Aplicación de metodologías y enfoques para el desarrollo de software de dominio específico en el contexto de Gobierno Digital

Mauro Cambarieri[1], Luis Vivas [1], Nicolás García Martinez[1], Alejandra Viadana[2]

[1] Universidad Nacional de Río Negro (UNRN), Laboratorio de Informática Aplicada LIA, Sede Atlántica, Viedma, Río Negro
{mcambarieri, lvivas, ngarciam}@unrn.edu.ar

[2] Universidad Nacional de Rosario (UNR)
alejandraviadama@gmail.com

Resumen. Este trabajo presenta un conjunto de herramientas, enfoques y estrategias que ayuden a resolver los problemas que se le presentan a los gobiernos para hacer frente a los desafíos de impulsar una transformación digital. Los gobiernos desarrollan iniciativas de Gobierno Digital con el objetivo buscar respuestas en herramientas y/o enfoques de desarrollo. Desde la perspectiva de una institución, en el contexto de Gobierno Digital, es deseable lograr la reutilización de software de manera sistemática con el fin de lograr beneficios asociados a utilizar los artefactos construidos previamente en cada desarrollo nuevo que se realiza. Para que la reutilización del software sea sistemática, los procesos de desarrollo deberían abordar la construcción colectiva de familias de productos relacionados con un dominio [21]. Por otro lado, para estar en consonancia con el modelo y los procesos de negocio, se debe considerar un enfoque de desarrollo que se centre en un dominio de negocio específico. Es por ello, que es evidente en el corto y mediano plazo, la necesidad de un cambio revolucionario en la forma de escribir el software. La modernización puede ser tanto estratégica como técnica y las instituciones deben estar preparadas cuando se presente la oportunidad. La modernización se trata principalmente de una reingeniería para aprovechar los beneficios de las arquitecturas y plataformas modernas. Este trabajo discute los enfoques y metodologías que permitan el desarrollo de software de dominio específico para Gobierno Digital.

Palabras clave: Domain Driven Design, Línea de Productos de Software (LPS), Model-Driven Engineering, Framework. Gobierno Digital.

1 Introducción

El presente trabajo se enmarca en el proyecto de investigación PI 40-C-551 “Herramientas Informáticas para el Desarrollo de Servicios Digitales Innovadores para Comunidades Urbanas y Rurales en el Marco de Ciudades y Regiones Inteligentes” desarrollado en el Laboratorio de Informática Aplicada, Sede Atlántica, Universidad

Nacional del Río Negro (UNRN). La propuesta se encuentra admitido en la "Convocatoria Ideas Proyectos de Desarrollo y Transferencia de Tecnología (IP-DTT)" llevada a adelante por la Secretaría de Investigación, Creación Artística, Desarrollo y Transferencia de Tecnología de la UNRN, para obtener financiamiento. El proyecto tiene una duración de 18 meses y los resultados serán transferidos a la Subsecretaría de Innovación, Desarrollo Sostenible y Ambiente del gobierno de Viedma.

La contribución del mismo es mostrar la factibilidad de las herramientas, enfoques y estrategias propuestas que requieren los gobiernos para hacer frente a los desafíos de uno de los ejes centrales para impulsar una transformación digital, como es la tecnología. El trabajo está estructurado de la siguiente manera: La Sección 2 explica el contexto, situación - problema u oportunidad. La Sección 3 explica los enfoques y metodologías propuesta. A continuación, la Sección 4 presenta los objetivos del proyecto. Finalmente, la Sección 5 presenta los resultados esperados.

2 Contexto - Situación-Problema u Oportunidad

En 1968, en el Congreso sobre Ingeniería del Software que organizó el Comité de Ciencia de la OTAN, se introdujo por primera vez la idea de la reutilización. En ese Congreso el Ing. M.D. McIlroy, de Bell Laboratories, afirmó que "La industria del software se asienta sobre una base débil, y un aspecto importante de esa debilidad es la ausencia de una subindustria de componentes" [1]. En Ingeniería de Software, se entiende por componente, un fragmento reemplazable de un sistema, dentro del cual se encuentran implementadas un conjunto de funcionalidades [2]. Un componente puede desarrollarse aislado.

Uno de los desafíos centrales en el desarrollo de software, justamente, es la reutilización, lo que permite utilizar nuevamente uno o más artefactos implementados como parte del desarrollo de un nuevo producto o sistema. Existen diferentes técnicas que han facilitado, de alguna manera, el uso de artefactos de desarrollo de granularidad cada vez mayor. Al no disponer de contextos suficientemente amplios como para detectar elementos reutilizables y donde pueden utilizarse se desemboca muchas veces en reutilización oportuna. Esto es, se utiliza algún artefacto ya desarrollado con las correspondientes adaptaciones, en el nuevo sistema.

Desde la perspectiva de una institución, en el contexto de Gobierno Digital, es deseable lograr, en cada desarrollo nuevo, un enfoque de reutilización sistemática. El fin, es obtener los beneficios asociados a utilizar artefactos construidos previamente (por ejemplo, la madurez y calidad). En el proceso de desarrollo se debe plantear la construcción colectiva de familias de productos relacionados por un dominio para lograr que la reutilización sea sistemática.

En un contexto de Gobierno Digital, a medida que las aplicaciones crecen en complejidad y tamaño, es posible que puedan dividirse en contextos o subdominios independientes. DDD es un enfoque de desarrollo de software destinado específicamente a abordar los sistemas complejos. Su base estratégica es el proceso de descubrir dominios. En el que entran en juego dos conceptos muy importantes, el

Lenguaje Ubicuo y los Contextos Delimitados (Bounded Context- BC). El Lenguaje Ubicuo es una colección de términos específicos del dominio, el lenguaje de negocios [1]. Los Contextos Delimitados son los límites dentro de los cuales cada modelo de dominio existe y opera. Este es un aspecto crucial y poderoso de la modelización de dominios, ya que permite que los modelos en diferentes contextos evolucionen independientemente unos de otros. Esto tiene un efecto significativo en el mantenimiento porque se hace mucho más sencillo aplicar y probar los cambios en un modelo que se centra únicamente en su propio dominio. El modelo sólo cambiará si cambian las reglas en su propio contexto. Por este motivo, el enfoque permite combatir la complejidad desde el lado de la "modularización". Al dividir grandes procesos en subprocesos más pequeños, es posible diseñar e implementar módulos separados y combinarlos para abordar diferentes aplicaciones [2].

A lo largo del tiempo se ha buscado crear artefactos de software de forma rápida, con menos errores y poniendo más énfasis en los requisitos y necesidades. Con el objetivo de cumplir con esto, MDE brinda más agilidad al desarrollo de aplicaciones. Es una propuesta que se ha venido trabajando desde hace varios años. Es un paradigma para el desarrollo software que considera a los modelos como el principal elemento del proceso de desarrollo [3]. Gracias al uso de modelos, es posible elevar el nivel de abstracción y de automatización. Esto permite atacar los principales problemas en la creación de software: a) la complejidad, b) el desarrollo de marcos de trabajo y lenguajes específicos para lograr la comprensión del dominio; c) mejorar diferentes aspectos de la calidad del software como la productividad y el mantenimiento; y d) obtener provecho de las transformaciones para automatizar trabajo repetitivo y mejorar la calidad del software.

En la actualidad, los objetivos de la industria de software están puesto en la reducción de costos, el aumento de la calidad, el aumento de la productividad y la reducción del tiempo de desarrollo del software. Enfoques como LPS, MDE y DDD buscan alcanzarlos. Adoptarlos dentro de una organización implica desafíos importantes con respecto a los artefactos de software existentes que deben transformarse (MDE) para respetar una arquitectura de software centrada en BC (DDD) y la reutilización (LPS).

3 Conceptos Utilizados

Las siguientes secciones presentan los conceptos utilizados en este trabajo: Ingeniería del Software Basada en Componentes y Reutilización (Sección 3.1), Línea de Producto de Software (Sección 3.2), Ingeniería de Software dirigido por Modelos (Sección 3.3) y Domain Driven Design (Sección 3.4).

3.1. Ingeniería del Software Basada en Componentes y Reutilización. La Ingeniería del Software basada en componentes (CBSE: component-based software engineering) se ocupa del desarrollo de sistemas a partir de componentes reutilizables. El enfoque busca favorecer la reutilización y, de este modo, evitar la duplicación de componentes, impidiendo que el análisis, el diseño, la implementación y la prueba de la misma solución, se desarrolle una y otra vez en muchos de los productos. Difiere del

enfoque tradicional de desarrollo de software que involucra generalmente la construcción de cada una de las piezas desde cero, cada una de ellas implementada a propósito para el producto en curso. Algunas de las piezas o elementos reutilizables provienen de productos anteriores y son reutilizados con ciertos cambios [1]. CBSE, tiende a concentrar el esfuerzo en la composición de componentes de software desarrollados (que por lo general poseen una funcionalidad concreta), trata el desarrollo de sistemas a partir de la composición e integración de componentes reutilizables. Algunos de ellos, pueden requerir una vinculación con otros componentes para proveer la funcionalidad completa. En cuanto a la comunicación, como la integración de los componentes con los elementos del nuevo sistema que se encuentra desarrollando se realiza entendiendo los componentes como cajas negras, esto permite que se utilicen tal y como están disponibles y sin considerar ninguna modificación interna. [1]. A diferencia del enfoque tradicional, CBSE incluye las siguientes actividades: 1) Selección y evaluación de componentes, 2) Adaptación, 3) Composición, y 4) Evolución [1].

Según [12], existen dos factores principales cuando el enfoque que se utiliza está basado en componentes: el primero, la reutilización – la facilidad para reutilizar componentes existentes en la creación de sistemas más complejos – y el segundo, la evolución – crear un sistema altamente formado por componentes pre-existentes facilita su mantenimiento, ya que si el diseño es correcto los cambios estarán muy localizados y los efectos tendrán poca o ninguna propagación en el resto de los componentes del sistema. Estos factores se aplican bajo las siguientes condiciones: en primer lugar, deben existir componentes para su reutilización. En segundo lugar, debe existir un conjunto de componentes bien diseñados y listos para su utilización. Por otro lado, debe existir un modelo de componentes que soporte el enlazado y la interacción de componentes, esto es, debe existir un marco de referencia estándar en el que los componentes puedan existir y comunicarse. Finalmente, es necesario un proceso y la definición y diseño de arquitecturas que den soporte al CBD.

La reutilización favorece el desarrollo de herramientas de software con un alto grado de flexibilidad, haciendo uso de componentes en diversos proyectos, ampliando las capacidades, mejoras, funcionalidades y mantenimiento del software.

Hay varias definiciones del término Reutilización de Software. Algunas de estas definiciones son las siguientes. “La reutilización de software es el proceso de implementar o actualizar sistemas de software usando activos de software existentes” [12]. “Reutilización de software es el proceso de crear sistemas de software a partir de software existente, en lugar de desarrollarlo desde el comienzo” [13].

Existen varias modalidades de reutilización utilizadas, lo mismo que beneficios y costos, según el enfoque que se aplique. En particular, el oportunista, se beneficia de componentes de software construidos en productos anteriores, pero que no fueron especialmente desarrollados para ser reutilizados. En el enfoque más planificado o sistemático, los componentes se construyen pensando en que serán reutilizados, lo que significa que debe plantearse el componente con mayor generalidad, y por ello en el momento de su desarrollo, se requiere mayor inversión de tiempo y recursos.

3.2. Línea de Producto de Software. La definición más comúnmente aceptada de una LPS procede del autor Clements, quien define "las líneas del producto de software como un conjunto de sistemas software, que comparten un conjunto común de características (features), las cuales satisfacen las necesidades específicas de un dominio o segmento particular de mercado, y que se desarrollan a partir de un sistema común de activos base (core assets) de una manera preestablecida" [5].

Las LPS tienen como objetivo cambiar el enfoque de la creación de aplicaciones de software individuales a la construcción de familias de productos mediante la identificación de elementos comunes y variabilidades en un conjunto de aplicaciones de software [14]. En este contexto, una LPS permite acotar el tiempo de producción de componentes software, con altos niveles de calidad y una alta capacidad de reutilización.

En cuanto a los beneficios y costos, según diferentes casos de estudios documentados, existe una tendencia un aumento en la productividad que duplica o triplica la producción de software con respecto a los enfoques tradicionales. Las LPS pueden incrementar la productividad del desarrollo de software, dada la reducción del esfuerzo y el costo necesario para desarrollar, mantener y poner en marcha el conjunto de productos software similares [15].

Una LPS no es únicamente una mera reutilización de código, sino que los activos base se benefician también de cada producto desarrollado en la línea dado que toman ventaja del análisis, diseño, implementación, planificación, prueba. Potencian la reutilización estratégica [11], dado que en este contexto es más planificado que oportunista [16]: en desarrollo de software tradicional, se aprecia la posibilidad de reutilizar un componente después de haberlo desarrollado; en LPS, la reutilización es planificada, de manera que se reutilizan la mayor parte de los activos base en todos los productos de la línea [16]. Una LPS comprende esencialmente dos etapas:

a) Ingeniería de dominio- ID (core asset development, por Clements [5]) es la encargada de diseñar los componentes de software que, de acuerdo a su trazabilidad, hacen parte del dominio común y que son sujetos a reutilización. En esta etapa se definen las similitudes y variantes del modelo de dominio en cada una de las fases. Como producto final en esta etapa se considera un componente altamente reusable y flexible que permite a partir de su definición, la consecución de nuevos modelos y artefactos.

b) Ingeniería de Aplicación - IA (product development, por Clements [15]) se encarga de articular los artefactos definidos en la etapa ID a partir de una plataforma específica. El objetivo de esta etapa consiste en lograr la integración de artefactos bajo uno o varios sistemas, logrando con ello un alto grado de reusabilidad de sus componentes a partir de las especificaciones, la definición y desarrollo, documentación e interoperabilidad entre componentes.

3.3. Ingeniería de Software dirigido por Modelos. La Ingeniería Dirigida por Modelos es una metodología de desarrollo de software que se centra en la creación de modelos, o abstracciones, es decir, identificar conceptos del dominio en lugar de concentrarse en el modelo de cómputo subyacente. A lo largo del tiempo se ha buscado crear artefactos de software de forma rápida, con menos errores y poniendo más énfasis

en los requisitos y necesidades. Con el objetivo de cumplir con esto, la metodología MDE brinda más agilidad al desarrollo de aplicaciones [19].

A partir del uso sistemático de los modelos en las diferentes etapas del ciclo de vida, han surgido un conjunto de paradigmas de desarrollo de software que conforman lo que se ha denominado "Ingeniería de Software dirigida por Modelos", MDE por sus siglas en inglés Model-Driven Engineering o "Desarrollo de Software Dirigido por Modelos" MDSD por sus sigleas en inglés Model-Driven Software Development. El uso de modelos promueve elevar el nivel de abstracción y de automatización y con ello combatir la complejidad en la creación de software, además de mejorar la productividad y mantenimiento del software y la interoperabilidad entre sistemas [18].

En particular, la programación dirigida por modelos es un paradigma para el desarrollo software cuyo principal elemento del proceso son los modelos. A partir de estos modelos el código puede ser generado semiautomáticamente. [17], es decir, como "dibujar un sistema y pasar directamente a ejecutarlo" [1].

3.4. Domain Driven Design (DDD). El diseño impulsado por el dominio es un enfoque de desarrollo de software destinado específicamente a abordar los sistemas complejos, el cual establece lo siguiente: 1-. Centrarse en el dominio. 2.- Explorar modelos de dominio en colaboración con los expertos y desarrolladores. 3- . Hablar un lenguaje común (Lenguaje Ubicuo) dentro de un contexto delimitado. Plantea un enfoque diferente para el diseño y construcción de sistemas y tiene como objetivo crear aplicaciones a partir de una comunicación entre contextos. Es por ello que dichos sistemas son flexibles y adaptables dentro de cada contexto [4].

DDD ofrece un enfoque sistemático y completo para el diseño y desarrollo de software. Proporciona un conjunto de principios, herramientas y técnicas que ayudan a combatir la complejidad, manteniendo el modelo de negocio como pieza central del enfoque. Se adapta bien a una implementación de microservicios, dado que permite la división de un sistema en diferentes contextos delimitados. El diseño esta dado de tal manera que sólo el contexto tiene que ser modificado para implementar cambios o introducir nuevas características. Por tal motivo, se obtendrá el máximo beneficio del desarrollo independiente en diferentes equipos (de desarrollo), ya que varias características pueden implementarse en paralelo sin necesidad de una coordinación. La reutilizando de los microservicios como componentes, permite lograr el aumento de la calidad y la reducción del tiempo de implementación de procesos de negocio de una forma ágil [20]. DDD apunta a centrarse fuertemente en el dominio y el desarrollo de servicios independientes que forman aplicaciones de software más grandes cuando se combinan [7].

4 Objetivos

El objetivo general de este proyecto es diseñar un marco de trabajo (framework) que permita el desarrollo de software de dominio específico para gobierno digital. El mismo consiste en investigar y desarrollar herramientas informáticas, cuyos dominios sean complejos de resolver.

Los objetivos específicos relacionados con el objetivo general que se abordará son:

1. Integrar herramientas existentes que soportan las metodologías y enfoques de desarrollo de software.
2. Analizar el dominio de aplicación, en Plataformas de servicios con intermediación, en particular se estudiará el contexto del gobierno de la Ciudad de Viedma - Río Negro.
3. Especificar requerimientos para identificar contextos delimitados, que representan unidades organizacionales bajo el enfoque DDD y bajo la metodología LPS los activos base y variables por cada uno (en base a lo analizado en 2).
4. Diseñar una arquitectura de referencia que permite la instanciación de la misma para la aplicación de los enfoques propuestos
5. Construir un marco de trabajo(Framewrok) que en base a la arquitectura (objetivo 4) y los requerimientos (objetivo 3) que considere un modelo metodológico y enfoque de construcción acorde para que permita la reutilización sistemática con el fin de lograr beneficios asociados a utilizar los artefactos(componentes) construidos previamente y la implementación de un modelo en constante evolución.
6. Validar el marco de trabajo producto del Proyecto en el caso de estudio, elaborando el prototipo funcional estable del framework.
7. Identificar resultados concretos sobre la usabilidad y utilidad del framework.

5 Resultados esperados

Como resultado de este proyecto, se espera identificar herramientas, metodologías y modelos conceptuales para el desarrollo de software de dominios complejos que permita, la integración de herramientas de construcción y sea considerado un modelo metodológico acorde para la reutilización sistemática del software construido previamente y la implementación de un modelo y/o reglas de negocio en constante evolución. En cuanto al diseño de la arquitectura de software se espera construir regiones o capas, para desacoplar las mismas y que evolucionen de manera aislada, independientes de la tecnología. También, se espera que el proyecto signifique un cambio revolucionario en la forma de escribir el software y una solución a la modernización, entre otras, para aquellos sistemas que se encuentran obsoletos por la tecnología que utilizan. Se obtendrá mediante un caso de estudio, una Plataforma de servicios de empleo, esta será implementada y transferida a la Subsecretaría de Innovación, Desarrollo Sostenible y Ambiente del gobierno de Viedma. La elaboración de este prototipo funcional y estable se espera que produzca, un sistema flexible, con capacidad de reutilización de los componentes que lo conforman, mantenible, independiente de la tecnología y tolerante a los cambios (reglas de negocio).

Referencias

[1] Usaola, M.: Desarrollo de software basado en reutilización. UOC. 2013.

[2] Montiliva, J et al: Desarrollo de software basado en Componentes. V Congreso de Automatización y Control Mérida. Noviembre 2013.

[3] Markus Voelter ; Iris Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. 2007.

[4] E. Evans, Domain-Driven Design:Tackling Complexity in the Heart of Software. Addison-Wesley, 2003

[5] Clements,P., Northrop,L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001).

[6] Oquendo, F.: p-Method: A Model-Driven Formal Method for Architecture-Centric Software Engineering. ACM SIGSOFT Software Engineering Notes. Volume 31 Number 3. (2006).

[7] Domain Driven Design Reference. Definitions and Pattern Summaries. Eric Evans.2015.

[8] Cabot, J. http://es.slideshare.net/jcabot/mdd-desarrollo-de-software-dirigido-por-modelos-que-funciona-de-verdad.

[9] C. W. Krueger:Homeaway's transition to software product line practice: Engineering and business results in 60 days. In Proc. of the 12th International Software Product Line Conference (SPLC'08), Ireland, pages 297–306. IEEE, September 2008.

[10] J. Gonzalez-Huerta, S. Abrahão, E. Insfran: Un enfoque Multi-modelo para la Introducción de Atributos de Calidad en el Desarrollo de Líneas de Producto Software. XVI Jornadas de Ingeniería del Software y Bases de Datos. Septiembre de 2011

[11] Peñalvo, F., et al.: Líneas de Productos, Componentes, Frameworks y Mecanos. Informe Técnico, Departamento de Informática y Automática - Universidad de Salamanca (2002).

[12] Sodhi, J., Sodhi,P: Software reuse: Domain analysis and design process. McGraw-Hill. (1999)

[13] Sametinger, J.: Software engineering with reusable components. Springer Verlag, (1997)..

[14] C.Parra, D. Joya, L. Giral, A.Infante: An SOA approach for Automating Software Product Line. Proceeding SAC '14 Proceedings of the 29th Annual ACM Symposium on Applied Computing Pages 1231-1238. Republic of Korea. March2014

[15] Díaz O., Trujillo, S.: Fábricas de Software: experiencias, tecnologías y organización" (2º edición). Editorial Ra-Ma, (2010).

[16] Trujillo, S., Batory, D. y Díaz, O: Feature Oriented Model Driven Development: A Case Study of Portlets. International Conference on Software Engineering (ICSE) (2007).

[17] H. Escobar, H.Triana, S.Silveira Netto: Conceptualización de arquitectura de gobierno electrónico y plataforma de interoperabilidad para América Latina y el Caribe. Naciones Unidas, Santiago de Chile. julio de 2007.

[18] J. García Molina, García Rubio, V. Pelechano, A. Vallecillo, JM. Vara, C.Vicente-Chicote: Desarrollo de Software Dirigido por Modelos: Conceptos,

Métodos y Herramientas. Edición Ra-Ma 2012.
[19] H. Ed-douibi et al. EMF-REST Generation of RESTful APIs from Models. Proceeding SAC '16 Proceedings of the 31st Annual ACM Symposium on Applied Computing Pages 1446-1453.Italy.2016
[20] Microservices: Flexible Software Architecture. Addison-Wesley Professional.2016
[21] LPS mediante un enfoque generativo, disponible en: http://portal.uned.es/portal/page?_pageid=93,56233773&_dad=portal&_schema=PORTAL&idAsignatura=31105043.