Desarrollo de un Modelo Integral para el Estudio de la Calidad en Aplicaciones Web por medio de Simulación

María Julia Blas[1*], Horacio Leone[1], Silvio Gonnet[1]

[1]Instituto de Desarrollo y Diseño INGAR
Universidad Tecnológica Nacional (UTN)
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Avellaneda 3657 – Santa Fe - Argentina

{mariajuliablas,hleone,sgonnet}@santafe-conicet.gov.ar

Resumen. *El estudio de los entornos de computación en la nube se divide en dos áreas principales: infraestructura y aplicación. A nivel de aplicación, los arquitectos de software especifican los productos siguiendo el modelo de servicios de software. Este trabajo propone un modelo integral basado en simulación que captura la información de las arquitecturas de software de aplicaciones web a fin de evaluar cuantitativamente aspectos de calidad en etapas tempranas de desarrollo. La propuesta integra un modelo semántico para la definición de las propiedades de calidad con una herramienta de modelado basada en patrones de diseño. Los modelos de simulación son generados en base a la combinación de ambos elementos. Estos modelos son especificados utilizando una adaptación del formalismo de simulación Discrete Event System Specification denominada Routed DEVS, la cual ha sido diseñada para abordar el estudio de problemas de ruteo.*

Palabras Clave. *Calidad de producto de software; Simulación basada en eventos discretos; Servicios de software; Patrones de diseño.*

Development of a Built-in Model for Studying Quality in Web Applications through Simulation

Abstract. *The study of cloud computing environments is divided into two main areas: infrastructure and application. At the application level, software architects specify the products following the software services model. This paper proposes a novel simulation-based model that captures data from the software architecture design of web-based applications in order to evaluate quality in the early phases of development. Our proposal integrates* i) *a semantic model that defines the desired quality properties of the software product, with* ii) *a modeling tool based on design patterns. The final simulation models are generated automatically by following the combination of both elements. These models are discrete-event simulation models. We use an adaptation of the Discrete Event System Specification called Routed DEVS to define their specification. The Routed DEVS formalism has been designed in a previous work to address the study of routing problems.*

Keywords. *Software product quality; Discrete-event simulation; Software services; Design patterns.*

BLAS, Maria Julia; LEONE, Horacio; GONNET, Silvio. Desarrollo de un Modelo Integral para el Estudio de la Calidad en Aplicaciones We b por medio de Simulación. Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação, [S.l.], v. 1, n. 14, out. 2021. ISSN 2446-7634. Disponível em: <https://revistas.setrem.com.br/index.php/reabtic/article/view/409>.

1. Introducción

Las prácticas de Ingeniería de Software (ISW) permiten a los desarrolladores analizar un problema y diseñar una solución sólida para el mismo, lo que contribuye a construir productos de software de alta calidad [Pressman and Maxim 2020]. En este contexto, la calidad del producto puede abordarse mediante un enfoque basado en mediciones o un enfoque basado en modelos [Hasselbring 2018].

Los enfoques basados en mediciones realizan mediciones sobre los productos de software reales [Van Hoorn et al. 2012], por lo que únicamente son aplicables cuando el sistema ya se encuentra implementado (motivo por el cual siempre brindan datos reales). Por su parte, al análisis de calidad basado en modelos proporciona información en una etapa temprana, es decir, antes de que se construya el sistema de software [Reussner et al. 2016]. De esta manera, este enfoque permite identificar los problemas en una etapa temprana, permitiendo reducir los costos de reparación de errores y/o mantenimiento. Sin embargo, la elaboración de los modelos apropiados para dicho estudio requiere de un esfuerzo adicional. Además, su construcción requiere de una inversión de tiempo de desarrollo adicional (el cual no siempre se dispone en los proyectos de desarrollo de software reales).

A pesar de los recientes avances en el área de gestión de la calidad [Tian 2005; Hussain and Abdulsalam 2014], las complejidades cognitivas y técnicas del proceso de diseño en los entornos de *computación en la nube* (CC por sus siglas en inglés, *cloud computing*) son evidentes. Tal proceso aún no se ha comprendido en forma acabada y, junto con el estudio de las propiedades de calidad del producto, existe una necesidad de herramientas que soporten la evaluación de las alternativas de diseño de forma eficiente y flexible, en etapas tempranas de desarrollo.

El diseño arquitectónico es considerado una de las primeras especificaciones de cualquier producto de software. En este contexto, el diseño de una arquitectura engloba las decisiones que el arquitecto ha tomado en relación con las propiedades de calidad, motivo por el cual representa la primera oportunidad para evaluar la adecuación de tales decisiones [Hasselbring 2018]. Cuando los componentes arquitectónicos son utilizados para formular un modelo de simulación, la arquitectura puede ser utilizada como vehículo para estimar el comportamiento del producto final. Este tipo de estrategias de análisis ha sido utilizado en entornos de CC a nivel de *infraestructura*, tal como lo proponen los autores en [Koziolek et al. 2009; Meiappane et al. 2013]. Sin embargo, estos enfoques suelen referir exclusivamente a un subconjunto reducido de propiedades de calidad que no incorpora el nivel de *aplicación de software* a la descripción del entorno bajo análisis.

En términos generales, existen dos aspectos básicos a ser tenidos en cuenta al momento de utilizar técnicas de simulación sobre arquitecturas de aplicaciones de software web, a saber: *i) definición de propiedades de calidad*: Es decir ¿qué propiedades deben ser medidas? ¿cómo se relacionan? ¿qué indicadores deben ser utilizados?, y *ii) formulación e implementación de modelos de simulación*: Esto es ¿qué formalismo de simulación debe ser utilizado? ¿cuántos modelos deben diseñarse? ¿cómo se vinculan estos modelos? Ambos aspectos deben resolverse de forma conjunta en virtud de obtener un modelo de evaluación basado en simulación que brinde una solución útil y completa a los arquitectos. Luego, la formulación de modelos de análisis y evaluación integrales que permitan capturar la información de arquitecturas de

software web para evaluar cuantitativamente la calidad en etapas tempranas del proceso de desarrollo constituye un área de investigación relevante en ISW.

En este trabajo se formula un modelo de evaluación integral basado en la simulación de componentes de software a nivel de arquitectura de aplicación. El principal objetivo es el desarrollo de herramientas computacionales que permitan estimar un conjunto de atributos de calidad predefinidos en base al diseño arquitectónico de una aplicación web. Tomando en consideración que los modelos de calidad juegan un papel crítico en el estudio de la calidad (ya que proporcionan una herramienta de soporte para la definición de los atributos a ser relevados sobre un producto [Nistala et al. 2019]), el modelo propuesto se basa en el modelo de calidad de producto de software propuesto en el estándar [ISO/IEC 25010 2011]. A partir de un conjunto predefinido de propiedades de calidad relevantes para este tipo de productos de software (aplicaciones de software web), se obtiene un conjunto de valores iniciales que conforman el objetivo del modelo de simulación. Los modelos de simulación finales se obtienen directamente a partir del modelo de diseño arquitectónico. Para esto, se presenta una herramienta de modelado de arquitecturas de aplicaciones de software basada en patrones de diseño que facilita la definición de arquitecturas válidas. Tales diseños son mapeados a modelos de simulación basados en eventos discretos, los cuales se definen en base a los formalismos Discrete Event System Specification (DEVS) [Zeigler et al. 2018] y Routed DEVS (RDEVS) [Blas et al. 2017a]. De esta manera, esta propuesta provee una perspectiva de ISW útil para la estimación de aspectos de calidad web utilizando el diseño de la arquitectura como esqueleto estructural para la definición del modelo de simulación que opera como soporte del modelo de análisis y evaluación.

El resto del trabajo se encuentra estructurado de la siguiente manera. La Sección 2 introduce el concepto de calidad de producto de software y su vínculo con la evaluación de arquitecturas. Además, establece los problemas que existen para estimar la calidad en aplicaciones de software web junto con los conceptos elementales utilizados como base de trabajo. La Sección 3 presenta formalmente la solución propuesta detallando la forma en la cual se vinculan cada una de las partes componentes. Para esto, presenta: *i)* una estrategia de documentación basada en ontologías que es usada para la especificación de las características de calidad y el conjunto de métricas útiles para su relevamiento; *ii)* una estrategia de representación de arquitecturas de software, que incluye un estudio de los patrones de diseño y la implementación de una herramienta de modelado; *iii)* un conjunto de transformaciones componente-a-modelo a partir de las cuales se obtienen los modelos de simulación requeridos para evaluar la arquitectura en base a las propiedades de calidad; y *iv)* un conjunto de modelos de simulación de comportamiento de usuario que son usados para alimentar el proceso de evaluación. Además, incorpora la prueba de conceptos usada para validar la propuesta, haciendo uso de dos arquitecturas genéricas comúnmente abordadas en la literatura. Finalmente, la Sección 4 se encuentra dedicada a las conclusiones y líneas de investigación futuras que se desprenden del trabajo realizado.

2. Estado del Arte

2.1. Calidad de Producto y Arquitecturas de Software

La IEEE define calidad de software como “el grado en el cual un software posee una combinación de atributos deseados” [IEEE 1061 1998]. Esta definición busca definir la

calidad de software por medio de las características o propiedades esperadas en un producto de alta calidad. Por lo tanto, requiere examinar las características individuales asociadas con la calidad y sus interrelaciones, poniendo especial atención en aquellas propiedades que hacen a la correctitud funcional del producto. Una definición más completa es propuesta en [Pressman and Maxim 2020], donde se define la calidad del software como "un proceso de eficaz aplicado de manera que cree un producto de software útil que proporcione un valor mensurable para los que lo desarrollan y los que lo utilizan". De esta definición, se destacan los siguientes aspectos: *i)* un *proceso eficaz* establece la infraestructura que brinda soporte a los esfuerzos realizados para construir un producto de software de alta calidad; *ii)* un *producto de software útil* ofrece el contenido, las funciones y las características que el usuario final desea, pero, al mismo tiempo, brinda estas características de forma confiable y sin errores; y *iii)* al añadir valor tanto para los desarrolladores como para el usuario final, un producto de software de alta calidad proporciona beneficios tanto para la organización encargada del desarrollo (ya que un producto de alta calidad requiere menor esfuerzo de mantenimiento, menos correcciones de errores y menor apoyo al cliente) como así también para la comunidad de usuarios finales (ya que el software proporciona una capacidad útil de forma tal que agiliza alguna tarea).

Aunque múltiples autores han definido de distinta forma el concepto de *calidad de software*, la mayoría de las definiciones ponen en evidencia la necesidad de evaluar la calidad de acuerdo con el estudio de los atributos (o propiedades) de calidad asociados a los requerimientos funcionales y no funcionales del producto. Esto es evidente en el caso de la definición propuesta en [IEEE 1061 1998]. Por su parte, la definición propuesta en [Pressman and Maxim 2020] considera que un *producto útil* siempre satisface los requisitos que han sido declarados explícitamente por los interesados. Sin embargo, al mismo tiempo, dicho producto también satisface un conjunto de requisitos implícitos (por ejemplo, la facilidad de uso) que se esperan de todo el software de alta calidad.

Aún más, es importante reconocer que la calidad de un producto de software debe considerarse en todos los estados de evolución del producto bajo desarrollo [Rempel and Mäder 2017; Sabri and Alfifi 2017]. Es decir, evaluar la calidad final una vez construido el producto no es recomendable ya que los defectos y/o errores que lleven a problemas de calidad en esta instancia del proceso de desarrollo suelen no tener solución (o su solución es más costosa que una solución similar aplicada en etapas tempranas). Sin embargo, debido a que el conjunto de requerimientos se obtiene y perfecciona a medida que se lleva a cabo el proceso de solución, resulta difícil diseñar productos adaptables a todo posible cambio requerido por los *grupos de interés* (ya sean desarrolladores y/o usuarios finales).

En etapas de desarrollo tempranas, alcanzar un nivel de calidad apropiado implica lograr un diseño arquitectónico adecuado a la naturaleza de los atributos de calidad relevantes. Los requerimientos de calidad son las propiedades más importantes del trabajo arquitectónico [Hasselbring 2018]. En este contexto, la *arquitectura* de un producto de software comprende los componentes del software, las propiedades visibles de dichos componentes y las relaciones que existen entre ellos [Bass et al. 2013]. Luego, al construir un diseño arquitectónico es necesario seleccionar los principios de trabajo adecuados a fin de lograr un balance apropiado entre las propiedades de calidad requeridas (las cuales compiten entre sí). Dado que la arquitectura muestra la

correspondencia entre los requerimientos y el producto de software final, este modelo proporciona una base para el análisis racional de las decisiones de diseño que se han tomado a lo largo del proceso de desarrollo. Por este motivo, las arquitecturas deben ser analizadas y evaluadas a fin de determinar su habilidad para cumplir con los atributos de calidad deseados.

Tanto el análisis como la evaluación arquitectónica han surgido con el objetivo de conectar los atributos de calidad con las decisiones de diseño arquitectónico [Clements and Kazman 2001]. Para el análisis de los atributos de calidad, se utilizan modelos de arquitectura de software para analizar si el sistema puede cumplir sus requisitos no funcionales [Crnkovic et al. 2005]. Por su parte, los objetivos de las evaluaciones arquitectónicas son estimaciones sobre los efectos de las decisiones arquitectónicas, en relación con los atributos de calidad del software. En todos los casos, la evaluación arquitectónica ayuda a: *i)* tomar decisiones de diseño comprendiendo sus consecuencias sobre el producto final, *ii)* identificar las compensaciones entre distintos requerimientos de calidad, *iii)* comprobar el cumplimiento de los requisitos, y *iv)* mejorar la gestión de riesgos [Hasselbring 2018]. Sin embargo, este proceso de evaluación no debe centrarse en la aplicación de una única métrica universal (inexistente), sino que debe cuantificar atributos individuales para, luego, realizar una compensación sobre las métricas obtenidas [Barbacci et al. 1995].

A lo largo de los años se han definido múltiples estrategias para llevar a cabo este tipo de evaluación arquitectónica. Desde un punto de vista tradicional existen dos clases de estrategias: *i)* basadas en escenarios, y *ii)* basadas en medición cuantitativa o cualitativa. Recientemente se han desarrollado estrategias de análisis y evaluación centradas en la combinación de técnicas existentes que buscan considerar el dinamismo constante con el cual se producen los cambios en el software. Con este tipo de técnicas se busca proveer una predicción razonable de los atributos de calidad del sistema de software objetivo aplicando herramientas computacionales sobre artefactos intermedios (en este caso, el diseño de arquitectura de software). Por ejemplo, en [Bachmann et al. 2005] los autores presentan un "framework de razonamiento" que actúa como una modularización del conocimiento referido a los atributos de calidad. En este caso, los requisitos que una arquitectura debe satisfacer se especifican como escenarios concretos de atributos de calidad. Luego, el framework proporciona mecanismos de transformación que modelan la arquitectura con respecto a una determinada teoría de atributos de calidad. De esta manera, se distingue entre el modelo arquitectónico y el modelo de atributos de calidad, caracterizando las acciones de razonamiento como transformaciones arquitectónicas básicas. De forma similar, los autores de [Schneider et al. 2018] definen un framework que hace uso de los conocimientos arquitectónicos logrados de manera informal (es decir, aquellos conocimientos que no pueden procesarse en enfoques estructurados y formalizados) a fin de brindar soporte a la toma de decisiones de diseño. En este caso, el trabajo conecta modelos de razonamiento cualitativo con modelos de estimación cuantitativa de la calidad para optimizar las arquitecturas de software en lo que respecta a ambas perspectivas. De esta manera, los atributos de calidad para los que no se dispone de un modelo de evaluación cuantitativa pueden ser usados en los enfoques de optimización de las arquitecturas de software automatizadas.

Considerando que simular es “el proceso de diseñar un modelo de un sistema real y realizar experimentos sobre el mismo a fin de entender su comportamiento y/o

evaluar estrategias que mejoren su operación" [Law and Kelton 1991] y, teniendo en cuenta que ya se han llevado a cabo experiencias de simulación a nivel de infraestructura de software [Koziolek et al. 2009; Meiappane et al. 2013]; en este trabajo se emplea el modelo de la arquitectura de software a nivel de aplicación como descriptor base de un modelo de simulación por medio del cual se evalúa el comportamiento del sistema (diseñado) de acuerdo a las propiedades de calidad (deseadas). Es decir, se propone el uso de simulación como técnica de análisis y evaluación arquitectónica en el contexto de aplicaciones de software.

2.2. Aplicaciones de Software y Evaluación de la Calidad

El paradigma de CC es "un modelo que habilita acceso a Internet bajo demanda de forma conveniente y generalizada a fin de compartir un conjunto de recursos de computación configurables que pueden ser asignados y entregados de forma rápida con un esfuerzo marginal reducido y/o con mínima interacción con el proveedor del servicio" [Mell and Grance 2011]. Uno de los principales objetivos que persigue este nuevo paradigma es el aumento en la capacidad y productividad en tiempo de ejecución sin requerir de una inversión en nueva infraestructura, licencias de software ni en capacitación para empleados [Khan et al. 2013]. A fin de lograr este objetivo, el modelo de CC propone entregar distintos tipos de tecnología de información bajo el formato de servicio.

La mayoría de las arquitecturas de CC propuestas en la literatura se basan en modelos de capas [Luo et al. 2011; Jadeja and Modi 2012; Odun-Ayo et al. 2018]. En su forma más abstracta, estas capas pueden reducirse a dos grandes capas generales, cada una de las cuales reúne un subconjunto de capas específicas, a saber: *aplicación* e *infraestructura*. Los usuarios acceden a los servicios de software alojados en la *capa de aplicación* por medio de un acceso indirecto a funcionalidades remotas. Tanto los recursos como los medios de comunicación sobre los cuales se ejecutan dichas funcionalidades quedan encapsulados en la *capa de infraestructura*. Luego, para garantizar los beneficios del modelo de CC, se debe lograr mantener un nivel de calidad apropiado en ambas capas. Sin embargo, al mismo tiempo, se debe garantizar el cumplimiento de los requerimientos de calidad en forma conjunta. En el caso de la *capa de aplicación*, esta calidad refiere a calidad de software.

Sin embargo, dada la falta de madurez en el área de calidad propia de la complejidad intrínseca de los productos de software [Pressman and Maxim 2020], al intentar estimar o evaluar la calidad en aplicaciones de software basadas en contextos de CC se presentan tres problemas fundamentales:

- *Presencia de nuevas propiedades de calidad específicas de este nuevo tipo de tecnologías*: Debido al rápido surgimiento de las tecnologías de CC, nuevos aspectos de calidad han surgido como consecuencia de su dinámica [Lewis 2010; Breu et al. 2014]. Estos aspectos no se encuentran contemplados en los enfoques de evaluación tradicionales de la calidad, por lo que se requieren nuevas técnicas de estudio (o modificaciones de las técnicas existentes) que se ajusten al dinamismo evidenciado en este tipo de software.
- *Ausencia de estrategias de representación basadas en patrones de diseño arquitectónico*: Dada la falta de madurez en el área, los patrones de diseño orientados a CC aún no están totalmente establecidos. Además, no existe una

técnica de representación que facilite la elaboración (y posterior comprensión) de este tipo de arquitecturas [Zalewski and Kijas 2013]. Frecuentemente, las estrategias de representación mezclan componentes de infraestructura con software de aplicación, generando confusión y ambigüedad tanto a quienes quieren aplicarlas como así también a quienes quieren interpretarlas.

- *La calidad final de los productos web depende de ambas capas (aplicación e infraestructura)*: La dependencia de la infraestructura imposibilita la objetividad de estudio en relación con el comportamiento de la aplicación, ya que su calidad siempre está afectada por las características propias de la capa subyacente (sobre la cual se ejecutan sus componentes de software). Esto obstaculiza el trabajo de los arquitectos ya que no disponen de los elementos necesarios para evaluar sus diseños de forma aislada a la infraestructura.

Luego, resulta necesario brindar toda la ayuda posible a los arquitectos de aplicaciones de software web a fin de aumentar su productividad. El objetivo es mejorar, no sólo la tarea de diseño, sino también la evaluación de las propuestas alternativas formuladas por el arquitecto como posibles soluciones que buscan cumplir de la mejor manera posible los requerimientos de calidad definidos. En este contexto, en este trabajo se diseña e implementa un ambiente de análisis y evaluación arquitectónica basado en simulación que da respuesta a estos problemas combinando estrategias de solución a distintos niveles del proceso de desarrollo.

3. Resultados

3.1. Conceptualización y Desarrollo del Modelo de Análisis y Evaluación

La Figura 1 esquematiza la estructura del modelo de evaluación propuesto en este trabajo. Tal como se ha enunciado con anterioridad, su principal objetivo es *transformar el diseño de una arquitectura de software web en un modelo de simulación ejecutable que permita obtener mediciones de calidad basadas en métricas predefinidas, las cuales han sido especificadas de forma transversal al proceso de desarrollo*. Tal como puede observarse, este modelo abarca todos los elementos requeridos para la definición de la calidad, la representación de la arquitectura, la obtención de un modelo de simulación equivalente y la generación de modelos de usuario asociados al dominio de aplicaciones de software.

En este punto, es importante destacar que no incluye el modelado ni la simulación de la infraestructura subyacente, por lo que únicamente abstrae la definición y el estudio de los elementos incluidos en la capa de aplicación. La incorporación de los elementos propios de la capa de infraestructura forma parte de una etapa de trabajo futura. Esto implica que, en este punto de desarrollo, el modelo de evaluación arquitectónica considera la arquitectura de aplicaciones de software web en un contexto de ejecución asociado a una infraestructura ideal.

En este escenario, el esquema propuesto vincula tres niveles de trabajo: *proceso de desarrollo de software*, *artefactos de software* y *nuevos modelos de software*. En el nivel de *proceso de desarrollo* se esquematizan (de forma secuencial, con un alto nivel de abstracción) las primeras tres etapas de un proceso de desarrollo genérico. Sobre este proceso de desarrollo en el nivel de *artefactos de software* se identifican los artefactos comúnmente creados como resultado de las dos primeras etapas de trabajo, a saber:

especificación de requerimientos y *arquitectura de software*. Tomando como referencia estos artefactos, el nivel denominado *nuevos modelos de software* actúa como mecanismo complementario a los enfoques de ISW tradicionales incorporando nuevos artefactos (derivados de la vinculación y/o transformación de los existentes) con el objetivo de facilitar la evaluación de la calidad a partir del diseño de las arquitecturas de software.

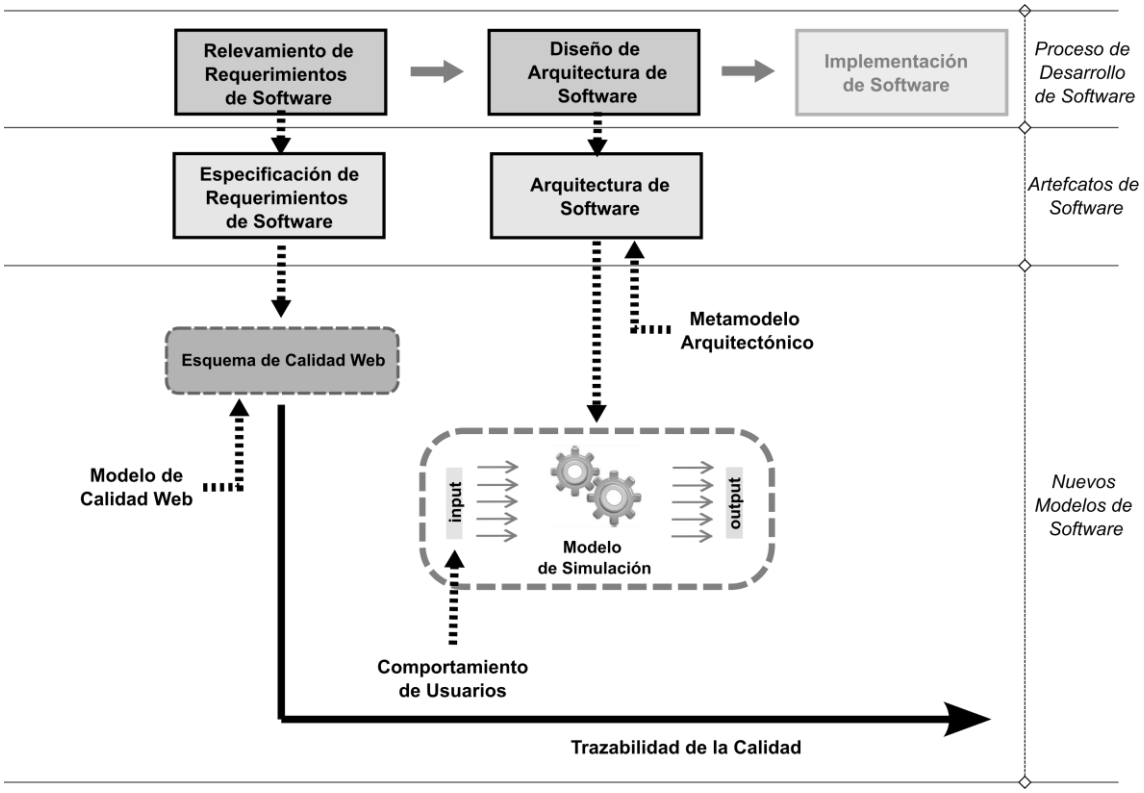

Figura 1. Ambiente propuesto en base a un modelo de evaluación centrado en artefactos de ISW.

La *especificación de requerimientos de software* es utilizada como elemento de entrada para la generación de un nuevo documento denominado *esquema de calidad web* que encapsula las relaciones existentes entre los requerimientos de calidad definidos para un producto de software específico y el modelo de calidad de producto de software propuesto en [ISO/IEC 25010 2011]. La definición de este documento sienta las bases necesarias para todo tipo de evaluación de calidad que deba realizarse a lo largo del proceso de desarrollo de la aplicación de software. Esto se debe a que dicho artefacto identifica características, atributos y métricas aplicables al producto de software bajo desarrollo en distintas etapas del proceso. Esto facilita la trazabilidad de las propiedades de calidad analizadas a lo largo del ciclo de vida del producto. Luego, por medio de la aplicación de criterios de análisis, es posible estudiar la información de calidad definida a fin de estimar la adecuación de un producto intermedio en una etapa de desarrollo específica. La especificación de este nuevo artefacto se presenta en la Sección 3.1.1.

En este contexto, los elementos que componen un *esquema de calidad* son utilizados como base para construir el proceso de evaluación de arquitecturas de software web. Estos elementos definen las propiedades de calidad de interés a ser

relevadas a lo largo del desarrollo motivo por el cual, en el caso de la aplicación de técnicas de simulación, tales elementos definen las variables de salida (es decir, aquellas variables que deben obtenerse como resultado de la ejecución del modelo). Sin embargo, aun cuando el arquitecto posea la experiencia requerida para formular el modelo de simulación a ser utilizado como base de la evaluación de calidad, no es deseable que invierta tiempo y esfuerzo en esta tarea. Esta particularidad, sumada a la falta de una especificación independiente para la definición de arquitecturas de software web, queda oculta en el modelo de evaluación propuesto como parte de una herramienta de modelado gráfico basada en un *metamodelo de componentes arquitectónicos*. Este metamodelo permite crear y validar instancias de patrones de diseño arquitectónico asociadas a aplicaciones de software web como parte de una estrategia de representación. Los patrones de diseño analizados corresponden a los propuestos en [Fehling et al. 2014]. En la Sección 3.1.2 se presenta la herramienta de modelado gráfico desarrollada en base a este metamodelo.

El uso de un subconjunto definido de elementos arquitectónicos (es decir, los componentes identificados en el metamodelo) brinda la posibilidad de transformar cada componente arquitectónico en un modelo de simulación específico de forma independiente. Dado que las composiciones y relaciones se encuentran explícitamente definidas como parte de los patrones de diseño incluidos en el metamodelo, es posible derivar los vínculos de simulación a partir del diseño de la arquitectura original (reflejando las influencias de componentes como conexiones entre modelos). Luego, la creación del modelo de simulación integral en base al diseño arquitectónico que permita estimar las métricas de calidad definidas en un esquema de calidad específico se convierte en una tarea transparente para el arquitecto de software. Es decir, en nuestro modelo de evaluación, el modelo de simulación se deriva a partir de la definición de un conjunto de modelos de simulación elementales que han sido especificados a partir de comportamientos y estructuras basadas en los componentes arquitectónicos empleados en el diseño. La Sección 3.1.3 describe esta transformación.

Finalmente, la definición de la calidad y de los modelos de simulación asociados no es la única actividad requerida para la evaluación de arquitecturas de software por medio de simulación. También es necesario disponer de mecanismos que permitan representar el comportamiento esperado de los usuarios de la aplicación de software a fin de generar un conjunto de solicitudes web que se asemejen a lo esperado en la ejecución real de la aplicación. Es decir, es necesario diseñar un proceso de generación de entradas para el modelo de simulación acorde con lo esperado en un contexto de ejecución real. Por este motivo, además de los modelos de simulación de arquitecturas, el modelo propuesto incluye modelos de simulación que representan el *comportamiento de usuario* según distintos tipos de cargas de trabajo. Estos modelos se presentan en la Sección 3.1.4.

3.1.1. Ontología para la Definición de Esquemas de Calidad en Aplicaciones de Software

La determinación del conjunto de elementos y propiedades que definen un nivel adecuado de calidad para un sistema de software específico es una pregunta altamente dependiente del contexto [Kitchenham and Pfleeger 1996]. Por este motivo, normalmente se llevan a cabo diversas actividades de control de calidad para prevenir o eliminar ciertas clases de problemas que provocan un impacto negativo en el producto

y/o para reducir la probabilidad o la gravedad de ese impacto negativo cuando es inevitable [Tian 2005].

Dado que la calidad está relacionada con todas las actividades del proceso de desarrollo, la especificación de los requerimientos de calidad debe elaborarse durante el transcurso de las primeras etapas de trabajo. De esta manera, su especificación puede ser utilizada tanto por los arquitectos de software como por los desarrolladores a fin de evaluar el ajuste de sus diseños e implementaciones respecto a la calidad esperada del producto (y así prevenir o reducir los impactos negativos).

En este contexto, es posible aislar un conjunto de propiedades de calidad iniciales tomando como punto de partida la Especificación de Requerimientos de Software en combinación con la aplicación de Modelos de Calidad de Producto. El uso de Modelos de Calidad es una práctica comúnmente aceptada para el estudio de la calidad en productos de software [Miguel et al. 2014]. Según [ISO/IEC 25010 2011], un Modelo de Calidad es "el conjunto de características, y las relaciones entre ellas que proporcionan una base para especificar tanto los requisitos de calidad como su evaluación". Luego, teniendo en cuenta que la especificación de atributos de calidad sin ambigüedad es difícil y tediosa, por medio del uso de Modelos de Calidad es posible dirigir este proceso a fin de construir artefactos que documenten la calidad en relación aspectos formalmente definidos.

En [Blas et al. 2017b], los autores proponen un nuevo tipo de documento denominado *esquema de calidad* que estructura la definición de propiedades de calidad en productos de software en base al Modelo de Calidad de Producto de Software definido en el estándar [ISO/IEC 25010 2011]. Este modelo de calidad determina las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto software determinado. Para esto, el modelo interpreta la calidad como el grado en que un producto de software satisface los requisitos de sus usuarios (o grupos de interés) aportando de esta manera algún tipo de valor. El modelo categoriza la calidad de un producto de software en características y subcaracterísticas, generando una jerarquía de propiedades de calidad y estableciendo sus relaciones de forma clara y concisa.

Sobre el modelo de calidad elegido, el *esquema de calidad* complementa la definición de las propiedades específicas haciendo uso de conceptos propios de los dominios de métricas de software y producto de software. De esta manera, como complemento de la definición formal, los autores presentan una implementación ontológica que brinda soporte a la instanciación de *esquemas de calidad* en dominios específicos.

Esta definición se presenta como una ontología compuesta de tres modelos semánticos, a saber: modelo semántico de calidad (*quality semantic model*), modelo semántico de métricas de software (*metric semantic model*) y modelo semántico de producto de software (*software semantic model*). Las relaciones entre estos modelos se definen de acuerdo con los siguientes lineamientos:

- una métrica de software (*metric*) es de utilidad para estimar un atributo de software (*attribute*), y
- un atributo (*attribute*) se encuentra asociado a una subcaracterística de calidad (*subcharacteristic*).

La Figura 2 esquematiza estas relaciones tomando como base los conceptos centrales de cada dominio.

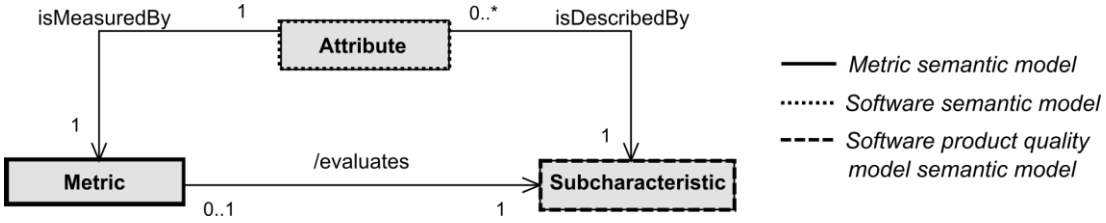

Figura 2. Relaciones entre los principales conceptos modelados como parte de la ontología esquemas de calidad.

Los modelos semánticos definidos incluyen un conjunto de reglas que permiten inferir nuevo conocimiento y verificar la integridad de las instancias creadas a partir de la ontología. Además, por medio de un conjunto de consultas específicamente diseñadas es posible explorar la definición de un *esquema de calidad* a fin de conocer la información requerida como base para la estrategia de evaluación de la calidad. Esto posibilita la obtención de la información necesaria para la estimación de un subconjunto de propiedades de calidad dadas.

Siguiendo el formato de documentación basado en la ontología, un esquema de calidad para el estudio de aplicaciones de software fue definido. La Tabla 1 resume el conjunto de elementos incluidos en dicha instanciación, poniendo énfasis en los principales conceptos abordados en cada caso. Aunque diferentes aplicaciones de software tendrán distintos requerimientos de calidad, el conjunto de elementos incluidos en el esquema diseñado es susceptible de ser usado como base en cualquier escenario.

Tabla 1. Elementos incluidos en el Esquema de Calidad Web.

Subcaracterística de Calidad	Atributo de Software	Métrica
Comportamiento temporal	Tiempo de invocación	Comportamiento visto por el usuario
Uso de recursos	Uso de infraestructura	Utilización de recursos de hardware
Madurez	Precisión del servicio	Precisión en las respuestas
Disponibilidad	Robustez del servicio	Robustez del servicio
Tolerancia a fallos	Estabilidad del servicio	Cobertura de tolerancia a defectos
		Cobertura de recuperación fallas
Escalabilidad	Cobertura de recursos	Cobertura de escalabilidad
Reusabilidad	Personalización del servicio	Cobertura de variabilidad
	Características funcionales comunes	Propiedades funcionales comunes
	Características no funcionales comunes	Propiedades no funcionales comunes

En base a la definición de estas propiedades y métricas de calidad, haciendo uso de las reglas y consultas definidas en cada modelo semántico, se articularon los aspectos de calidad con la información mínima requerida para su evaluación dentro del proceso de simulación. De esta manera, del análisis de la información de calidad a nivel ontológico, se obtuvo un conjunto de propiedades de calidad susceptibles de ser evaluadas en etapa de diseño arquitectónico. Estos elementos corresponden a las filas

resaltadas en la Tabla 1. Estas propiedades fueron estudiadas a nivel de métricas a fin de definir los datos mínimos a ser relevados durante el proceso de simulación. La Tabla 2 resume el conjunto de indicadores elegidos como objetivo de simulación a fin de relevar las subcaracterísticas de calidad detalladas en el esquema de calidad web.

Tabla 2. Métricas que Definen el Objetivo de Simulación.

Variable	Descripción	Unidad
ET	Tiempo de procesamiento de solicitud de usuario.	Tiempo
TSIT	Tiempo total de solicitud (ingreso a respuesta).	Tiempo
TR	Cantidad total de solicitudes respondidas.	Solicitud
IR	Cantidad de solicitudes con respuesta erronea.	Solicitud
FT	Tiempo que el servicio estuvo inactivo por falla.	Tiempo
TT	Tiempo total que el servicio estuvo operativo.	Tiempo
FNF	Cantidad de defectos que no son fallas.	Defectos
TF	Cantidad total de defectos.	Defectos
RF	Cantidad de fallas reparadas en el sistema.	Fallas

3.1.2. Herramienta de Modelado Gráfico para el Diseño de Arquitecturas de Aplicaciones de Software

Teniendo en cuenta la rapidez con la que han evolucionado los productos basados en CC, es difícil establecer un conjunto específico de elementos que posibilite modelar cualquier tipo de solución arquitectónica. En este contexto, los autores de [Fehling et al. 2014] proponen un conjunto de patrones de diseño arquitectónico para entornos de CC que brindan una base de diseño a los arquitectos de aplicaciones de software web. Un patrón de diseño arquitectónico define una familia de sistemas en base a un esquema de organización estructural, determinando un vocabulario de componentes y conectores y un grupo de restricciones que indican la forma en la cual estos elementos pueden ser usados [Garlan and Shaw 1993]. Luego, los patrones propuestos en [Fehling et al. 2014] hacen uso de tres tipos de componentes arquitectónicos, a saber:

- *Componentes de administración*: Elementos que se encargan de gestionar el comportamiento de la aplicación de software web mediante el monitoreo automático de las réplicas de los elementos de software.
- *Componentes de aplicación*: Elementos que describen las funcionalidades de la aplicación de software. Según su comportamiento quedan divididos en: *i) Componentes definidos*: Elementos arquitectónicos básicos que son comunes a toda aplicación de software web (por ejemplo, el balanceador de carga). *ii) Componentes no definidos*: Elementos arquitectónicos de dominio que deben ser modelados por el arquitecto a fin de dar respuesta a una funcionalidad específica de la aplicación de software bajo desarrollo.
- *Componentes funcionales*: Elementos arquitectónicos que representan funciones básicas. Estos elementos se utilizan para modelar el comportamiento de los *componentes de aplicación no definidos*.

Teniendo en cuenta que los patrones definidos en [Fehling et al. 2014] indican la forma en la cual deben estructurarse estos componentes a fin de lograr diseños

arquitectónicos útiles y válidos, los componentes y conectores de estos patrones actúan como guías de diseño. Luego, el modelo de análisis y evaluación propuesto utiliza un *metamodelo de componentes y conectores* que ha sido definido en base al estudio de dichos patrones [Blas et al. 2019]. La definición de los elementos arquitectónicos se especifica en un diagrama de clases UML (Unified Modeling Language), mientras que mediante la definición de restricciones OCL (Object Constraint Language) se garantiza la consistencia de los diseños instanciados. De esta manera, el metamodelo propuesto posibilita el diseño y la validación de arquitecturas de software de aplicaciones web según las estructuras definidas en los patrones de diseño analizados.

Sin embargo, aunque el objetivo de cualquier descripción arquitectónica es brindar un mecanismo de soporte para el proceso de diseño, también es deseable que este mecanismo pueda llevarse a un formato computacional que permita su análisis y posterior instanciación [Albin 2003]. Por este motivo, en [Blas et al. 2019] se presenta una herramienta de software (implementada como un plug-in para la plataforma Eclipse) que facilita el modelado gráfico de arquitecturas de software web en base al metamodelo propuesto. La Figura 3 muestra una captura de pantalla de dicha herramienta, destacando el área de trabajo (sector 1, donde el arquitecto construye su diseño) y el conjunto de elementos disponibles (sector 2, paleta de componentes arquitectónicos).

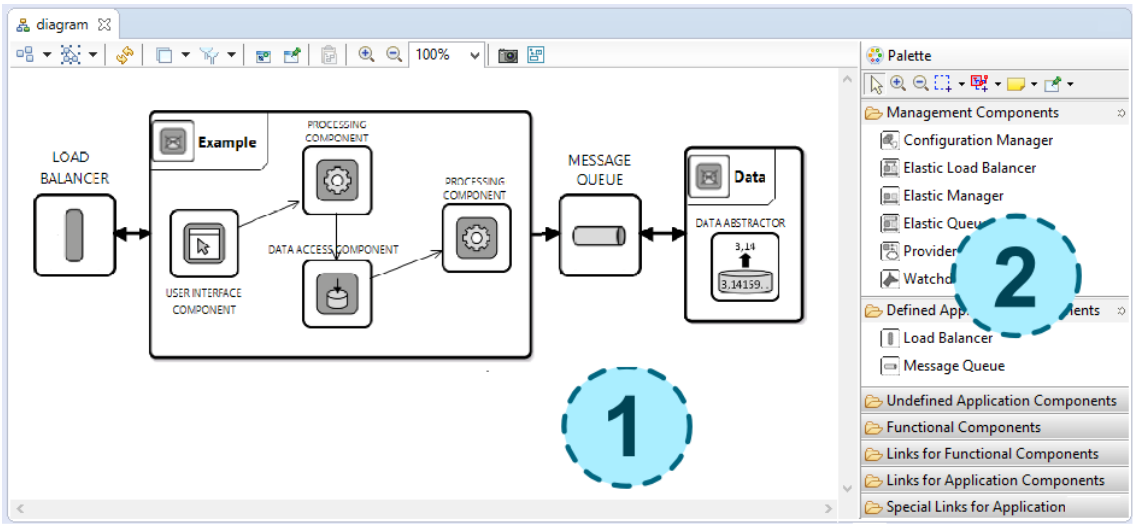

Figura 3. Herramienta gráfica para el diseño y validación de arquitecturas de software web que forma parte del modelo de análisis y evaluación arquitectónico.

3.1.3. Obtención de Modelos de Simulación basados en Eventos Discretos a partir del Diseño de la Arquitectura de Software

Tal como se ha enunciado con anterioridad, la formulación de los modelos de simulación no debe afectar el normal desarrollo de las tareas asociadas a los arquitectos de software. Luego, tomando como punto de partida los elementos arquitectónicos identificados en la representación arquitectónica (esto es, los elementos que componen el metamodelo UML-OCL), el modelo de evaluación incluye un conjunto de reglas de transformación *componente-a-modelo* que posibilitan la obtención automática de los modelos de simulación requeridos.

Cada modelo corresponde, individualmente, a un tipo de elemento arquitectónico. Teniendo en cuenta que los sistemas de software pueden ser vistos como sistemas basados en eventos (donde los cambios de estado en los componentes de software se producen en respuesta al arribo de solicitudes de usuario), en una primera instancia se planteó utilizar el formalismo de simulación DEVS [Zeigler et al. 2018] como base para la especificación de dichos modelos. Sin embargo, dado que las conexiones entre componentes arquitectónicos reflejan dependencias entre elementos, la obtención de los enlaces entre los modelos no es sencilla de derivar en DEVS. Por este motivo, la simulación de las solicitudes de usuario sobre componentes de software es tratada como un problema de ruteo. Bajo esta concepción, el modelo de simulación se diseña considerando que las solicitudes de usuario deben ser "ruteadas" dentro de los componentes de software definidos en la arquitectura.

En este contexto, el formalismo RDEVS [Blas et al. 2017a] propone una adaptación del formalismo DEVS que permite mantener las influencias entre componentes bajo la forma de acoplamientos fijos, pero al mismo tiempo, regula el flujo de eventos entre modelos en base a una función de ruteo. Los modelos RDEVS utilizan información de ruteo para determinar la aceptación o el rechazo de un determinado evento, quedando definidos según tres tipos de especificaciones: modelo de red, modelo de ruteo y modelo esencial. Un modelo de red se encuentra compuesto de un conjunto de modelos de ruteo. Un modelo de ruteo se compone de un modelo esencial junto con una función de ruteo que actúa como directriz para la recepción y el envío de eventos. Finalmente, un modelo esencial define el comportamiento de un componente de dominio. En el Anexo A se presenta una descripción más detallada de los modelos que componen el formalismo RDEVS. Haciendo uso de esta formalización, se definen los modelos de simulación para las arquitecturas de software web, donde los eventos a ser ruteados entre componentes corresponden a solicitudes de usuario.

La Figura 4 muestra un escenario de simulación para la arquitectura diagramada en la Figura 3. Este escenario muestra que el arquitecto ha decidido simular su diseño arquitectónico (partiendo de la definición gráfica) con un conjunto de réplicas preestablecidas para los componentes de dominio propios de la aplicación de software (los cuales, en este caso, corresponden a los componentes denominados *Example* y *Data*). Esta definición incluye un *Load Balancer* (componente de aplicación definido), tres réplicas del componente *Example* (componente de aplicación no definido), un *Message Queue* (componente de aplicación definido) y dos réplicas del componente *Data* (componente de aplicación no definido).

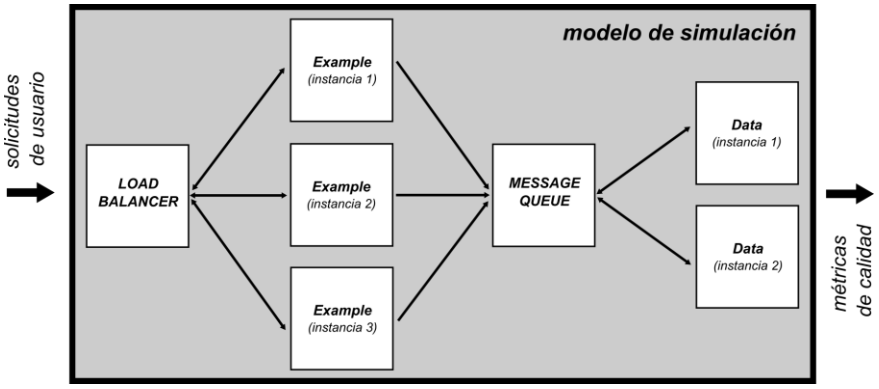

Figura 4. Ejemplo de estructura de escenario de simulación.

Tal como se ha enunciado con anterioridad, el tipo de componente de aplicación define el conjunto de transformaciones a aplicar. Para el caso de la arquitectura completa, el modelo de simulación final es definido como un modelo de red RDEVS. Siguiendo el ejemplo de la Figura 4, este modelo quedará compuesto de 7 modelos de ruteo RDEVS que encapsulan cada uno de los elementos definidos en el escenario. Luego, cada uno de estos 7 modelos definirá un nodo de la red. Todo nodo está asociado a un modelo esencial RDEVS que especifica su comportamiento. El comportamiento de cada nodo queda definido según el tipo de componente arquitectónico que se ha asociado al mismo.

Debido a que los *componentes de aplicación definidos* corresponden a elementos comúnmente utilizados en las arquitecturas de aplicaciones de software web (como es, por ejemplo, el caso de *Load Balancer* y *Message Queue*), los modelos esenciales RDEVS que representan estos comportamientos fueron predefinidos durante el proceso de transformación (manteniendo el objetivo de relevar las métricas de calidad propuestas). Sin embargo, en el caso de los *componentes de aplicación no definidos* (es decir, aquellos componentes propios del dominio de la aplicación bajo desarrollo cuyo comportamiento es definido en base a *componentes funcionales*), es necesario obtener una descripción de comportamiento a partir del diseño especificado por el arquitecto. Este es el caso de los componentes *Example* y *Data*. Luego, la descripción de comportamiento se genera en base al flujo de ejecución detallado en la secuencia de *componentes funcionales* que componen la definición del elemento en el diseño arquitectónico.

En términos generales, un modelo esencial RDEVS se comporta como una máquina de estado dentro de la cual se producen transiciones entre un estado y el siguiente. Todos los estados incluyen una fase. Cuando arriba un evento (evento de entrada), el modelo ejecuta una función de transición externa que indica cual será el nuevo estado teniendo en cuenta el estado actual y el evento recibido. Una vez vencido el tiempo de permanencia en un estado, se produce una transición (por medio de la ejecución de una función de transición interna) que tiene como consecuencia la generación de un evento de salida. De esta manera, el comportamiento del modelo evoluciona en el tiempo de acuerdo con la ocurrencia de eventos. Haciendo uso de esta dinámica, en base al conjunto de *componentes funcionales* definidos en el diseño de cada *componente de aplicación no definido*, se genera este comportamiento.

Para cada componente funcional incluido en el flujo de ejecución de un componente arquitectónico se detallan dos posibles fases: *procesando* y *procesando con funcionamiento defectuoso*. Además, a nivel de componente de aplicación se definen fases comunes a todos los componentes funcionales, a saber: *esperando solicitudes*, *en estado de falla* e *inactivo*. La secuencia de estados definida en base a estas fases permite simular el comportamiento de los *componentes de aplicación no definidos* de acuerdo a los siguientes casos: *i)* cuando el componente está activo esperando solicitudes (es decir, el componente de software está asignado a recursos de infraestructura pero no se encuentra procesando solicitudes), *ii)* cuando el componente procesa solicitudes entregando las respuestas esperadas, cuando el componente procesa solicitudes con respuestas posiblemente incorrectas, *iii)* cuando el componente no responde a las solicitudes debido a algún fallo de procesamiento y, por último, *iv)* cuando el componente se encuentra inactivo (es decir, el componente de software no está asignado a recursos de infraestructura).

A modo de ejemplo, la Figura 5 presenta una versión simplificada de un diagrama de estados que ilustra la secuencia de estados resultante de la transformación *componente-a-modelo* ejecutada sobre la definición funcional del componente de aplicación *Example*.

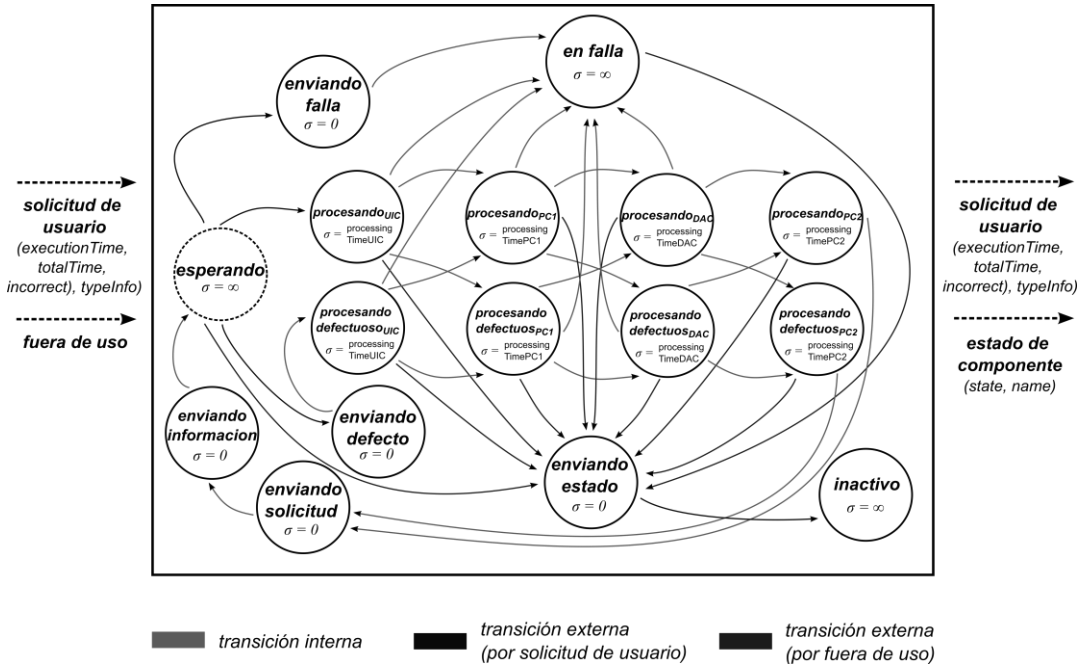

Figura 5. Diagrama de estados simplificado generado en base a la descripción de comportamiento del componente "Example".

Luego, los elementos que componen el metamodelo son asociados de forma directa a modelos de comportamiento (en el caso de *componentes de aplicación definidos*) o, en su defecto, se derivan modelos de comportamiento tomando en consideración las responsabilidades arquitectónicas identificadas (es decir, el flujo de componentes funcionales que forman los componentes de aplicación no definidos). Así, la transformación *componente-a-modelo* posibilita que la estructura del modelo de simulación final sea similar al diseño arquitectónico. Esto conlleva dos ventajas para los arquitectos de software:

- Comprensión del modelo (aun cuando no tengan conocimientos específicos de los modelos de simulación RDEVS).
- Visión del modelo (utilizan el modelo para estimar la calidad sin percibirlo como una barrera u obstáculo que ralentiza el proceso de desarrollo).

El proceso de transformación se logra a partir de una implementación de modelos RDEVS desarrollada en Java que mantiene compatibilidad con la herramienta de modelado arquitectónico propuesta en la Sección 3.1.2. Durante el proceso de transformación, los modelos de simulación relevan las métricas de calidad definidas en la Tabla 2 por medio de la medición de los tiempos asociados a los diferentes estados y a la propagación de eventos específicos ante la ocurrencia de defectos y/o fallos. Con esta integración se fomenta la transformación automática de los componentes arquitectónicos a modelos de simulación RDEVS cuya ejecución brinda una estimación de las métricas de calidad propuestas.

3.1.4. Generación de Eventos de Entrada a partir de Modelos de Simulación de Comportamiento de Usuario

Dado que el formalismo RDEVS se corresponde con una adaptación del formalismo DEVS, los modelos de componentes arquitectónicos diseñados en RDEVS pueden ser combinados con modelos DEVS como parte de un mismo proceso de simulación. En este contexto, como complemento del enfoque de simulación RDEVS definido para la arquitectura de software de aplicaciones web, un mecanismo de entrada basado en DEVS fue definido. Su objetivo es generar las solicitudes de usuario a ser procesadas como parte de la simulación.

Según [Fehling et al. 2014], la carga de trabajo refiere al uso de los recursos de tecnología de la información en los que se aloja una aplicación de software. Su resultado puede verse como tipos de patrones temporales que se evidencian de acuerdo con la forma en la cual arriban las solicitudes de los usuarios a la aplicación de software bajo estudio. Haciendo uso de este enfoque, en [Blas et al. 2017c] se presenta un esquema de modelos de simulación DEVS diseñados e implementados con el objetivo de crear eventos (asociados a solicitudes de usuarios) según distintos tipos de patrones temporales. Estos modelos abordan, específicamente, cargas de trabajo estáticas, periódicas, únicas, impredecibles y de cambio continuo.

El esquema genérico diseñado para tales modelos de simulación combina modelos continuos (para la definición de la señal de carga de trabajo) con modelos de eventos discretos (para la generación de los eventos referidos a las solicitudes de usuario). De esta manera, se genera un modelo de simulación híbrido general que permite representar distintos tipos de comportamientos según la señal elegida para el patrón de carga de trabajo. Como resultado, los autores obtienen un conjunto altamente configurable de modelos de eventos discretos basados en DEVS que permiten simular el comportamiento de diferentes tipos de usuarios.

El modelo de análisis y evaluación propuesto hace uso de estos modelos de usuario altamente configurables tomando como punto de partida su especificación DEVS. Estos modelos de simulación fueron encapsulados dentro de la herramienta Java en un nuevo módulo denominado “eventos de entrada”. De esta manera, el arquitecto puede elegir el tipo de usuario con el cual desea evaluar su diseño.

Luego, el modelo brinda la posibilidad de estimar la calidad sobre la arquitectura de la aplicación de software según solicitudes producidas por diferentes tipos de usuario.

3.2. Discusión y Prueba de Conceptos: Evaluación de Arquitecturas Genéricas para Aplicaciones de Software

La validación del modelo de análisis y evaluación propuesto fue realizada tomando como referencia los diseños más comunes de arquitecturas de software asociadas a aplicaciones web. Específicamente se evaluaron dos esquemas arquitectónicos: arquitectura basada en dos bandas y arquitectura basada en tres bandas. En [Fehling et al. 2014] pueden encontrarse los detalles referidos a la estructura de estos diseños.

En base a la generación de múltiples escenarios alternativos, se compararon los desempeños de ambas arquitecturas respecto a los comportamientos esperados como parte de su definición teórica. Para cada caso se generaron los modelos de simulación

arquitectónica RDEVS equivalentes al diseño propuesto, con el objetivo de relevar los valores asociados a las métricas de calidad definidas. La estructura de ejecución elegida para las pruebas fue establecida en base a una cantidad fija de réplicas para los *componentes de aplicación no definidos*. Esto permitió especificar escenarios de simulación reales.

Todas las réplicas de un mismo componente se configuraron con los mismos parámetros a fin de garantizar una semejanza de comportamiento en los componentes arquitectónicos definidos en el nivel más interno (esto es, el comportamiento de las responsabilidades arquitectónicas definidas en los *componentes funcionales*). Además, para cada caso analizado en relación con un mismo escenario, se mantuvo fija la configuración interna de los modelos de generación de solicitudes de usuario en virtud de obtener resultados comparables.

Los criterios utilizados durante las pruebas se clasificaron en dos grandes grupos: pruebas basadas en la cantidad de réplicas disponibles y pruebas basadas en las probabilidades de defecto y/o falla en los componentes de software. En el primer caso, se analizaron: *i)* la cantidad de solicitudes que son atendidas en la aplicación, *ii)* la disponibilidad de los distintos componentes, y *iii)* la forma en la cual estos componentes entran en estados de defecto y/o falla como consecuencia del proceso de ejecución. En cambio, en el segundo caso se generaron escenarios con distintas probabilidades de falla y/o defecto en un conjunto fijo de réplicas de componentes de aplicación. En estos escenarios, el objetivo fue estudiar la forma en la cual responden los elementos de software a los estados de falla. Además, se analizó junto con la cantidad de solicitudes resueltas, la demora que tiene la aplicación en dejar de responder las solicitudes de usuario (es decir, en quedar inactiva).

La Tabla 3 resume las configuraciones utilizadas (detallando el número de réplicas y la probabilidad de defecto/falla seteada) junto con la cantidad de modelos de simulación ejecutados en cada caso.

Tabla 3. Configuraciones Utilizadas durante la Prueba de Conceptos del Ambiente de Evaluación.

<table>
<tr><th rowspan="2"># Réplicas</th><th rowspan="2">Probabilidad Defecto/Falla</th><th colspan="2"># Modelos</th></tr>
<tr><th>Arq. 2 bandas</th><th>Arq. 3 bandas</th></tr>
<tr><td>5</td><td rowspan="4">0</td><td>9</td><td>25</td></tr>
<tr><td>10</td><td>14</td><td>40</td></tr>
<tr><td>20</td><td>24</td><td>70</td></tr>
<tr><td>50</td><td>54</td><td>85</td></tr>
<tr><td rowspan="2">25</td><td>0.5</td><td rowspan="2">29</td><td>160</td></tr>
<tr><td>1</td><td></td></tr>
</table>

En todos los casos de análisis modelados y simulados en el ambiente de evaluación, los modelos obtenidos del proceso de transformación reflejaron un comportamiento consistente con los patrones que representan. A modo de ejemplo, la Figura 6 presenta la comparativa de la cantidad promedio de solicitudes (métrica TR, *total number of requests*) que han sido procesadas (eje y) en el modelo de simulación de la aplicación web para ambas arquitecturas según la cantidad de réplicas existentes (eje x). Como puede observase, cuando existen únicamente 5 réplicas de los componentes de

aplicación no definidos es evidente que el modelo de la arquitectura de tres bandas tiene un mejor desempeño para el procesamiento general de solicitudes. En menor escala, se presenta una situación similar cuando existen 10 réplicas. En ambos casos, el comportamiento se debe a que el diseño estructural de una arquitectura de 3 bandas presenta una separación entre la presentación y la lógica de negocios que posibilita la atención de nuevas solicitudes por parte de cualquier réplica disponible. Debido a que esto no ocurre en la arquitectura de dos bandas, su desempeño es peor. Asimismo, el valor de TR en los casos de 20 y 50 réplicas es similar para ambos modelos. Esta similitud se debe a que la configuración de los modelos no admite mayor escalabilidad de solicitudes.

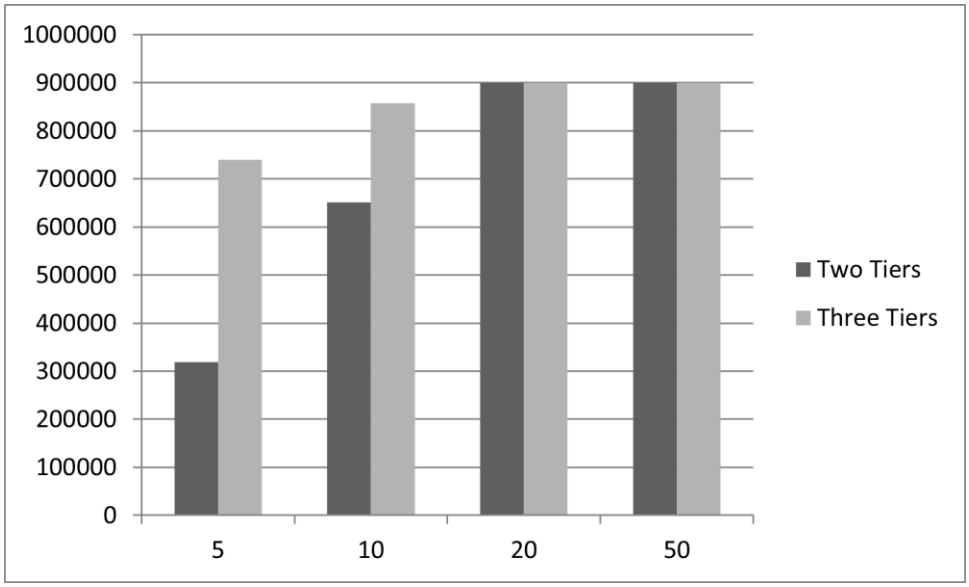

Figura 6. Comparativa del promedio de solicitudes resueltas en múltiples simulaciones de las arquitecturas de software web analizadas.

Luego, los resultados obtenidos de la prueba de conceptos realizada dan lugar a la validación del modelo integral de análisis y evaluación implementado.

4. Conclusiones

La evaluación de la calidad esperada en un producto de software se ha convertido en un tema de interés de la Ingeniería de Software. En base a las propiedades de calidad requeridas para estimar el comportamiento de una aplicación de software web, en este trabajo se ha presentado un modelo de análisis y evaluación integral que ha sido diseñado en base a un conjunto de nuevos artefactos de software. Este modelo, de forma complementaria a las estrategias tradicionales, da lugar a una nueva forma de evaluación de calidad basada en simulación. La ejecución de los modelos de simulación obtenidos a partir de la transformación de los componentes de diseño en estados, transiciones y eventos; da lugar a una evaluación cuantitativa de la calidad esperada.

Existen múltiples ventajas de emplear simulación en la etapa de diseño arquitectónico, entre las que se destacan: *i)* reducción del tiempo de desarrollo, y *ii)* verificación de las decisiones arquitectónicas en escenarios artificiales. De esta manera, el dinamismo con el cual se generan los modelos de simulación haciendo uso del modelo propuesto contribuye, no sólo a la estimación de la calidad futura, sino también

al estudio de múltiples alternativas de diseño según los requerimientos no funcionales del producto de software.

El trabajo de investigación realizado impacta en aspectos tecnológicos y económicos vinculados a la industria del software, específicamente en el área de CC. La aplicación de los modelos y herramientas propuestas contribuye a mejorar los resultados obtenidos en el proceso de desarrollo, permitiendo evaluar la calidad en etapas tempranas de trabajo. En este punto, es importante destacar que el impacto de la propuesta no sólo beneficia la etapa de diseño arquitectónico, sino que además brinda una estimación de la calidad esperada sobre el producto final. Si se reformula la arquitectura, es posible evaluar el impacto de los cambios introducidos en relación con la estimación previa.

Algunas de las líneas de investigación futuras que se desprenden de este trabajo incluyen la utilización de esquemas de calidad para el estudio de otros artefactos de software, la incorporación de componentes de infraestructura para una simulación conjunta de entornos de CC y, eventualmente, la evaluación de arquitecturas de sistemas auto-adaptativos mediante la reformulación automática de la estructura de los modelos de simulación.

Anexo A. Formalismo Routed DEVS

El formalismo DEVS es un formalismo de modelado basado en la teoría de sistemas que proporciona una metodología general para la construcción jerárquica de modelos de simulación reutilizables de forma modular [Zeigler et al. 2018]. Recientemente, se ha desarrollado el formalismo RDEVS [Blas et al. 2017a] como una nueva extensión del formalismo DEVS que facilita el modelado y simulación de problemas centrados en el ruteo de eventos sobre modelos de simulación discretos. De esta manera, los modelos especificados en este formalismo buscan dar solución a la identificación de eventos como funcionalidad embebida dentro de modelos de simulación basados en eventos discretos. Para esto, el formalismo RDEVS define tres modelos de simulación: modelo esencial, modelo de ruteo y modelo de red. La definición formal de estos modelos haciendo uso de teoría de conjuntos puede ser consultada en [Blas et al. 2017a].

En RDEVS, cada modelo representa un nivel de abstracción utilizado para conceptualizar los distintos elementos que conforman un problema de ruteo. Un problema de ruteo es definido como el problema que surge cuando existe la necesidad de identificar el origen y/o indicar el destino de eventos a fin de garantizar su correcto procesamiento. Para resolver este problema, en RDEVS todo el proceso de ruteo es modelado en base a un único modelo de red. Cada nodo de este proceso se representa con un modelo de ruteo. Estos modelos se encuentran vinculados por acoplamientos todos contra todos (excepto si mismos) a fin de respetar la definición del formalismo original (es decir, DEVS). Luego, cada modelo de ruteo estructura una entidad específica dentro del proceso de ruteo integrando un modelo esencial (esto es, un componente de dominio) con su propia función de ruteo. Teniendo en cuenta que un modelo esencial se define como un modelo DEVS atómico, el conjunto de componentes requeridos como parte de la simulación de un proceso de ruteo puede ser modelado en DEVS y, posteriormente, se puede aplicar el formalismo RDEVS sobre estos modelos para resolver el ruteo. Para mayores detalles respecto al formalismo se sugiere consultar [Blas et al. 2017a].

Referencias

Albin, S. T. (2003). The Art of Software Architecture: Design Methods and Techniques, John Wiley & Sons.

Bachmann, F., Bass, L., Klein, M. and Shelton, C. (2005). Designing Software Architectures to Achieve Quality Attribute Requirements. In *Software*, vol. 152, no. 4, pages 153-165. IEEE.

Barbacci, M., Klein, M., Longstaff, T. and Weinstock, C. (1995). Quality Attributes (CMU/SEI-95-TR-021), Carnegie-Mellon University.

Bass, L., Clements, P. and Kazman, R. (2013). Software Architecture in Practice, Addison-Wesley, 3rd Edition.

Blas, M. J., Gonnet, S. and Leone, H. (2017a). Routing Structure over Discrete Event System Specification: A DEVS Adaptation to Develop Smart Routing in Simulation Models. In *Proceedings of the 2017 Winter Simulation Conference*, pages 774-785. Piscataway, New Jersey: IEEE.

Blas, M. J., Gonnet, S. and Leone, H. (2017b). An Ontology to Document a Quality Scheme Specification of a Software Product. In *Expert Systems*, vol. 34, no. 5, https://doi.org/10.1111/exsy.12213.

Blas, M. J., Gonnet, S. and Leone, H. (2017c). Modeling User Temporal Behaviors Using Hybrid Simulation Models. In *IEEE Latin America Transactions*, vol. 15, no. 2, pages 341-348.

Blas, M. J., Leone, H. and Gonnet, S. (2019). Modelado y Verificación de Patrones de Diseño de Arquitectura de Software para Entornos de Computación en la Nube. In *Revista Ibérica de Sistemas e Tecnologias de Informação*, vol. 35, http://dx.doi.org/10.17013/risti.35.1-17.

Breu, R., Kuntzmann-Combelles, A. and Felderer, M. (2014). New Perspectives on Software Quality. In *IEEE Software*, vol. 31, no. 1, pages 32-38.

Clements, P., Kazman, R. and Klein, M. (2001). Evaluating Software Architectures: Methods and Case Studies, Addison-Wesley.

Crnkovic, I., Larsson, M. and Preiss, O. (2005). Concerning Predictability in Dependable Component-based Systems: Classification of Quality Attributes. In *Architecting Dependable Systems (Lecture Notes in Computer Science)*, vol. 3549. Springer.

Fehling, C., Leymann, F., Retter, R., Schupeck, W. and Arbitter, P. (2014). Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications, Springer Science & Business Media.

Garlan, D., and Shaw, M. (1993). An Introduction to Software Architecture. In *Advances in Software Engineering and Knowledge Engineering*, pages 1-39.

Hasselbring, W. (2018). Software Architecture: Past, Present, Future. In *The Essence of Software Engineering*, pages 169-184.

Hussain, M., and Abdulsalam, H. M. (2014). Software Quality in the Clouds: A Cloud-based Solution. In *Cluster Computing*, vol. 17, no. 2, pages 389-402.

IEEE 1061 (1998). Software Quality Metrics Methodology: Description, Institute of Electrical Electronic Engineering.

ISO/IEC 25010 (2011). Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models, Institute of Electrical Electronic Engineering.

Jadeja, Y., and Modi, K. (2012). Cloud Computing - Concepts, Architecture and Challenges. In *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877-880.

Khan, A., Kiah, M., Khan, S. and Madani, S. A. (2013). Towards Secure Mobile Cloud Computing: A Survey. In *Future Generation Computer Systems*, vol. 29, no. 5, pages 1278-1299.

Kitchenham, B. and Pfleeger, S. L. (1996). Software Quality: The Elusive Target. In *IEEE Software*, vol. 13, no. 1, pages 12-21.

Koziolek, H., Becker, S., Reussner, R. and Happe, J. (2009). Evaluating Performance of Software Architecture Models with the Palladio Component Model. In *Software Applications: Concepts, Methodologies, Tools, and Applications*, pages 1111-1134.

Law, A. M. and Kelton, W. D. (1991). Simulation Modeling and Analysis, McGraw-Hill.

Lewis, G. (2010). Basics About Cloud Computing. Carniege-Mellon University.

Luo, J. Z., Jin, J. H., Song, A. B. and Dong, F. (2011). Cloud Computing: Architecture and Key Technologies, In *Journal of China Institute of Communications*, vol. 32, no. 7, pages 3-21.

Meiappane, A., Chithra, B. and Venkataesan, P. (2013). Evaluation of Software Architecture Quality Attribute for an Internet Banking System. In *International Journal of Computer Applications*, vol. 62, no. 19, pages 21-24.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing.

Miguel, J. P., Mauricio, D., and Rodríguez, G. (2014). A Review of Software Quality Models for the Evaluation of Software Products. In *International Journal of Software Engineering & Applications*, pages 31-53.

Nistala, P., Nori, K. V., and Reddy, R. (2019). Software Quality Models: A Systematic Mapping Study. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 125-134, IEEE.

Odun-Ayo, I., Ananya, M., Agono, F., and Goddy-Worlu, R. (2018). Cloud Computing Architecture: A Critical Analysis. In *18th International Conference on Computational Science and Applications (ICCSA)*, pages 1-7, IEEE.

Pressman, R. and Maxim, B. (2020). Software Engineering: A Practitioner's Approach, McGraw Hill, 9th Edition.

Rempel, P. and Mäder, P. (2017). Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality. In *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pages 777-797, https://doi.org/10.1109/TSE.2016.2622264.

Reussner, R.H., Becker, S., Happe, J., Heinrich, R., Koziolek, A., Koziolek, H., Kramer, M. and Krogmann, K. (2016). Modeling and Simulating Software Architectures: The Palladio Approach, MIT Press.

Sabri, O. and Alfifi, F. (2017). Integrating Knowledge Life Cycle within Software Development Process to Produce a Quality Software Product. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1-7, https://doi.org/10.1109/ICEngTechnol.2017.8308172.

Schneider, Y., Busch, A. and Koziolek, A. (2018). Using Informal Knowledge for Improving Software Quality Trade-off Decisions. In *European Conference on Software Architecture*, pages 265-283, Springer.

Tian, J. (2005). Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, John Wiley & Sons.

Van Hoorn, A., Waller, J. and Hasselbring, W. (2012). Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, pages 247–248, ACM.

Zalewski, A. and Kijas, S. (2013). Beyond ATAM: Early Architecture Evaluation Method for Large-Scale Distributed Systems. In *Journal of Systems and Software*, vol. 86, no. 3, pages 683-697.

Zeigler, B. P., Muzy, A. and Kofman, E. (2018). Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations, Academic Press.