Polo del Conocimiento

Pol. Con. (Edición núm. 34) Vol. 4, No 6
Junio 2019, pp. 173-191
ISSN: 2550 - 682X
DOI: 10.23857/pc.v4i6.1005

Recepción: 20 / 01 / 2019
Aceptación: 19 / 04 / 2019
Publicación: 05 / 06 / 2019

Ciencias técnicas y aplicadas

Artículo de revisión

Catálogo de refactorización de código fuente para la herramienta CASE GeneXus

Catalog of Refactorings for GeneXus CASE Tool

Catálogo de código-fonte de refatoração para a ferramenta CASE GeneXus

Franklin I. Reibán-Lucero [I]
ireiban@etapa.net.ec

Diego M. Cordero-Guzmán [II]
dcordero@ucacue.edu.ec

Correspondencia: ireiban@etapa.net.ec

I. Ingeniero en Sistemas, Universidad Católica de Cuenca, Cuenca, Ecuador.
II. Ingeniero en Sistemas, Doctor en Administración, Decano de la Unidad Académica de Tecnologías de la Información, Universidad Católica de Cuenca, Cuenca, Ecuador.

Resumen

La herramienta de ingeniería de software asistido por computador (CASE) GeneXus se utiliza para desarrollar software en base a la abstracción del conocimiento, permite modelar entidades del mundo real y definir sus características y comportamiento de forma declarativa pero también permite crear objetos basados en lógica procedural cuya construcción depende exclusivamente del desarrollador pues la herramienta no brinda asistencia de ningún tipo para el desarrollo, esto implica que no se verifique exhaustivamente la congruencia con el conocimiento que se captura en las entidades modeladas y que tampoco se optimicen los procesos implementados en los mencionados objetos basados en lógica procedural.

El objetivo de esta investigación fue crear un catálogo de refactorización de código fuente adaptando las mejores prácticas, de este ámbito, al desarrollo de software con la herramienta CASE GeneXus, de forma que se evite el deterioro causado por los ajustes de mantenimiento y la adición de nuevas funciones en una aplicación desarrollada con GeneXus. Esto se cumplió estableciendo un conjunto de reglas para optimizar las aplicaciones desarrolladas con GeneXus como un catálogo de refactorización de código fuente.

Los resultados obtenidos benefician especialmente a la comunidad de desarrolladores GeneXus, una comunidad basada en una cultura «Open Source» y en constante crecimiento. La relación entre la reducción del costo global de mantenimiento y la mejora de la inteligibilidad del código fuente de una aplicación desarrollada con GeneXus, es el beneficio más notorio, y, es también, el aspecto que se analiza mediante la discusión y conclusiones de este documento.

Palabras claves: Refactorización; GeneXus; Desarrollo de software; Optimización.

Abstract

Computer-Aided Software Engineering Tool called GeneXus is used for Software Development based upon knowledge. GeneXus allows to model real world objects and defines their characteristics and behavior declaratively but with GeneXus it is also possible to develop procedural logic objects that rely exclusively upon developer's logic and goals because there's no GeneXus support at all. This means neither strict knowledge congruency verification nor optimization for the procedural logic objects.

The main goal of this research project was the creation of a catalogs of refactorings based upon the refactoring best practices adaptation to GeneXus Software Development in order to avoid software decay as an unwanted effect of maintenance changes and new features addition. This was accomplished through a catalog of refactorings which contains a set of rules to optimize GeneXus developed systems.

The constant growing and open-source driven GeneXus developers community would benefit by this project results. The relationship between maintenance cost reduction and an easier-to-read GeneXus source code is the most important benefit and also is the subject which is analyzed in this paper discussion and conclusions.

Keys words: Refactoring; GeneXus; Software development; Optimization.

Resumo.

A ferramenta de engenharia de software auxiliada por computador (CASE) GeneXus é usada para desenvolver software baseado na abstração de conhecimento, permite modelar entidades reais e definir suas características e comportamento declarativamente, mas também permite criar objetos baseados em lógica procedural cuja construção depende exclusivamente do desenvolvedor porque a ferramenta não fornece assistência de qualquer tipo para desenvolvimento, isso significa que a consistência com o conhecimento capturado nas entidades modeladas não é completamente verificada e que os processos implementados nos objetos acima mencionados não são otimizados com base na lógica processual.

O objetivo desta pesquisa foi criar um catálogo de refatoração de código fonte, adaptando as melhores práticas neste campo ao desenvolvimento de software com a ferramenta CASE GeneXus, a fim de evitar a deterioração causada pelos ajustes de manutenção e a adição de novas funções em uma aplicação desenvolvida com o GeneXus. Isso foi conseguido através do estabelecimento de um conjunto de regras para otimizar os aplicativos desenvolvidos com o GeneXus como um catálogo de refatoração de código fonte.

Os resultados obtidos beneficiam especialmente a comunidade de desenvolvedores GeneXus, uma comunidade baseada em uma cultura de "código aberto" e em constante crescimento. A relação entre reduzir o custo geral de manutenção e melhorar a inteligibilidade do código-fonte de um aplicativo

desenvolvido com GeneXus é o benefício mais notável, e é também o aspecto que é analisado através da discussão e das conclusões deste documento.

Palavras chaves: Refatoração; GeneXus; Desenvolvimento de software; Otimização.

Introducción.

GeneXus es una herramienta inteligente cuyo objetivo es asistir al desarrollador y a los usuarios durante todo el ciclo de vida de las aplicaciones. La idea básica de GeneXus es automatizar todo aquello que sea automatizable: desde la normalización de datos hasta el diseño, generación y mantenimiento tanto de las bases de datos como de las aplicaciones. Esta automatización desvincula al desarrollador de las tareas mencionadas, permitiendo que se enfoque en entender los problemas del usuario y en desarrollar software basado en conocimiento. El desarrollo de software con GeneXus tiene una premisa básica: “no es posible construir un modelo de datos estable” y, en contraposición, una filosofía incremental resulta más natural, al no afrontar este gran problema sino al ir resolviendo pequeños problemas que surgen a medida que se capturan las distintas visiones de los usuarios (GeneXus, 2019). Este aspecto denota la relevancia actual del desarrollo con GeneXus por su similitud con las metodologías ágiles y enfoques para el desarrollo de Software.

La relevancia de GeneXus en el ámbito del desarrollo de software permitió que sea una de las cinco soluciones seleccionadas en el prestigioso informe “Cool Vendors in Spanish Latin America, 2016” publicado por Gartner. La actualidad de GeneXus se evidencia también por eventos como el primer encuentro GeneXus en China o el GxDay en Japón que se realizaron en el año 2018.

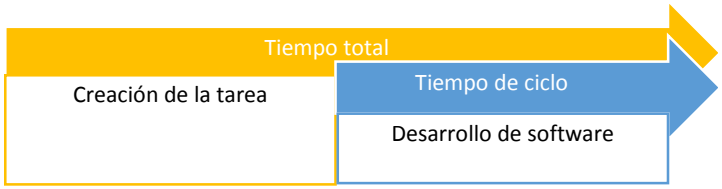
Una aplicación construida en GeneXus se basa, casi por completo, en la abstracción de conocimiento. La representación de entidades del mundo real y sus características se describen de forma declarativa; pero también, es posible crear objetos basados en lógica procedural que depende exclusivamente del desarrollador, pues GeneXus no verifica su congruencia con el conocimiento: riguroso, operable y representable en forma objetiva; que capturan la mayoría de los elementos de una aplicación.

A medida que se incrementa el tamaño de una aplicación desarrollada en GeneXus, es posible que las deficiencias de la lógica procedural deterioren la aplicación e incrementen sus costos de mantenimiento, con la inducción de defectos como: funciones duplicadas, consultas ineficientes de datos o código fuente inteligible.

Este trabajo plantea un conjunto de reglas para optimizar las aplicaciones implementadas con GeneXus y reducir los costos de mantenimiento mediante un catálogo de refactorización de código fuente.

La reducción de costos de mantenimiento del software desarrollado con GeneXus se presenta utilizando la métrica «tiempo de ciclo» o Cycle Time, la misma que se puede definir como (Nicolette, 2015): el tiempo total que transcurre desde que inicia una tarea de desarrollo de software hasta que la mismo es completada.

Figura 1 Tiempo de ciclo del desarrollo de software.



La forma de utilizar está métrica en el desarrollo con GeneXus consiste en tomar las fechas inicial y final de modificación de cada objeto y calcular el tiempo de ciclo en días, como se muestra en la tabla 1.

Tabla 1 Tiempo de ciclo para tareas de mantenimiento de software con GeneXus.

OBJETO	FECHA INICIAL	FECHA FINAL	TIEMPO DE CICLO
Objeto 1	07/05/2019	22/05/2019	16
Objeto 2	06/05/2019	22/05/2019	17
Objeto 3	14/05/2019	16/05/2019	3
Objeto 4	08/05/2019	16/05/2019	9
Objeto 5	06/05/2019	16/05/2019	11
Objeto 6	06/05/2019	16/05/2019	11
Objeto 7	14/05/2019	15/05/2019	2

Este documento presenta un catálogo de refactorización de código fuente adaptando las mejores prácticas existentes al desarrollo de Software con la herramienta CASE GeneXus. La sección 2 presenta un marco de referencia de refactorización para GeneXus como la metodología para establecer las condiciones bajo las que el catálogo de refactorización pueda ser utilizado adecuadamente en un

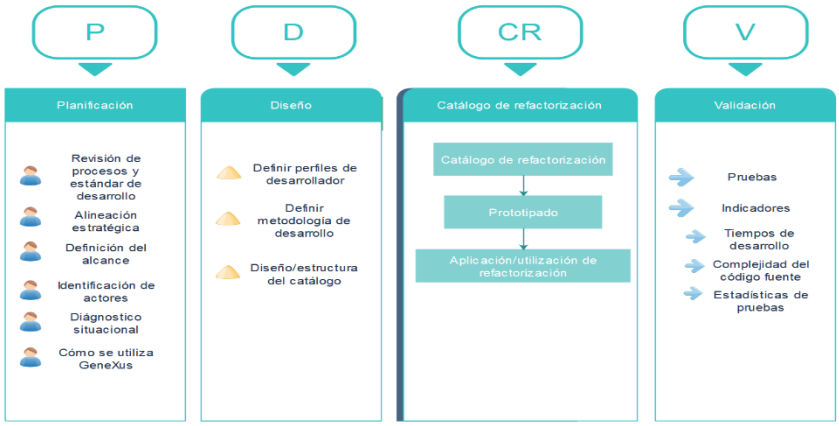
ámbito específico de forma que cumpla el objetivo de reducir los costos de mantenimiento de una aplicación desarrollado con GeneXus. La sección 3 presenta 3 casos que forman parte del catálogo de refactorización, para cada caso se presentan los pasos a seguir al utilizarlo y un ejemplo que contrasta la versión optimizada de un objeto GeneXus, que se consigue al ejecutar los pasos descritos en el catálogo de refactorización para GeneXus, contra las deficiencias de la lógica procedural del objeto en su versión previa a la refactorización. La sección 4 presenta las conclusiones de este trabajo, entre las que se destaca la siguiente: el tiempo de desarrollo de tareas de mantenimiento de software con GeneXus se redujo notablemente e incluso se estabilizó en la muestra analizada de acuerdo a los valores recogidos mediante la métrica tiempo de ciclo.

Metodología.

El desarrollo de este trabajo se centra en la creación del catálogo de refactorización para GeneXus, sin embargo, es necesario iniciar con la definición de un Framework o «marco de referencia» que permita adecuar y utilizar cada una de las tareas de refactorización al ámbito específico en el que se deba aplicar.

El marco de referencia está conformado por los 4 componentes mostrados en la figura 2.

Figura 2 Marco de referencia para refactorización en GeneXus.



El desarrollo metodológico tiene la siguiente estructura:

Planificación.

- Revisión de procesos y estándar de desarrollo: Es necesario establecer una línea base sobre normas y reglas que se deben cumplir en el desarrollo de software con GeneXus, la utilización del catálogo de refactorización debe seguir los procesos y estándares existentes.
- Alineación estratégica: Es primordial establecer la relación con el Plan Estratégico de TI (PETI), incluso con los objetivos estratégicos de la organización; uno de los principales obstáculos para llevar a cabo tareas de refactorización es que la alta gerencia no logra percibir el verdadero valor de la optimización interna de las aplicaciones informáticas, generalmente se asume que «arreglar lo que no está roto» es un gasto innecesario.
- Definición del alcance: Los objetivos del negocio o el ámbito de trabajo del área de tecnologías de la información siempre irán más allá del desarrollo de software, incluso habrá que considerar que GeneXus podría no ser la única herramienta de desarrollo que se utiliza; por lo que será necesario definir los resultados que se puede esperar de la utilización del catálogo de refactorización.
- Identificación de actores: Similar a la gestión de cualquier tipo de proyecto, es necesario identificar a las partes interesadas o actores involucrados, fundamentalmente se tratará del equipo o área de desarrollo de software, pero se debe considerar que muchas metodologías definen roles híbridos que conectan la tecnología con el giro de negocio, como ejemplo se puede mencionar el rol de Scrum Master.
- Diagnóstico situacional/utilización de GeneXus: Es el punto de partida para la aplicación del catálogo de refactorización propuesto, debido a la especialización que involucra la oferta presentada en este trabajo, habrá un enfoque específico sobre la forma en la que se utiliza la herramienta GeneXus.

Diseño.

- Definir perfiles de desarrollador: De acuerdo al nivel de experiencia de cada desarrollador se le asignará un perfil que permite saber si está en capacidad de entender y utilizar correctamente el catálogo de refactorización para GeneXus o si quizá, en el caso de un desarrollador principiante, serán necesarias actividades de capacitación y monitoreo del uso correcto de la herramienta propuesta.
- Definir metodología de desarrollo: Establecer una forma de trabajar regida por los procesos y el estándar de desarrollo existentes en conjunto con un catálogo de refactorización adecuado

a la organización; incrementará la probabilidad de disminuir el costo de mantenimiento de aplicaciones desarrolladas con GeneXus.

- Diseño/estructura del catálogo: Considerando todos los aspectos analizados, es recomendable definir un catálogo único para todos los actores, al tratarse de un marco de referencia es probable que cada persona interprete y utilice el catálogo de refactorización de distintas maneras.

Catálogo de refactorización.

- Catálogo de refactorización: Obtener un documento formal y conocido por todos los actores involucrados, será el punto de partida para cambiar la forma de desarrollar software con GeneXus.
- Prototipado: Basando el trabajo propuesto en las metodologías de desarrollo ágil, cada tarea de refactorización debería ser validada mediante un prototipo, identificando un módulo u objeto dentro una aplicación en el que se pueda usar y verificar los pasos propuestos y resultados esperados.
- Aplicación/Utilización: Los casos verificados de refactorización para GeneXus podrán ser utilizados consistentemente en todos los desarrollos de software, ya sea para implementar nuevas funciones o para optimizar funciones existentes.

Validación.

- Pruebas: Comprobar que cada tarea de refactorización realizada cumpla con la premisa inicial de no afectar el comportamiento de una aplicación es indispensable. Tentativamente podría bastar con las pruebas unitarias que se hacen como parte del desarrollo, pero siempre será mejor que se cumplan pruebas de regresión e incluso verificaciones de parte de «aseguramiento de calidad», pues se debe considerar que el desarrollo de nuevas funciones y la refactorización se pueden llevar a cabo de forma simultánea.
- Indicadores: Definir aspectos que se puedan medir y presentar en informes, es también necesario para justificar la validez y la necesidad de basar el desarrollo de software en el marco de referencia propuesto.

Resultados.

El catálogo de refactorización creado está compuesto por los distintos casos de refactorización que resultan útiles y adecuados para el desarrollo de software bajo el paradigma específico de GeneXus. Todos los casos de refactorización considerados se exponen mediante 3 secciones:

- Una descripción del estado inicial del código fuente y el objetivo de la refactorización planteada.
- Los pasos que se deben seguir para aplicar la refactorización de código fuente, y,
- Un ejemplo mediante imágenes que muestran el código fuente de una aplicación empresarial desarrollada con GeneXus.

A continuación, se presenta una versión reducida del catálogo de refactorización desarrollado en este proyecto, se incluyen únicamente 3 de los casos de refactorización creados.

Enlazar procedimientos.

Estado inicial y objetivo de la refactorización.

Varios objetos GeneXus realizan operaciones similares, pero cada uno tiene una implementación particular, con lo que un mismo fragmento de código fuente se duplica en varios lugares. El costo del mantenimiento se incrementa a la par con el número de objetos que realizan una misma operación, pues si se requiere un cambio en la lógica, este debe ser realizado en todos los objetos. Así también, existe un alto riesgo de generar diferencias e introducir defectos debido a cambios en un único objeto o al no modificar ciertos objetos en una actualización.

Pasos.

Identificar 2 o más objetos que contienen código similar, determinar qué objeto es el más completo, es decir, en qué objeto se llevan a cabo todas las operaciones necesarias para que todos los casos funcionen correctamente.

El objeto más completo, de ser un procedimiento existente o un nuevo procedimiento al que se mueve el código similar, será el objeto base (que se denomina X) al que se enlazarán los demás. El

procedimiento X podrá ser llamado por el proceso Y, que a su vez es llamado por el proceso Z, o podrá ser llamado por los procedimientos A y B, etc.

Los valores que se utilizan como condiciones en la operación son enviados como parámetros de entrada y los valores retornados que no sean útiles en uno u otro caso, simplemente estarán almacenados en variables no utilizadas.

Compilar y probar los cambios

Ejemplo

Una aplicación implementada en GeneXus tiene una tabla de configuración en la que se almacenan parámetros de dos tipos: texto y numérico. Un único procedimiento que permita recuperar los 2 tipos de valores será el procedimiento X y en base al mismo, se crean 2 procedimientos específicos para consultar el tipo de parámetro; esto evita duplicar código fuente en 2 objetos GeneXus, como se observa en las figuras 3 a 5.

Figura 3 Enlazar procedimientos: procedimiento GeneXus base.

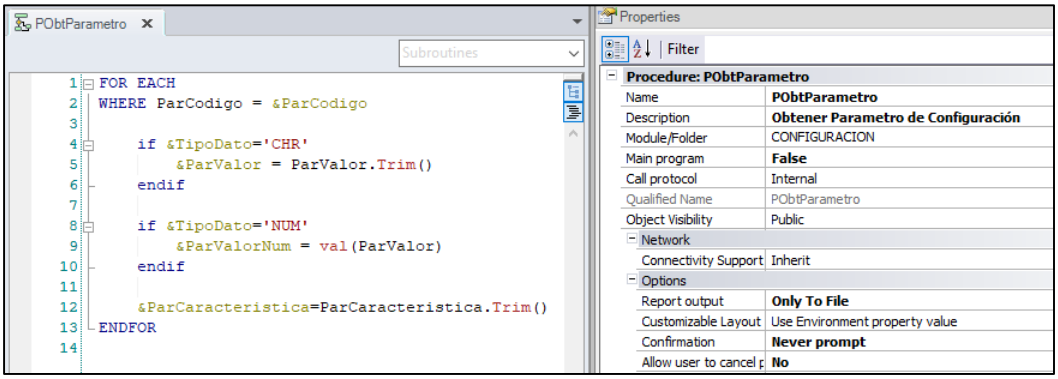


Figura 4 Enlazar procedimientos: referencia a procedimiento base desde procedimiento que retorna valor CHAR.

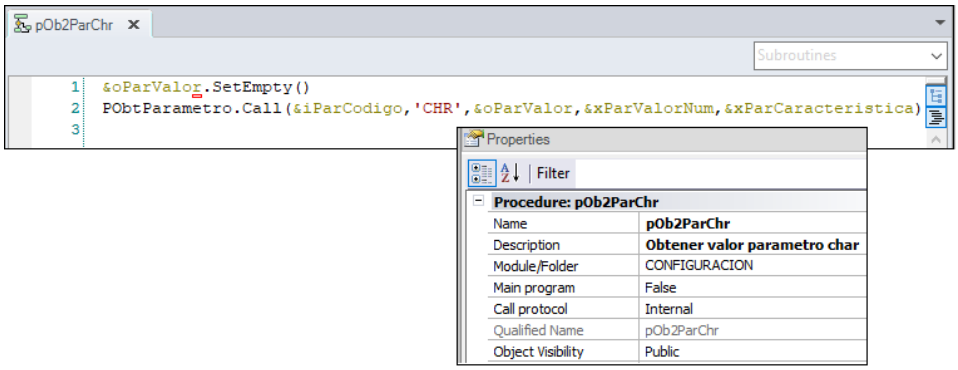
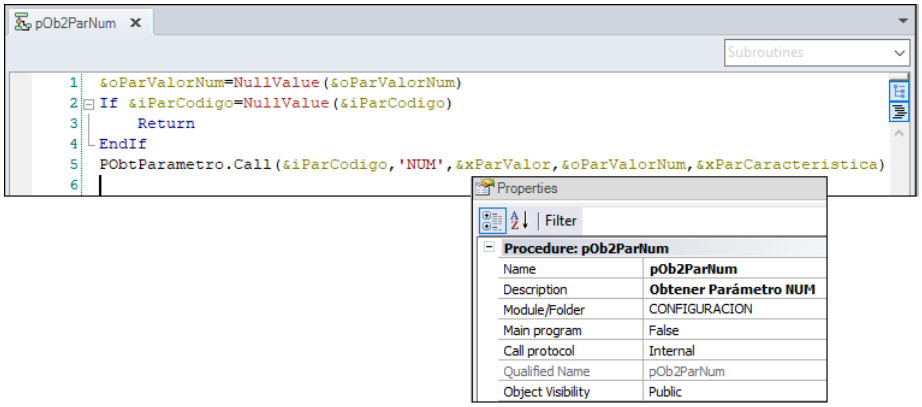


Figura 5 Enlazar procedimientos: referencia a procedimiento base desde procedimiento que retorna valor número.



Extraer subrutina.

Estado inicial y objetivo de la refactorización.

Un objeto GeneXus puede contener un extenso código fuente, que resulta difícil de entender si en un único bloque se realizan varias tareas sin diferenciar claramente el objetivo de cada una. Esta complejidad innecesaria incrementaría el tiempo y esfuerzo en el mantenimiento; por lo que se pretende mejorar la legibilidad del código fuente creando subrutinas con nombres que muestren claramente su objetivo.

Pasos.

- Identificar un fragmento de código fuente que realiza una tarea específica (por ejemplo: un bloque For each que obtiene los datos de cliente)
- Extraer el código y crear una subrutina con un nombre que explique claramente su propósito.
- Los objetos GeneXus solo tienen variables globales por lo que es necesario:
- Inicializar las variables que son útiles únicamente dentro de la subrutina con valores nulos (por ejemplo: cero o un string vacío)

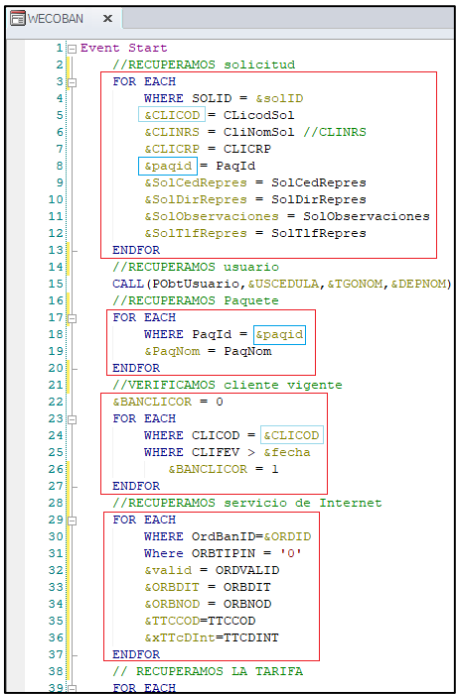
- Inicializar las variables que son parámetros de entrada con el valor adecuado, antes de la llamada a la subrutina creada.

Si un bloque de código se divide en varias subrutinas, es necesario considerar el orden de ejecución de las mismas en base a si algunos valores retornados por la subrutina X son valores de entrada para la subrutina Y

Ejemplo

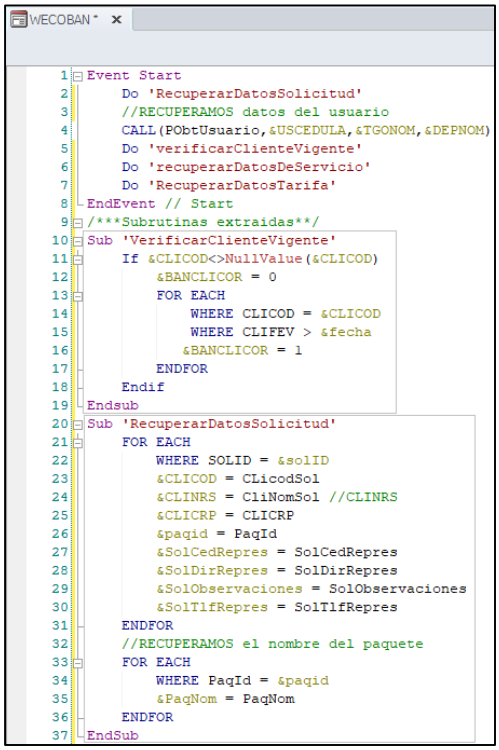
En las figuras 6 y 7 se muestra la carga inicial de un formulario Windows mediante la que se recuperan todos los datos de una solicitud de Internet residencial en el evento Start, donde se ejecutan 4 consultas de datos (una tras otra) y únicamente las líneas de documentación proporcionar información sobre el objetivo de cada una. Así también, la dependencia entre cada bloque no es evidente. Al extraer subrutinas, se simplifica la inteligibilidad del evento Start y la dependencia entre subrutinas resulta evidente.

Figura 6 Extraer subrutina: estado inicial.



Pol. Con. (Edición núm. 34) Vol. 4, No 6, junio 2019, pp. 173-191, ISSN: 2550 - 682X

Figura 7 Extraer subrutina: código fuente refactorizado.



Remplazar procedimiento con Data Provider.

Estado inicial y objetivo de la refactorización.

Un procedimiento GeneXus que debe devolver como único resultado un objeto o una colección de objetos de un tipo de datos estructurados o compuestos (representados en GeneXus como un objeto SDT que es similar a una clase en el paradigma orientado a objetos) suele tener código fuente excesivo e ineficiente, enfocado en la obtención de datos en lugar del resultado que debe devolver el procedimiento. Es recomendable utilizar un objeto Data Provider con el que este tipo de funcionalidad se puede implementar de forma declarativa, enfocado en el resultado a obtener.

Pasos.

Identificar el tipo de objeto SDT generado por el procedimiento a remplazar y utilizarlo como resultado (output) en la creación del Data Provider.

La consulta inicial de datos (comando For Each) se convierte en el grupo inicial del Data Provider, se copian las mismas condiciones (sentencia where), este grupo se asocia al objeto SDT a generar.

Las consultas de información adicional (comandos For Each) se convierten en grupos que no generan salida, esto se indica con la cláusula [NoOutPut]

Las condiciones para los grupos adicionales se simplifican, se descartan las sentencias where que incluyen variables auxiliares, GeneXus inferirá automáticamente la relación entre datos.

Remplazar el procedimiento por el Data Provider en los objetos GeneXus que lo utilizan.

Ejemplo

El estado inicial de un objeto que recupera datos mediante lógica procedural se muestra en la figura 8, la figura 9 muestra una programación declarativa similar en un objeto de tipo Data Provider y la utilización del Data Provider en lugar del procedimiento original se muestra en la figura 10.

Figura 8 Reemplazar procedimientos con Data Provider: estado inicial.

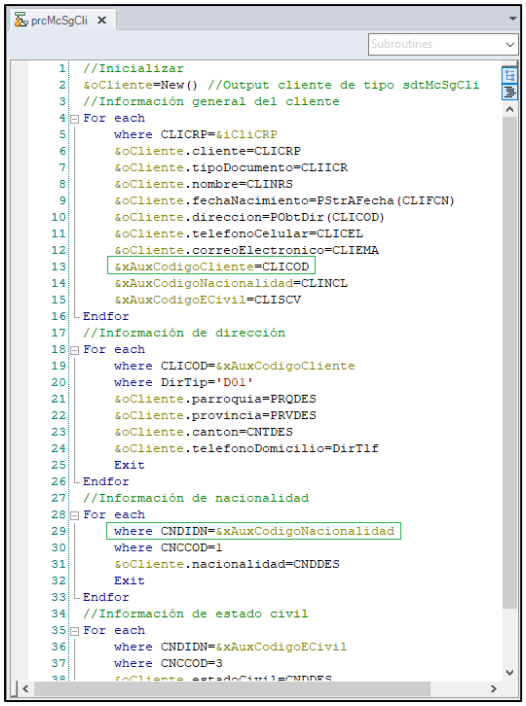


Figura 9 Reemplazar procedimientos con Data Provider: Data Provider.

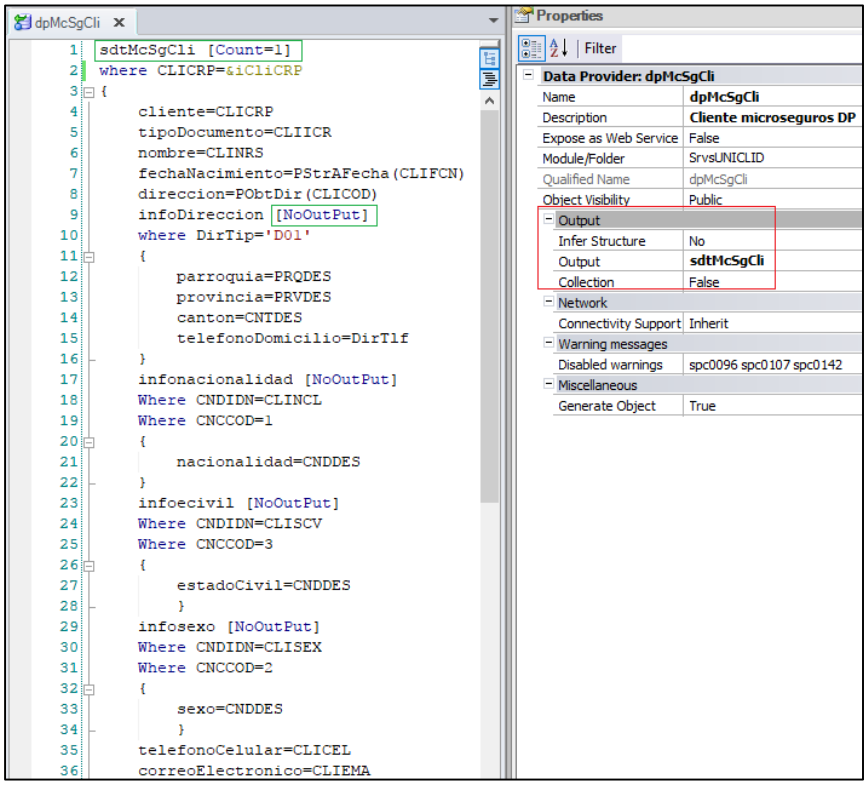
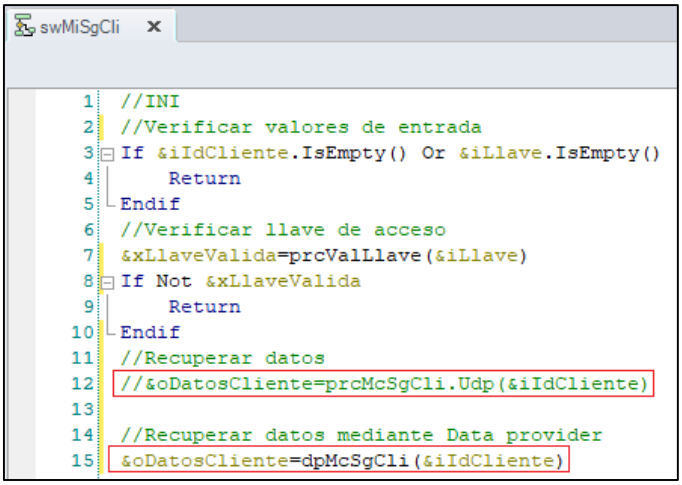


Figura 10 Remplazar procedimientos con Data Provider: remplazo.

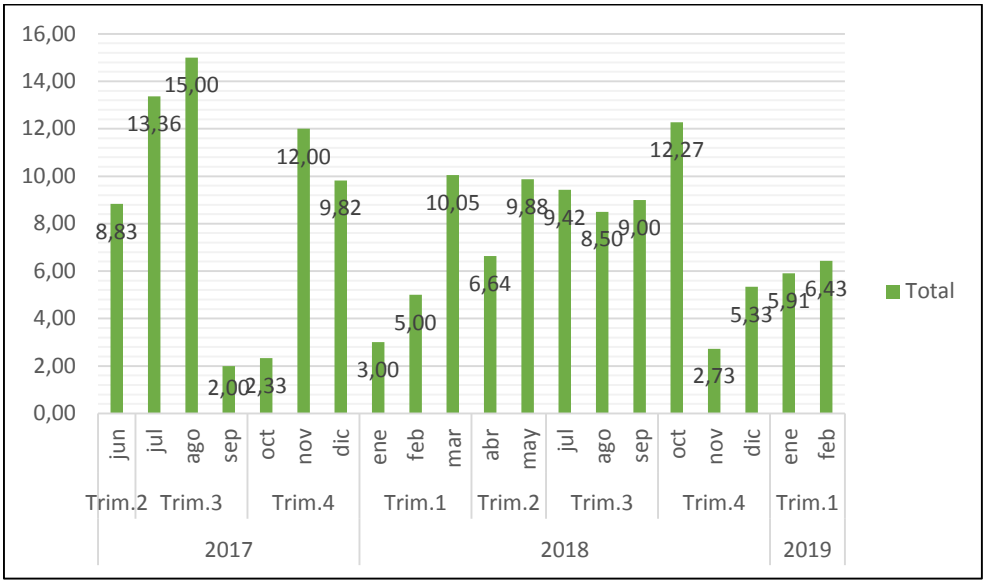


Conclusiones.

El objetivo fundamental de la refactorización es mejorar la estructura interna del software, los casos de refactorización presentados evidencian que efectivamente se logra «limpiar» el código fuente GeneXus, haciéndolo más fácil de entender y modificar.

Se planteó que el costo de mantenimiento de software debería disminuir si la refactorización realmente funciona. Si el código fuente es fácil de entender y modificar, el tiempo requerido para una tarea específica de mantenimiento debería ser menor; por lo que se procura verificar el efecto de la refactorización en base valores históricos de la métrica «tiempo de ciclo».

Figura 11 Valores históricos de promedio de tiempo de ciclo en GeneXus.

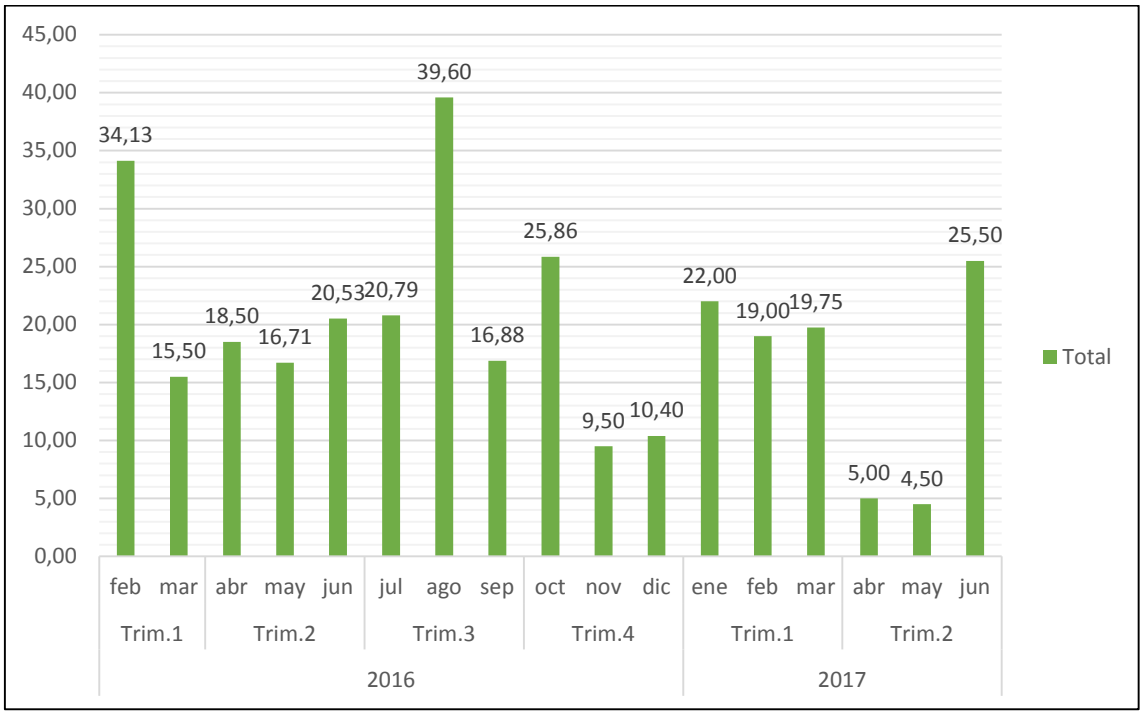


El tiempo de ciclo de distintas tareas de mantenimiento de software con GeneXus, bajo un enfoque de refactorización, tiende a disminuir de acuerdo a la muestra presentada en la figura 11. Cabe además resaltar el período de diciembre de 2018 a febrero de 2019 en el que es evidente la uniformidad del promedio del tiempo de ciclo.

El efecto positivo de la refactorización se ve también en el mes de octubre de 2018, en el que un desarrollador externo entrega un módulo e incluso sin regirse explícitamente al enfoque de refactorización, encuentra un código fuente más fácil de entender y modificar, optimizando su tiempo de ciclo.

En el análisis similar en un proyecto desarrollado con GeneXus entre febrero de 2016 y junio de 2017, en el que no se realizó ninguna tarea de refactorización, es difícil identificar un comportamiento o patrón en los tiempos promedio de ciclo obtenidos, como se observa en la figura 12.

Figura 12 Promedios de tiempo de ciclo sin refactorización.



Es importante considerar el efecto de la refactorización sobre la calidad del software. La calidad del software es el grado en que el software posee una combinación deseable de atributos como adaptabilidad o reusabilidad (International Organization for Standardization, 2001), el catálogo de refactorización para la herramienta CASE GeneXus pretende contribuir con la calidad del software mejorando el diseño, la inteligibilidad del código fuente y reduciendo el número de defectos.

Bibliografía.

Alshayeb, M. (2011). The Impact of Refactoring to Patterns on Software Quality Attributes. Arabian Journal for Science and Engineering, 1241–1251.

Artech Consutores S.R.L. (1 de Enero de 2012). GeneXus documentación. Obtenido de https://www.genexus.com/genexus/documentacion?es

Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., & Houston, K. (2007). Object-oriented analysis and design with applications. Addison-Wesley Professional.

Buschmann, F., Stal, M., & Rohnert, H. (1996). Pattern-Oriented Software Architecture. Wiley.

Cristain, V. &. (2002).

Cunningham, W. (1992). The WyCash portfolio management system. In: Addendum to the proceedings on object -oriented programming systems, languages, and applications. OOPSLA.

Cunningham, W., & Beck, K. (1987). Using Pattern Languages for Object-Oriented Programs. Orlando.

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring: improving the design of existing code. Addison-Wesley Professional.

Gallardo, D. (9 de Septiembre de 2003). Refactoring for everyone How and why to use Eclipse's automated refactoring features. Obtenido de https://www.ibm.com/developerworks/library/os-ecref/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns Elements of Reusable Object-Oriented Software. Indianapolis: Addison-Wesley.

GeneXus. (17 de Junio de 2019). Sobre GeneXus. Obtenido de GeneXus: https://www.genexus.com/files/wp-vision-general?es.

Gonda, B., & Jodal, N. (Mayo de 2007). Desarrollo basado en conocimiento filosofía y fundamentos teóricos de GeneXus.

Gonda, B., & Jodal, N. (01 de Enero de 2009). Evolución de los objetivos de Genexus hacía la cuarta dimensión. Obtenido de Evolución de los objetivos de Genexus hacía la cuarta dimensión: https://www.genexus.com/files/Evolucion-de-los-objetivos-de-GeneXus-hacia-la-Cuarta-Dimension.pdf?es

Highsmith, J. (2009). Zen and the art of software quality. Agile 2009 Conference.

International Organization for Standardization. (2001). ISO/IEC 9126-1:2001.

Kerievsky, J. (2004). Refactoring To Patterns. Addison-Wesley.

Kniesel, G., & Koch, H. (2004). Static composition of refactorings. Science of Computer Programming, 9-51.

Lippert, M., & Stephen, R. (2006). Refactoring in large software projects. West Sussex, Inglaterra: John WIley & Sons, Ltd.

Riehle, D., & Züllighoven, H. (1996). Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems, 3-13.

Ritchie, P. (2010). Refactoring with Microsoft Visual studio 2010. PACKT Publishing.

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2015). Refactoring for Software Design Smells Managing Technical Debt. En G. Suryanarayana, G. Samarthyam, & T. Sharma, Refactoring for Software Design Smells Managing Technical Debt. (págs. 14-16). Waltham, MA, 02451, USA: Elsevier Inc.