Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte

Pattern Languages Software Architecture: An Approach To State of the Art

Victor Hugo Jimenez-Torres[1], Wilman Tello-Borja[2], Jorge Iván Rios-Patiño[3].
[1,2,3]*Maestría en Ingeniería de Sistemas y Computación, Facultad de Ingenierías, Universidad Tecnológica de Pereira, Pereira, Colombia.*
wilman@utp.edu.co
vjimene@utp.edu.co
jirios@utp.edu.co

Resumen— **El propósito principal de este artículo es el de mostrar el estado del arte en un área de la arquitectura de software llamada "Lenguajes de Patrones", desde sus orígenes, los avances actuales y sus aplicaciones en la construcción de arquitecturas de software en diferentes dominios de aplicación.**

Este último aspecto es relevante ya que como se verá en este artículo, la extensibilidad y aplicabilidad de los lenguajes de patrones a diferentes dominios se convierte en una herramienta importante para diseñadores y desarrolladores de diversos tipos de sistemas de información.

Palabras clave— **Patrones de Arquitectura, Lenguajes de Patrones de Arquitectura, Arquitectura de Software.**

***Abstract*— The main purpose of this paper is to show the state of the art about the area of software architecture known as "Pattern Languages", based on their origins, current advances and their application in constructing software architectures to various information systems in specific domains.**
This last aspect is about great importance because as you will see later in this article, the extensibility and applicability of the language patterns to different domains becomes an important tool for designers and builders of all types of information systems.

***Keywords*— Architecture Patterns, architecture Pattern Leguages, Software Arquitecture**

I. INTRODUCCIÓN

La programación de la producción, o scheduling, es una El propósito principal del presente trabajo es mostrar el estado del arte en investigación del área de la Arquitectura de Software conocida como "Lenguajes de Patrones". Partiendo de sus orígenes, llegando hasta los avances actuales y analizando su aplicación en la construcción de arquitecturas de software para diversos sistemas de información en dominios específicos. Este último aspecto es de gran importancia, ya que como se verá más adelante en este artículo, la extensibilidad y aplicabilidad de los Lenguajes de Patrones, se convierte en una herramienta importante para diseñadores e implementadores de todo tipo de sistemas de información.

La Arquitectura de Software en general, se define como un nuevo paradigma o nueva forma de ver un sistema de información desde un punto de vista holístico, donde cada componente afecta los requerimientos fundamentales, ya sean funcionales o no funcionales. Esta disciplina aborda estos requerimientos rigurosamente y garantiza un buen diseño de la aplicación final, lo que redunda en mejor calidad, un mayor retorno de inversión de los proyectos y garantiza una mejor mantenibilidad de los sistemas de información construidos. George Fairbanks [1] define la Arquitectura de Software como: La ciencia que trata el diseño de un sistema de información y del impacto de sus cualidades como: desempeño, seguridad, modificabilidad entre otras.

Como fuentes de consulta primarias para la redacción de este artículo, la información se obtuvo accediendo a bases de datos indexadas, libros y artículos académicos.

II. De la Ingeniería de Software a la Arquitectura de Software

La unidad de estudio es una pequeña empresa de calzado La Ingeniería de Software tuvo sus inicios en la década de los sesenta y a partir de allí a evolucionado de forma constante a través de las últimas seis décadas. En los años cincuenta se inició con el intentó imitar la ingeniería de hardware y se pensó en ese entonces que la utilización del método científico brindaría un soporte investigativo suficiente para obtener un proceso técnico coherente. Finalizando el siglo XX la ingeniería evolucionó como un área de estudio más estructurada y con fundamentos

Fecha de Recepción: 02 de Mayo de 2014
Fecha de Aceptación: 10 de Noviembre de 2014

teóricos fuertes. Desde entonces la voluntad de los ingenieros fue de mejorar la calidad del software construido y de dar herramientas que facilitaran el trabajo a diseñadores e implementadores de sistemas de información en general.

Quizá uno de los científicos que dio origen a la Ingeniería de Software fue Barry Boehm quien definió la Ingeniería de Software como*: "... es la aplicación de la ciencia y las matemáticas a la construcción del software de tal forma que sus propiedades lo hacen útil para las personas". Este ingeniero estadunidense en su artículo científico titulado: "*A View of 20th and 21st Century Software Engineering" muestra la evolución de la Ingeniería de Software en las últimas décadas, partiendo de la década de los 50's hasta finales del siglo XX. Se pueden rescatar los siguientes aspectos para cada una de las épocas descritas [12]:

Años Cincuenta: Se aplica al desarrollo de software el mismo proceso que al desarrollo de hardware, tipo cascada rigurosa.

Buenos Principios

No ignorar las matemáticas, las ciencias de la computación, las ciencias sociales, las ciencias económicas y administrativas.

Usar el método científico para aprender a través de la experiencia.

No comprometerse demasiado antes de entender la complejidad de un proyecto

Evitar seguir rigurosamente el proceso de desarrollo secuencial.

Años Sesenta: El desarrollo de software es artesanal. Las propiedades de software, tales como: fácil de modificar, fácil de copiar, no se gasta, es invisible, fomentaron el proceso de desarrollo tipo "codifica y corrige" (code and fix). Se inició la cultura del hacker en el buen sentido de la palabra, es decir experto en programación, y la del vaquero (cowboy) que hace desarrollos heroicos de última hora.

Buenos Principios

Atreverse a hacer prototipos novedosos, no limitarse a repetir lo que ya se conoce.

Respetar que el software es diferente. No se puede incrementar la velocidad de su desarrollo de manera infinita.

Evitar la programación al estilo vaquero. Parches de último minuto o trabajo de última noche que pueden traer graves consecuencias.

Años Setentas: Se identifican las diferentes fases del desarrollo: requerimientos, análisis, diseño, codificación y pruebas. Se introduce la programación estructurada y métodos formales para especificar software. Se identifican principios de diseño, como modularidad, encapsulación, abstracción de tipos de datos, acoplamiento débil y alta cohesión, entre otros. Se publica el modelo de cascada y se definen los conceptos de verificación y validación.

Buenos Principios

Eliminación temprana de los defectos y su prevención a través del análisis de causa.

Determinación temprana del propósito de sistema para tener una visión compartida con el cliente.

Evitar desarrollo descendente (top-down) a toda costa. Los requerimientos emergentes y los cambios lo hacen poco realista, para la mayoría de los casos.

Años Ochenta: Se busca la productividad y escalabilidad de sistemas y equipos de desarrollo. La Orientación a Objetos renace con fuerza a través de las múltiples propuestas de lenguajes de programación. Se crea el primer modelo de madurez de capacidades de procesos (SW-CMM) y los primeros estándares. Nace el concepto de Fábricas de Software y se generan las primeras herramientas para incrementar la productividad a través de la programación por el usuario, tales como 4GLs.

Buenos Principios

Hay muchos caminos para incrementar la productividad que incluyen la selección del personal, capacitación, herramientas, reutilización, mejora de procesos, entre otros.

Lo que es bueno para el producto es bueno para el proceso, por ejemplo: arquitectura, composición y adaptación.

Evitar pensar que existe una solución mágica (silver bullet) que aplica a toda clase de problemas.

Años Noventa: La concurrencia (paralelismo y distribución) adquiere mayor importancia con respecto a procesos secuenciales. La Orientación a Objetos se extiende a las fases de análisis y diseño. Se acuerda un lenguaje de modelado (UML) y se genera el primer proceso comercial de desarrollo orientado a objetos (RUP). Los diseñadores y los arquitectos de software

empiezan a recaudar las mejores experiencias a través de patrones de diseño y de arquitectura. Se define el Modelo Espiral para el desarrollo basado en el análisis de riesgos y su vertiente conocida como desarrollo iterativo e incremental. El Software Libre toma fuerza y se crean los primeros ejemplos exitosos. La usabilidad de sistemas se convierte en el foco de atención e investigación. Software empieza a ocupar la posición crítica en el mercado competitivo y en la sociedad (web).

Buenos Principios

El tiempo es dinero. La gente invierte en software esperando retorno de inversión, mientras más rápido se desarrolle el software, más rápido se recupera la inversión, pero eso pasa sólo en el caso cuando la calidad de software es satisfactoria.

El software tiene que ser útil para la gente, es la parte crucial de la definición de Ingeniería.

Evitar hacer las cosas demasiado rápido. Los hitos muy ambiciosos a menudo traen como consecuencia las especificaciones incompletas, que resultan en re-procesos.

Retos Actuales: Arquitecturas adaptativas, calidad de software y líneas de producción de software, Model Driven Architectures (MDA), Arquitecturas de Software orientadas a servicios y Arquitecturas de software orientadas a la Nube.

III. La Investigación en Ingeniería de Software

Dado que la Ingeniería de Software como la Arquitectura de Software son áreas de estudio relativamente nuevas en comparación con las ciencias básicas y otras ingenierías. Vale la pena mencionar cómo han evolucionado en materia de investigación y qué desafíos presentan en un futuro cercano. En el caso de la Ingeniería de Software, Esperanza Marcos [2] hace una comparación, entre la investigación básica y la investigación ingenieril. Afirma por ejemplo la autora: "El objeto de estudio en las ingenierías (y en particular en la Ingeniería de Software), difiere del objeto de estudio de las ciencias formales, humanas y naturales. Mientras estas ciencias se ocupan de estudiar fenómenos u objetos ya existentes, las ciencias de la ingeniería se ocupan de estudiar los métodos y técnicas para la creación de nuevos objetos e incluso de crear estos métodos y técnicas" [3]

La Ingeniería de Software como tal se preocupa más del método y los principios de diseño en el proceso de construcción del software. Por otro lado la Arquitectura de Software se centra en los aspectos del cómo se debe construir el sistema, siempre pensando en los atributos de calidad inherentes al mismo y sobre todo los atributos que satisfacen completamente al usuario final.

En la conferencia de la NATO de 1969, un año después de la sesión en que se fundara la Ingeniería de Software, P. I. Sharp formuló estas sorprendentes apreciaciones comentando las ideas de Dijkstra [13]: "Pienso que tenemos algo, aparte de la Ingeniería de Software: algo de lo que hemos hablado muy poco pero que deberíamos poner sobre el tapete y concentrar la atención en ello. Es la cuestión de la Arquitectura de Software. La arquitectura es diferente de la Ingeniería. Como ejemplo de lo que quiero decir, echemos una mirada a OS/360. Partes de OS/360 están extremadamente bien codificadas. Partes de OS, si vamos al detalle, han utilizado técnicas que hemos acordado constituyen buena práctica de programación. La razón de que OS sea un amontonamiento amorfo de programas es que no tuvo arquitecto. Su diseño fue delegado a series de grupos de ingenieros, cada uno de los cuales inventó su propia arquitectura. Y cuando esos pedazos se clavaron todos juntos no produjeron una tersa y bella pieza de software" [NATO76: 150].

La creación de nuevas formas de hacer las cosas que redunden en mayor productividad y mayor calidad en los productos de software, permite que problemas cada vez más complejos puedan ser resueltos y estas soluciones sean replicables o generalizables en un dominio determinado, lo que brinda una constante extensión de conocimiento y adentrándonos en la "Arquitectura de Software", Este enfoque de generalización da origen a un término ya ampliamente conocido como "Patrón de Diseño".

IV. Los Patrones de Diseño y los Lenguajes de Patrones

Quizá el autor más referenciado cuando se inicia un proceso de revisión de literatura acerca de Arquitectura de Software y en particular desde lo básico, es Christopher Alexander, este arquitecto de construcciones civiles fue el que inicialmente acuñó este término en sus trabajos iniciales [4].En su trabajo investigativo Christopher Alexander propuso una aproximación sistemática a los problemas comunes en la arquitectura civil, donde propone una descomposición analítica de cada problema en sub problemas, cada uno de ellos caracterizado por un grupo de fuerzas en competencia, donde sí se resuelven las fuerzas que atacan cada sub problema y se sintetiza la solución individual, el arquitecto genera una solución genérica al problema general [5].

En otro de sus trabajos [6], Christopher Alexander propone patrones específicos y la posibilidad de generar instancias de estos. Y es allí cuando introduce el concepto del "Lenguajes de Patrones de Arquitectura".

Un lenguaje de patrones de Alexander, comienza con el nombre y número de referencia del mismo, ejemplo: LUCES A LOS DOS LADOS DE CADA HABITACIÓN (Patrón 159). El nombre del patrón es conciso, evocativo pero no ambiguo [7].En este caso el problema atacado por el patrón es: "Cuando las personas pueden elegir, ellas siempre van circular por las habitaciones que tienen luces a ambos lados de la misma, y dejarán vacía las habitaciones que sólo tienen una luz a un lado de la misma" [8].

Este conciso enunciado, sigue a una detallada discusión de la razón del problema que incluye evidencia empírica y la descripción de las fuerzas en competencia mencionadas anteriormente. Todo esto envuelto en una posible solución al problema [9] en este caso la solución quedaría de la siguiente forma: "Busque que cada habitación tenga acceso a aire libre por lo menos por dos lados, luego ubique ventanas en estos espacios para que la luz natural ingrese por más de una dirección e ilumine completamente la habitación".

Luego el patrón incluye un diagrama y una explicación de cómo este patrón se integra al lenguaje de patrones propuesto.

Christopher Alexander siempre consideró de las posibilidades de implementar estos patrones mediante herramientas computacionales que agilizaran el diseño y construcción de edificaciones, pero quizá no se imaginó de la gran utilidad del concepto décadas después en el área del desarrollo de software y más aun dando inicio a un área nueva del conocimiento como la Arquitectura de Software.

Ya en tiempos más cercanos varios autores han abordado el tema bajo diferentes ópticas o perspectivas. Entre los autores más destacados en la bibliografía académica se encuentra Frank Buschmann, quien define varias aproximaciones a los lenguajes de patrones de arquitectura en particular. El autor da la siguiente definición para un Lenguaje de Patrones de Arquitectura de Software: "Así como un patrón de diseño independiente es mucho más que "una solución a un problema que se presenta en un contexto específico, un lenguaje de patrones es mucho más que una red de patrones estrechamente enlazados que definen un proceso para resolver sistemáticamente un conjunto de problemas comunes e interdependientes de desarrollo de software".

El autor también propone varias aproximaciones interesantes al Lenguaje, las cuales en resumen son formas de categorización u agrupamiento de patrones aislados. Entre ellas: La Composición de patrones, las Historias con patrones, las secuencias y por último las colecciones.

Complementos de Patrones: Muestran cómo resolver familias de problemas específicos mediante el uso de patrones, más no soluciones a espacios más amplios; como por ejemplo un sistema completo de información.

Composición de Patrones: Es la configuración específica de un grupo de patrones para solucionar un problema en particular, donde cada patrón se enfoca en un su problema y no impacta significativamente una Arquitectura de Software, ya que no se preocupa por la interacción de todos los patrones entre sí como un todo.

Historias de Patrones: Son como un diario, que le cuenta al lector los patrones aplicados en una solución completa de un sistema, subsistema o componente. Explica qué problemas se resolvieron, en qué orden se aplicaron los patrones de solución y como se relacionan con la Arquitectura de Software resultante. Estas son muy acopladas al contexto (requerimientos y restricciones del sistema en particular); por ejemplo si se tienen dos problemas del mismo dominio pero con restricciones diferentes se obtienen como resultado dos historias completamente diferentes, debido a la aplicación de patrones en diferente orden. Solo se esquematiza una solución, pero pueden existir otras soluciones al problema según varíen las restricciones, y no son tomadas en cuenta por la historia de patrones representada así que no es posible adaptar fácilmente la historia aunque el problema difiera en algunas restricciones del dominio.

Secuencias de Patrones: Remueven de las historias de patrones las restricciones particulares, lo redunda en la posibilidad de generalizar, si dos sistemas comparten restricciones que se mencionan en una secuencia de patrones, ambos pueden diseñarse en base a esta secuencia o al menos las partes más importantes.

El diseño a través de secuencias de patrones es bastante limitado ya que cada una de ellas denota sólo un posible diseño, lo que significa que si varían los requerimientos debería considerarse una secuencia diferente.

Comprimen y abstraen las historias de patrones, permitiendo a desarrolladores usarlas para construir aplicaciones dado el caso en que sus requerimientos se direccionen a los patrones definidos en la secuencia.

Si se construye una solución usando un grupo de secuencias de patrones interconectadas como resultado obtenemos una nueva secuencia para añadir al grupo inicial, estaríamos frente a lo último en cuanto al estado del arte en este tipo de aplicaciones.

Colecciones de Patrones: Categorización personalizada de los patrones por diferentes formas, podría verse como un catálogo de patrones sin ninguna profundidad y sin

posibilidad de impactar o guiar el diseño de arquitecturas, existen diversidad de estos catálogos dependiendo el sector de desarrollo que se quiere atacar, un ejemplo son catálogos para sistemas de alto rendimiento, catálogos para sistemas distribuidos, al revisar estos catálogos se puede observar que simplemente agrupan patrones que se pueden utilizar en la solución de sistemas del contexto de describe el catálogo pero no se pueden aplicar directamente como un conjunto de pasos o patrones a la hora de construir el sistema.

V. Aplicación a Dominios Específicos

Los lenguajes de patrones han ocupado un lugar muy importante en los diseños de Arquitectura de Software. Se aplican tanto en proyectos de investigación y como en proyectos comerciales de desarrollo de software. A continuación se mostrarán ejemplos concretos de su uso y extensión.

Mediante la investigación realizada por la universidad de la Florida en Estados Unidos, los estudiantes Nelly Delessy, Eduardo Fernández y Maria Larrondo-Petrie, todos del departamento de ciencias de la computación, se logró obtener una aplicación concreta de los lenguajes de patrones hacia la solución del problema de administración de identidades en el contexto de la seguridad de sistemas de información [10].

Los problemas de autenticación y autorización que normalmente han sido cuello de botella en muchos sistemas han sido abordados a partir de los autores del artículo mediante el uso de varios patrones de diseño ya conocidos como el Patrón Proveedor de Identidad y el Patrón Circulo de confianza. El trabajo obtuvo como resultado un lenguaje que se integra directamente con el ciclo de desarrollo de cualquier software y permite de forma ágil y correcta administrar la seguridad de servicios, recursos privados e identidad de usuarios en cualquier instante de ejecución de un sistema de información.

Para realizar el lenguaje fue necesario partir este problema en etapas de acuerdo al aspecto que se quería analizar y modelar.

Una primera etapa se basó en el estudio de los patrones de seguridad que se han referenciado a la fecha de realización del artículo. Se realizó un estado del arte y resumen de los principales patrones con el enfoque y dominio al cual hacen referencia.

La segunda etapa se basó en la descripción y modelamiento del lenguaje construido, en éste punto se mostraron figuras que describen la unión y relación de los patrones de seguridad que se integraron para construir el lenguaje, haciendo uso de una figura se describió paso a paso qué relación tenían todos estos patrones y cómo debían ser usados para una implementación y desarrollo de un sistema basado en este lenguaje.

La tercera etapa mostró los patrones de seguridad que se generaron a partir de la creación del lenguaje de patrones, fueron 3 patrones exactamente, el patrón Círculo de verdad, proveedor de identidad y federación de identidad. La forma de describirlo comprendía el nombre general del patrón, el contexto en el cual se aplica, el problema que ataca y la como solucionar dicho problema.

Una segundo ejemplo de cómo los lenguajes de patrones pueden ser usados en dominios específicos se puede observar en el proyecto realizado por IBM donde se define un lenguaje de patrones para aplicaciones e-Business [11].

Este lenguaje divide los patrones de arquitectura en cuatro categorías: Patrones de negocios, patrones de integración, patrones de aplicación y patrones de ejecución, los cuales parten la arquitectura completa en tres capas principales como se sigue: Capa de negocios, capa de integración y Middleware o Capa de infraestructura. A continuación se define cada uno de las categorías que conformaron el lenguaje propuesto.

Patrones de negocio: mapea las interacciones entre los actores del negocio y las funcionalidades del sistema en particular.

Patrones de Integración: Estos se definen como ortogonales a la capa de negocios, creando soluciones particulares que al integrarse soportan las funcionalidades de un sistema en particular.

Patrones de Aplicación: Estos patrones refinan los patrones de negocio e integración subdividiendo los subsistemas en componentes que proveen interfaces claras de implementación de funcionalidades del sistema a diseñar.

Patrones de Ejecución: Refinan los componentes del patrón de aplicación descomponiéndolos en componentes más pequeños sobre la infraestructura o middleware. Ya en este lenguaje los patrones intervienen directamente en la capa física de la aplicación con infraestructura y tecnología.

Esta subdivisión de patrones permite una variedad de soluciones de diseño reusables por diversas aplicaciones en este dominio en particular.

Los Lenguajes de Patrones citados anteriormente han sido utilizados por la industria, y estos se originaron de

proyectos de investigación. Otras universidades e investigadores han desarrollado diversos Lenguajes los cuales hoy en día se han convertido en estándares de facto. A continuación se mencionan algunos estos.

Una serie de lenguajes de patrones desarrollados por los arquitectos Frank Buschamn y Kevlin Henney, son ahora fuentes obligadas de consulta como base para construir otros Lenguajes de Patrones para diversos dominios, entre ellos están: Lenguaje de Patrones para la construcción de sistemas distribuidos, Lenguaje de Patrones para sistemas concurrentes e interconectados, Lenguaje de Patrones para sistemas con administración de recursos, Lenguaje de Patrones para seguridad de sistemas de información, Lenguaje de Patrones para aplicaciones remotas, Lenguaje de Patrones para sistemas con alta tolerancia a fallos.

VI. CONCLUSIONES

La evolución de la ingeniería y Arquitectura de Software han llegado a unos niveles superiores, las herramientas que se mostraron son de vital importancia para el desarrollo de software, hoy por hoy un arquitecto o diseñador de software tiene a su disposiciones grandes cantidades de patrones y lenguajes de patrones todos ellos frutos de la experiencia y el gran avance de esta ciencia, ellos permiten resolver diversos problemas de arquitectura con mejor calidad y en un menor tiempo.

Este artículo muestra el contexto y evolución de los lenguajes de patrones enmarcados en la disciplina de la Arquitectura de Software, para futuros trabajos es importante profundizar en metodologías y procesos que han seguido expertos del área para resolver sus problemas, esto abre las puertas a encontrar métodos eficientes y de referencia al momento de iniciar un diseño de lenguaje de patrones.

REFERENCIAS

[1] FAIRBANKS, George: Just Enough Software Architecture: A Risk Driven Approach. Bolder: Marshall & Brainerd, 2010 15 p

[2] LÁZARO, María y MARCOS, Esperanza: Research in Software Engineering: Paradigms and Methods. Kybele Research Group, Rey Juan Carlos University, 2005

[3] LÁZARO, María y MARCOS, Esperanza, Op. cit., p. 6.

[4] ALEXANDER, Christopher: Notes on the Synthesis of Form, (1964).

[5] DEARDEN, A. M. and FINLAY, J: Pattern languages in HCI: a critical review, Sheffield Hallam University, (2006).

[6] ALEXANDER, Christopher: The Timeless Way of Building (1979) y A Pattern Language (1977).

[7] DEARDEN, A. M. and FINLAY, J: Op. cit., p. 5.

[8] ALEXANDER, Christopher: A Pattern Language (1977).

[9] DEARDEN, A. M. and FINLAY, J: Pattern languages in HCI: a critical review, Sheffield Hallam University, (2006).

[10] DELESSY, Nelly, FERNANDEZ B, Eduardo y LARRONDO PETRIE, Maria. A pattern language for identity manadment. USA: IEEE, (2007), p.1.

[11] ZHAO, Liping, MACAULAY, Linda, ADAMS Jonathan, VERSCHEUREN, Paul. A pattern language for designing e. business architecture. Sience Direct (2007).

[12] Boehm, Barry, A View of 20th and 21st Century Software Engineering, ACM 1-59593-085-X/06/0005 (2006)

[13] Reynoso, Carlos, Una Introducción a la Arquitectura del Software, disponible en: http://carlosreynoso.com.ar/archivos/arquitectura/Arquitectura-software.pdf