOR*, vol. 207, nº 3, pp. 1147-

1161, 2010.

[2] J. Mingers, «Soft ORcomesofage—but noteverywhere!,» *Omega*, vol. 39, nº 6, pp. 729-741, 2011.

[3] L. Franco y G. Montibeller, «Facilitated modelling in operational research,» *European Journal of Operational Research*, vol. 205, nº 3, pp. 489-500, 2010.

[4] R. Ormerod, «The transformation competence perspective,» *Journal of the Operational Research Society*, vol. 59, nº 11, p. 1435–1448, 2008.

[5] L. Fortuin y M. Zijlstra, «Operational research in practice: Consultancy in industry revisited,» *European Journal of Operational Research*, vol. 120, nº 1, pp. 1-13, 2000.

[6] A. Wierzbicki, «Modelling as a way of organising knowledge,» *European Journal of Operational Research*, vol. 176, nº 1, pp. 610-635, 2007.

[7] D. von Winterfeldt y B. Fasolo, «Structuring decision problems: A case study and reflections for practitioners,» *European Journal of Operational Research*, vol. 199, nº 3, pp. 857-866, 2009.

[8] D. Coleman, D. Ash, B. Lowther y P. Oman, «Using metrics to evaluate software system maintainability,» *Computer*, vol. 27, nº 8, pp. 44-49, 1994.

[9] L. Ping, «A Quantitative Approach to Software Maintainability Prediction,» de *International Forum on Information Technology and Applications (IFITA)*, Kunming, China, 2010.

[10] J.-C. Chen y S.-J. Huang, «An empirical analysis of the impact of software development problem factors on software maintainability,» *Journal of Systems and Software*, vol. 82, nº 6, pp. 981-992, 2009.

[11] R. Hämäläinen, J. Luoma y E. Saarinen, «On the importance of behavioral operational research: The case of understanding and communicating about dynamic systems,» *EJOR*, vol. 238, nº 3, p. 623–634, 2013.

[12] M. Vidoni, J. Montagna y A. Vecchietti, «Improving the Assessment of Advanced Planning Systems by Including Optimization Experts' Knowledge,» de *19th International Conference on Enterprise Information Systems (ICEIS)*, Porto, Portugal, 2017.

[13] R. Al-Qutaish, «Quality Models in Software Engineering Literature: An Analytical and Comparative Study,» *Journal of American Science*, vol. 6, nº 3, pp. 166-175, 2010.

[14] M. Rawat, A. Mittal y S. Dubey, «Survey on Impact of Software Metrics on Software Quality,» *International Journal of Advanced Computer Science and Applications*, vol. 3, nº 1, pp. 137-141, 2012.

[15] ISO/IEC, 25010:2011 - Systems and software engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, I.J.1.7, Suiza, ISO, 2011.

[16] F. Bachmann, L. Bass y R. Nord, «Modifiability Tactics,» Software Engineering Institute (SEI), USA, 2007.

[17] ISO/IEC/IEEE , 24765:2010(E) - Systems and software engineering - Vocabulary, International Standardization Office, 2010.

[18] Y. B. Leau, W. K. Loo, W. Y. Tham y S. F. Tan, «Software Development Life Cycle Agile vs Traditional Approaches,» de *International Conference on Information and Network Technology (ICINT)*, Singapur, 2012.

[19] N. M. A. Munassar y A. Govardhan, «A Comparison Between Five Models Of Software Engineering,» *International Journal of Computer Scie*, vol. 7, nº 5, pp. 94-101, 2010.

[20] V. Guntamukkala, H. J. Wen y J. M. Tarn, «An empirical study of selecting software development life cycle models,» *Human Systems Management*, vol. 25, nº 4, pp. 265-278, 2006.

[21] M. Jorgensen y M. Shepperd, «A Systematic Review of Software Development Cost Estimation Studies,» *IEEE Transactions on Software Engineering*, vol. 33, nº 1, pp. 33-53, 2007.

[22] L. Bass, P. Clements y R. Kazman, «Architectural Tactics and Patterns,» de *Software Architecture in Practice*, Tercera ed., S. E. Institute, Ed., Addison-Wesley Professional, 2012, pp. 203-250.

[23] G. Gui y P. Scott, «Measuring Software Component Reusability by Coupling and Cohesion Metrics,» *Journal of Computers*, vol. 4, nº 9, pp. 797-805, 2009.

[24] M. O'Keeffe y M. Cinnéide, «Search-based refactoring for software maintenance,» *Journal of Systems and Software*, vol. 81, nº 4, pp. 502-516, 2008.

[25] F. Deissenboeck y M. Pizka, «Concise and consistent naming,» *Software Quality Journal*, vol. 14, nº 3, p. 261–282, 2006.

[26] D. Jones, «Operand names influence operator precedence decisions,» de *ACCU Conference*, 2008.

[27] K.-K. Lau y F. Taweel, «Data Encapsulation in Software Components,» de *Component-Based Software Engineering (CBSE)*, vol. LNCS 4608, H. Schmidt, I. Crnkovic, G. Heineman y J. Stafford, Edits., Springer, Berlin, Heidelberg, 2007, pp. 1-16.

[28] P. Grubb y A. Takang, Software Maintenance: Concepts and Practice, Segunda ed., Singapur: World Scientific Publishing Company, 2003.

[29] S. McConnell, Code Complete: A Practical Handbook of Software Construction, Segunda ed., D. Musgrave, Ed., USA: Microsoft Press, 2004.

[30] R. Morelli y R. Walde, Java, Java, Java: Object-Oriented Problem Solving, Tercera ed., USA: Prentice Hall, 2006.

[31] R. Lutowski, Software Requirements: Encapsulation, Quality, and Reuse, Primera ed., USA: Auerbach Publications, 2005.

[32] L. Zhu, A. Aurum, I. Gorton y R. Jeffery, «Tradeoff and Sensitivity Analysis in Software Architecture Evaluation Using Analytic Hierarchy Process,» *Software Quality Journal*, vol. 13, nº 4, p. 357–375, 2005.

[33] N. Harrison y P. Avgeriou, «Leveraging Architecture Patterns to Satisfy Quality Attributes,» de *European Conference on Software Architecture (ECSA)*, vol. LNCS 4758, Springer, Berlin, Heidelberg, 2007, pp. 263-270.