Director
Teoría de la Computación

Ulises C. Ramirez [ulir19@gmail.com, ulisescolrez@gmail.com]
Héctor J. Chripczuk [hectorejch@gmail.com]

01 de Noviembre, 2018

Versionado

Para el corriente documento se está llevando un versionado a fin de mantener un respaldo del trabajo y además proveer a la cátedra o a cualquier interesado la posibilidad de leer el material en la última versión disponible.

Repositorio: *https://github.com/ulisescolina/UC-TC*

–Equipo Director

Índice general

Capítulo 1

Estado de la Cuestión

1.1. Problemática

En el campo del desarrollo de software el advenimiento de notaciones como UML generaron un nuevo paradigma para la implementación de software, si bien esta notación no es una idea nueva, por el hecho de que tiene sus inicios en la decada de los 1990 siendo la idea principal del mismo, el modelado de sistemas de software en varios niveles de abstracción [1], además considerandose por estudios como la herramienta de notación para modelado de software dominante [2] aunque usado predominantemente de manera informal y que las actividades realcionadas al ámbito tienden a ser más fáciles en un desarrollo con un enfoque en el modelado [3] con la capacidad de además de tener beneficios dentro del entorno de desarrollo dado al nivel de expresión que se tiene en las notaciones gráficas comparadas con las formas textuales de expresión [4].

A pesar de lo mencionado, encuestas realizadas en uno de los estudios que trata a UML como una notación dominante dentro del desarrollo de software realizada años atras, muestran que desarrolladores prefieren centrar el trabajo en vastas cantidad de líneas de código, raramente mirando diagramas, y mucho menos confeccionándolos o editándolos, posibles razones para esto se pueden para este comportamiento se descubre en [5] en donde se identifican cuatro razones por las cuales los efectos positivos de UML pueden verse reducidos, aunque para el presente trabajo se puede hacer hincapié en tres de los mencionados: *la no-factibilidad de realizar ingenieria inversa en codigo heredado*, *costo de entrenamiento del equipo* y *los requerimientos no estaban distribuidos en unidades funcionales del sistema*.

En el mismo estudio, se consultó cuáles son las cuestiones de trabajar con el desarrollo orientado a modelos que lo hacen difícil de tratar o sean la razon por la cual no se modele mucho. Las razones principales fueron:

1. *Los modelos se desactualizan con respecto al código*, es decir no se encuentra un flujo de trabajo que contemple a ambas actividades tal que estas dos avancen a la par, sino que se deben realizar por separado, causando

así que el modelo quede desactualizado con frecuencia.

2. *Los modelos no son intercambiables fácilmente*, aquí se habla de la dificultad de, nuevamente, encontrar un flujo de trabajo que permita el paso de los modelos de forma más "natural".

3. *Las herramientas para modelado son pesadas*, haciendo alusión a los recursos computacionales en los cuales se debe incurrir para utilizar dichas herramientas.

4. *El código generado desde una herramienta no es de mi agrado.*

5. *Crear y modificar un modelo es lento.*

Además de la encuesta anterior, también se realizó una centrándose en el desarrollo *centrado en la codificación*, los siguientes fueron los mayores inconvenientes identificados por los desarrolladores.

1. Ver de "un pantallazo" todo el diseño es un problema, una forma de transmitir la dificultad que se tiene de brindar una visión a un alto nivel del proyecto, lo cual lleva al tercer punto de la lista.

2. Comportamiento del sistema difícil de entender.

3. La calidad del código se degrada con el tiempo.

4. Muy dificil reestructurar el proyecto.

5. Cabios de codigo pueden implementar bugs.

1.2. Lenguaje Específico de Dominio

Un lenguaje especifico de dominio (DSL) es un lenguaje de software el cual se especializa en abarcar un problema dentro de la ingeniería en particular, las cuales son características para un dominio de aplicación, éste, se basa en abstracciones alineadas a este dominio y provee una sintaxis propicia para aplicar estas abstracciones de forma efectiva [6].

Capítulo 2

Problema

Descripcion del problema que se pretende mitigar con el lenguaje

Capítulo 3

Director

Estructura del Lenguaje

3.1. Palabras Reservadas

3.2. Símbolos Especiales

3.3. Expresiones Regulares

3.4. Autómatas Finitos

3.5. Analizador Sintáctico, Gramática Libre de Contexto

3.6. Arboles de Análisis Sintáctico

3.7. Diagrama de Sintáxis

Referencias

[1] C. Michel R. V., "Empirical Studies into UML in Practice: Pitfalls and Prospects", *IEEE/ACM 9th International Workshop on Modelling in Software Engineering*, 2017. DOI: 10.1109/MiSE.2017.24.

[2] A. Aldaeej y O. Badreddin, "Towards Promoting Design and UML Modelling Practices in the Open Source Community", *IEEE/ACM 38th International Conference on Software Engineering Companion*, 2016. DOI: 10.1145/2889160.2892649.

[3] A. Forward y T. Lethbridge, "Perceptions of Software Modeling: a Survey of Software Practitioners", *5th Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of MDD*, 2010.

[4] B. Rumpe, *Modelling With UML, Language, Concepts, Methods*. Aachen, Alemania: Springer, 2004. DOI: 10.1007/978-3-319-33933-7.

[5] B. Anda, K. Hansen, I. Gullesen y H. K. Thorsen, "Experience from Introducing UML-based Development in a Large Safety-Critical Project", *Empirical Software Engineering*, 2006. DOI: 10.1007/s10664-006-9020-6.

[6] S. Sobernig, B. Hoisl y M. Strembeck, "Extracting Reusable Design Decisions for UML-based Domain-specific Langages: A Multi-Method Study", *Journal of Systems and Software*, 2016. DOI: 10.1016/j.jss.2015.11.037.