DOI: https://doi.org/10.5281/zenodo.16944373

Uso de elementos lingüísticos en la educación de la programación: un enfoque naturalístico

Use of linguistic elements in programming education: a naturalistic approach

Lizbeth Alejandra Hernández González[1] y Ulises Juárez Martínez[2]

Resumen: Este trabajo presenta la propuesta de incorporar lenguajes naturalísticos de propósito general en la enseñanza de la programación. Estos lenguajes utilizan elementos del lenguaje natural como la reflexividad, que es la capacidad de hacer referencia a elementos que han sido mencionados previa o posteriormente. La intención es que el programador forme oraciones más expresivas, es decir, más claras, a partir de una gramática acotada que la computadora pueda procesar. El trabajo presenta los lenguajes representativos del paradigma y sus características principales. Se incluyen los resultados de un estudio de caso de un estudiante de maestría que utilizó el lenguaje Cal-4700. Finalmente, se presenta una prueba con enfoque cualitativo, en la que participaron cinco estudiantes universitarios que programaron una tarea mediante el mismo lenguaje. Los resultados sugieren que este enfoque favorece la claridad del código y promueve la motivación en el aprendizaje de la programación.

Palabras clave: paradigma naturalístico, programación naturalística, lenguajes naturalísticos

Abstract: This paper presents a proposal to incorporate general-purpose naturalistic languages into programming education. These languages use natural language elements, such as reflexivity, which is the ability to refer to elements mentioned previously or subsequently. The intention is for programmers to form more expressive sentences, which means clearer sentences based on a limited grammar that the computer can process. The paper presents the languages representative of the paradigm and their main characteristics. The results of a case study of a master's student who used Cal-4700 are included. Finally, a qualitative test is presented, in which five university students participated by programming a task using the same language. The results suggest that this approach favors code clarity and promotes motivation in learning programming.

Keywords: naturalistic paradigm, naturalistic programming, naturalistic languages.

1 Universidad Veracruzana, Xalapa, Veracruz, México, lizhernandez@uv.mx; https://orcid.org/0000-0002-8524-3209

2 Tecnológico Nacional de México-Instituto Tecnológico de Orizaba, Veracruz, México, ulises.jm@orizaba.tecnm.mx; https://orcid.org/0000-0002-5911-3136

Introducción

El proceso de desarrollo de *software* es inherentemente complejo. En primer lugar, se considera el aspecto mental, en el que el ingeniero de *software* modela la solución abstrayendo los elementos principales del problema, para, a partir de ello, escribir código que satisfaga la solución propuesta. Durante este proceso es habitual que las ideas originales se deformen a causa de la necesidad de adaptarse a la forma en la que funciona actualmente la tecnología, de manera que, muchas veces, se desvía el resultado de los objetivos iniciales.

Como alternativa de solución se propone hacer uso de la programación naturalística. Esta es una nueva forma de crear lenguajes de programación, utilizando elementos del lenguaje natural como la reflexividad, es decir, la capacidad de hacer referencia a elementos mencionados anterior o posteriormente. El paradigma naturalístico tiene la capacidad de analizar, modelar e implementar *software*, reduciendo las dificultades propias del lenguaje de programación y preservando las ideas originales con mayor fidelidad, ya que se utilizan elementos del lenguaje natural para escribir las instrucciones del código fuente. Además, el uso de estos elementos en el código fuente aumenta la expresividad y la claridad de este, haciendo incluso que el código fuente se auto-documente. De esta manera, es posible que cualquier persona ajena al proceso de ingeniería de *software*, sea capaz de leer y comprender las instrucciones escritas en el lenguaje naturalístico.

Con las bondades ya conocidas del lenguaje natural, y con herramientas desarrolladas hacia el esfuerzo de lograr una mayor expresividad al programar, se plantea la pregunta: ¿por qué la programación de todos los días no se realiza utilizando lenguajes naturales? Este trabajo presenta la propuesta de incluir lenguajes naturalísticos de propósito general en la formación de nuevos programadores, haciendo un recorrido por la fundamentación del paradigma, sus orígenes y lenguajes que lo representan. Además de una experiencia metodológica con un estudiante de maestría, basada en el desarrollo de una aplicación lúdica, usando un método de desarrollo propuesto por los autores, que utiliza un lenguaje naturalístico. Finalmente, se presentan los resultados de una prueba realizada con estudiantes de licenciatura, quienes desarrollaron un pequeño programa utilizando un lenguaje naturalístico de propósito general; la discusión y conclusiones correspondientes.

Fundamentación teórica

El paradigma naturalístico fue propuesto por Lopes y otros (2003), dada la necesidad de contar con programas más expresivos; es decir, más claros y cercanos al dominio del problema más que al de la solución. La propuesta consiste en incluir elementos del lenguaje natural en la construcción de lenguajes de programación de propósito general, dada la riqueza que este tiene para lograr aclarar las ideas.

Si bien la riqueza del lenguaje natural lo hace expresivo, también es inherentemente ambiguo. El utilizar al lenguaje natural como herramienta de programación, ha sido un sueño de las ciencias de la computación que data de muchos años atrás, sin embargo, ha sido muy difícil de lograr (Liu y Wu, 2018). En el caso de los lenguajes de programación naturalísticos, se propone obtener lenguajes de tipo natural, pero restringidos y formalizados para ser procesados por las computadoras, ya que esto sí es posible (Yin, 2010). No debe confundirse con procesamiento de lenguaje natural, ya que los lenguajes naturalísticos no infieren ni aprenden.

Pulido-Prieto y Juárez-Martínez (2017) analizaron varias herramientas que utilizan una gramática controlada del idioma inglés con un medio para traducir esta versión controlada, a un lenguaje de programación o a generadores de código de programación que controlan la ambigüedad. Ejemplos se pueden observar en los trabajos de Liu y Lieberman (2005), Yin (2010) y Cozzie y otros (2011). Sin embargo, los autores observaron que muchas de las tecnologías dependen de diccionarios de datos o bibliotecas pre-compiladas, lo cual reduce su uso a dominios particulares y aleja el objetivo de definir un modelo naturalístico de propósito general. También se encontraron con lenguajes con alto nivel de expresividad, con un proceso cognitivo similar al humano para lidiar con la ambigüedad; no obstante, estos están limitados a dominios particulares y utilizan un lenguaje natural controlado. Como resultado de su investigación, los autores propusieron, en Pulido-Prieto y Juárez-Martínez (2019), un modelo naturalístico que identifica los elementos del lenguaje natural que todo lenguaje naturalístico de propósito general debe incluir.

Lenguajes naturalísticos de propósito general

A continuación se presentan las características más importantes de los tres lenguajes naturalísticos de propósito general encontrados en la literatura, los cuales pueden generar código directamente ejecutable.

Pegasus

En Knöll y Mezini (2006) se realizaron los primeros esfuerzos de desarrollar un lenguaje de programación naturalístico mediante la construcción de un prototipo científico llamado Pegasus. Los autores definen por primera vez el término de programación naturalística como “Escribir programas de computadoras con la ayuda del lenguaje natural”. Los autores consideran que los principales problemas de los lenguajes de programación de alto nivel son:

- El problema mental. El programador se ve forzado a ajustar sus ideas a las condiciones de un lenguaje de programación específico. Por ejemplo, para un lenguaje orientado a objetos significaría estructurar las ideas del programa a la forma de clases, métodos y atributos.
- El problema del lenguaje de programación. El mismo programa debe

implementarse una y otra vez para estar disponible en los sistemas de computadora contemporáneos y los nuevos lenguajes de programación.
- El problema del lenguaje natural. Es ineficaz escribir, comentar y documentar software en inglés, el idioma más comúnmente utilizado, para los hablantes no nativos del idioma.
- El problema técnico. Los desarrolladores dedican la mayor parte del tiempo en depurar el programa, además de lidiar con problemas menores como elegir el conjunto de caracteres correcto, hacer conversiones numéricas, administrar el acceso a la base de datos, entre otros; en lugar de enfrentar la tarea de describir y mejorar la idea del programa.

Los autores identifican que el uso de los elementos del lenguaje natural en la programación genera los siguientes beneficios:
- Al escribir los programas en lenguaje natural, no es necesaria una transformación de ideas en otras estructuras de lenguajes como los objetos.
- La programación en lenguaje natural libera a los desarrolladores de la carga de aprender una y otra vez nuevos lenguajes de programación.
- Una vez que la idea se expresó en un lenguaje natural será válida durante mucho tiempo.
- La programación se volverá más fácil: escribir un programa de computadora será como explicar los pensamientos a un colega.

Con lo anterior en consideración se desarrolló el prototipo Pegasus. Su arquitectura tiene tres características básicas: leer el lenguaje natural, generar una salida de código de programación y expresar los resultados en otros idiomas del lenguaje natural (inglés, italiano, alemán, entre otros). Los autores están seguros de que la programación en lenguaje natural será una tendencia futura en el desarrollo de técnicas de programación, enfocadas hacia el pensamiento humano: “La ventaja de un lenguaje de programación naturalístico se trata del potencial para expresar y organizar ideas de una manera natural” (Knöll et ál., 2011).

Apoyándose de los conceptos previos se desarrolla un sistema de tipos naturalísticos con los conceptos de referenciación natural, generalización y descripción de instancia, con las siguientes características:
- Referencias naturales. Se basan en caracterizar objetos por sus propiedades. “Tomar la manzana roja” donde la referencia natural de la manzana es su propiedad roja.
- Generalización. Expresar enunciados generales; hechos generales sobre propiedades o relaciones de objetos. “El gato es un animal”, “Una casa tiene techo”.
- Descripciones de instancias. Los tipos naturalísticos podrían servir como “constructores”, ya que son capaces de iniciar una instancia al momento de crearla. “Una casa roja”: en este caso, la asignación de la propiedad color es implícita omitiendo redundantes asignaciones explícitas como “color = rojo”.

Estudiar el lenguaje natural no solo es importante para mejorar los lenguajes de programación, sino también para mejorar el estilo de programación dentro de los lenguajes existentes.

Las ideas detrás de la construcción de Pegasus promovieron la evolución del paradigma naturalístico, aunque no se cuenta con acceso a una versión de su compilador, lo cual se intentó a través de su sitio oficial en Knöll (2019) o vía comunicación personal con su autor (Knöll, comunicación personal, diciembre 2020). Para una mejor comprensión del paradigma, el siguiente código representa un ejemplo de lo propuesto por Pegasus (Knöll, 2019):

```
"If the quarterly VAT-declaration has not come in
until the 20th of the first month of the new quarter,
     send a reminder to the dutiable company."
```

La teoría dice que el código se auto-documenta, por lo que debería comprenderse su función sin necesidad de explicarla. Se deja al lector juzgar el nivel de expresividad del código.

Traducción de casos de uso a código fuente

En Mefteh y otros (2018) se presentó un nuevo enfoque hacia la programación naturalística. El enfoque adoptado toma requisitos en forma de casos de uso multilingües, a partir de los cuales un generador de código produce el código de destino (*target*). La riqueza de este nuevo enfoque consiste en la implementación de código XML que representa la información de los escenarios de casos de uso.

Para resolver el código XML en código de lenguaje de programación —Java, en este caso—, el enfoque opera en tres etapas. Primero, transforma los escenarios de casos de uso en patrones del modelo semántico creando las reglas de mapeo correspondientes. El modelo semántico permite que se utilice información al explorar e interpretar la sintaxis de los requisitos, independientemente del lenguaje. En la segunda etapa, el enfoque extrae la implementación de las reglas de mapeo generadas como un código XML siguiendo una estructura específica. En la tercera etapa, el código XML resultante se utiliza para construir las entradas que solicita un generador de código llamado ReDSeeDS. El generador es una herramienta que transforma las entradas en código Java siguiendo la arquitectura modelo-vista-presentador. El enfoque desarrollado, a diferencia de otros, es muy simple porque no requiere ningún estudio previo del idioma para utilizarse, gracias al modelo semántico. Además, el enfoque tiene el potencial de extraer la información adecuada de los escenarios de caso de uso ambiguos y generar entradas bastante completas y correctas para, en consecuencia, obtener un buen código Java.

Los autores evaluaron cuantitativamente la precisión del enfoque y determinaron que existe una precisión media del 87.43% al 89% en términos de modelos. Además, obtuvieron una precisión de entre el 64.4% y 93.43% en términos de eficiencia de código Java.

El enfoque desarrollado es útil para el proceso inicial de desarrollo de cualquier proyecto, recopilando información relacionada con el dominio de la aplicación. Al emplear esta herramienta, los desarrolladores podrían ahorrar tiempo al tener acceso a elementos de código bastante completos y correctos, a partir de los escenarios de casos de uso involucrados en la fase del análisis de la aplicación; sin embargo, el enfoque requiere la construcción de una API (Application Programming Interface) que comunique los modelos generados por el enfoque y los del generador de código ReDseeDS. Este generador usa formatos de modelo específicos no públicos, lo que ha limitado el desarrollo del potencial del enfoque.

SN (Sicut Naturali)

SN es un prototipo de lenguaje de programación naturalístico de propósito general basado en el idioma Inglés, propuesto por Pulido-Prieto y Juárez-Martínez (2020). El lenguaje puede trabajar con referencias indirectas, además de complementar una entidad para darle comportamiento especializado. Lo anterior, a partir de la composición de un sustantivo y varios adjetivos. SN soporta ciclos y condiciones y permite definir el contexto de una oración a partir de la asociación de una oración con otra de forma indirecta. La sintaxis de SN permite trabajar con diversas abstracciones que lo asemejan a un lenguaje natural.

SN es un lenguaje emergente que permite expresar código con una sintaxis simple y similar a la del idioma inglés, que incluye elementos naturales y permite tener un nivel alto de expresividad. SN cuenta con todas las propiedades ya documentadas de los lenguajes de programación naturalístico como lo es la ventaja de generar un código autodocumentado, dadas las construcciones similares a las de un lenguaje natural. Las pruebas realizadas en Pulido-Prieto y Juárez-Martínez (2020) dieron como resultado pequeños programas con instrucciones similares a oraciones, en las que los resultados mejoraron su expresividad según avanzaron las pruebas.

SN es una prueba de concepto del modelo propuesto en Pulido-Prieto y Juárez-Martínez (2019), el cual describe los elementos necesarios para diseñar lenguajes de programación naturalísticos de propósito general. Este modelo integra abstracción, elementos temporales y referencias indirectas en su gramática. El modelo presentado consta de cuatro elementos principales:

- El sustantivo. Es la palabra que se usa para identificar “una cosa” o un conjunto de “cosas”.
- El adjetivo. Se consideró como una abstracción que se usa para proporcionar las propiedades del sustantivo.

- El verbo. Representa acciones del sustantivo; es equivalente a las funciones en otros paradigmas, como en programación funcional o los métodos en la programación orientada a objetos.
- Escritura basada en propiedades. Permite usar propiedades para identificar una entidad particular.

Los autores consideraron que estos elementos brindan la información más relevante en una oración escrita en inglés. Los elementos seleccionados permitirán, además, el diseño de otros lenguajes de programación naturalísticos de propósito general sin requerir diccionarios de datos u otras técnicas. Se observó que SN, a partir del modelo implementado, contribuye a reducir la brecha entre los dominios del problema y de la solución, generando código auto-documentado.

Como resultado, SN requiere un proceso de aprendizaje centrado no sólo en la gramática sino también en adaptar el proceso cognitivo porque, aunque permite la programación de abstracciones como las encontradas en Java, cuando estas se implementan, el lenguaje pierde la naturalidad, lo cual va en contra de los objetivos de la programación naturalística. SN carece todavía de optimización suficiente para su uso en entornos reales; sin embargo, su trabajo está encaminado a lograr ese potencial; SN permite desacoplar las abstracciones y con ello aumentar la modularidad. Además, permite el uso de identificadores para el manejo de gran número de instancias.

Los investigadores sabían que el principal obstáculo del lenguaje natural es que es ambiguo y depende del contexto. Hasta ahora las computadoras procesan sin problema lenguajes de programación porque estos se basan en formalismos que no presentan ambigüedad. En SN las instrucciones se definen de tres formas: sujeto-verbo-objeto, verbo-objeto y verbo-objeto-preposición-objeto. La implementación del lenguaje SN dio como resultado un lenguaje que permite expresar instrucciones con un nivel de expresividad más cercano al idioma inglés, gracias a la gran cantidad de reglas que utilizan para su creación. El lenguaje permite reducir la ambigüedad de los lenguajes naturales gracias al modelo. El siguiente código muestra un programa que calcula el exponente de un número escrito con SN:

```
adjective Exponential:
     attribute states as some Integer Numbers.
     attribute prices are 25, 4 y 50.
     attribute nested as some Integer Numbers are 25.
     verb power num as Integer Number to itself that returns an
Integer Number:
          exp is 1.
          repeat the next instruction num times.
          exp is exp * value of this.
          return exp.
main Exponenciacion:
```

```
exp is 1.
num is 3.
number is 4.
repeat the next instruction num times.
exp is exp * number.
System prints exp.
```

Cal-4700

Compiler y Linker-4700 es un lenguaje naturalístico creado por padre e hijo (Rzeppa y Rzeppa, 2019), cuya idea principal es programar las ideas directamente como se piensan, como si se escribiera un pseudocódigo, sin necesidad de traducirlas a un lenguaje de programación. Los autores querían dar respuesta a las siguientes preguntas:

1. ¿Es más fácil programar cuando no tienes que trasladar tus pensamientos en lenguaje natural a sintaxis alternas?
2. ¿Pueden los lenguajes naturales analizarse de una manera descuidada (como aparentemente lo hacen los seres humanos), y aun así proveer de un ambiente lo suficientemente estable para la programación productiva?
3. ¿Pueden los programas de bajo nivel (como los compiladores) escribirse cómoda y eficazmente en lenguajes de alto nivel (como el inglés)?

Los autores respondieron afirmativamente a estas preguntas con su compilador Cal-4700, escrito en inglés plano, con cerca de 25,000 frases de código fuente descargable desde su página principal en (Rzeppa y Rzeppa, 2019). Los autores lo definen como un lenguaje vivo, ya que el mismo programador puede definir tipos y rutinas que hacen que el lenguaje evolucione. El siguiente es un fragmento de código en Cal-4700 que ordena una lista de frutas:

```
To sort some fruits:
   If the fruits' first is the fruits' last, exit.
   Split the fruits into some left fruits y
          some right fruits.
   Sort the left fruits.
   Sort the right fruits.
Loop.
    Put the left fruits' first into a left fruit.
    Put the right fruits' first into a right fruit.
    If the left fruit is nil,
          append the right fruits to the fruits; exit.
    If the right fruit is nil,
          append the left fruits to the fruits; exit.
    If the left fruit's name is greater
          than the right fruit's name,
```

```
    Move the right fruit from the right fruits
          to the fruits; repeat.
    Move the left fruit from the left fruits to the fruits.
Repeat.
```

Como se puede observar en el código anterior, Cal-4700 hace uso de frases imperativas y posesivos en inglés. Todo procedimiento comienza con la palabra “To”.

Comparación entre lenguajes

En Hernández-González y otros (2021) se realiza una revisión de los tres lenguajes naturalísticos de propósito general que pueden generar código ejecutable: Pegasus, SN y Cal-4700. Estos, a consideración de los autores, son los lenguajes de programación naturalísticos más representativos en la actualidad, que trazan una línea de tendencia en la evolución del paradigma de programación naturalístico de acuerdo a una comparación de sus características y ejemplos realizados en los tres lenguajes. En general, los tres lenguajes crean declaraciones que utilizan elementos comunes del lenguaje natural como los sustantivos. Además, los tres lenguajes pueden identificar frases, hacer referencias anafóricas, ejecutar instrucciones de software (como responder a eventos y definir declaraciones condicionales) y usar iteradores naturalísticos.

Pegasus no permite definir tipos, sustantivos ni adjetivos de manera explícita, de acuerdo a lo reportado en la literatura, se infiere que están definidos en un diccionario, sin embargo, estos son factibles de usarse en el lenguaje, de acuerdo al sitio web oficial que muestra su forma de uso. SN y Cal-4700 pueden usar sustantivos (“cosas”) usando declaraciones en singular y plural y permiten al programador administrar el conjunto de “cosas”. Cal-4700 a diferencia de SN y Pegasus, que no usan estructuras gramaticales estrictas, funcionan mediante instrucciones imperativas guiadas de una estructura flexible que no realiza validaciones gramaticales sino de similitud. Para su ejecución, SN no necesita archivos externos, simplemente se instala como cualquier otro lenguaje de programación. Pegasus necesita una base de datos con significados y Cal-4700 cuenta con su propio entorno que incluye un archivo de compilación y una biblioteca de acciones. Pegasus, al usar un generador de código, permite a los usuarios elegir el lenguaje de programación resultante, mientras que SN y Cal-4700 compilan directamente a código ejecutable. En Cal-4700 resalta la ventaja de que consigue incrementar el dominio de su lenguaje, al permitirle a los programadores agregar definiciones no existentes a su biblioteca, además, es posible nombrar de diferentes maneras a las definiciones existentes para hacer que el lenguaje sea más adecuado al estilo de sintaxis del programador.

Se concluyó que Pegasus, Cal-4700 y SN ofrecen más de lo necesario para denominarlos lenguajes de uso general, siguen la filosofía de la programación

naturalística al integrar elementos del lenguaje natural, ser reflexivos y generar programas expresivos y ejecutables. Sin embargo, en la práctica, SN y Pegasus no se encuentran disponibles al público ya que aún están en fase de desarrollo. Cal-4700 sí cuenta con una versión estable disponible al público a través de su sitio web (Rzeppa y Rzeppa, 2019), la cual incluye documentación y ejemplos, por lo que se consideró el lenguaje más estable para los estudios de caso presentados en este trabajo.

Método de desarrollo naturalístico

Como una apuesta más al paradigma, surge una propuesta de método para el análisis y diseño naturalísticos hecha por Hernández-González y otros (2024). El objetivo de la propuesta es aportar formalidad al proceso de desarrollo naturalístico, mediante la elaboración de artefactos de análisis y diseño que permiten guiar la construcción del software bajo este enfoque.

Metodología basada en el desarrollo naturalístico

El presente capítulo se enmarca en un enfoque cualitativo de tipo aplicado, mediante el uso del estudio de caso como estrategia metodológica. La investigación se centró en explorar el uso del lenguaje de programación naturalístico Cal-4700, el más estable según las investigaciones, en un entorno educativo real, con el objetivo de evaluar su expresividad, comprensión y aplicabilidad en estudiantes con formación previa en lenguajes de programación convencionales.

Enfoque y diseño metodológico

El enfoque cualitativo fue seleccionado debido a que se pretendía comprender las experiencias de los estudiantes al interactuar con un paradigma emergente que prioriza la expresividad sobre la formalización técnica tradicional. Primero se probó el enfoque adaptándolo a un contexto educativo lúdico, en el cual se desarrolló una aplicación interactiva dirigida a niños pequeños. En un segundo estudio se diseñó un caso único con una intervención pedagógica breve, que consistió en una práctica guiada de programación utilizando Cal-4700 (Rzeppa y Rzeppa, 2019).

Participantes

Para el primer estudio se contó con el apoyo de un estudiante de la maestría en Sistemas Computacionales del Instituto Tecnológico de Orizaba (ITO); para el segundo estudio el grupo participante estuvo conformado por cinco estudiantes de séptimo semestre de la carrera de Ingeniería en Sistemas Computacionales del ITO. Todos los participantes contaban con experiencia previa en programación imperativa y orientada a objetos, lo que permitió establecer un punto de comparación con el paradigma naturalístico.

Técnicas e instrumentos

El primer estudio de caso fue reportado como parte del trabajo de grado del estudiante de maestría, quien trabajó en él durante año y medio aproximadamente, realizando revisiones semanales de sus avances. Durante el primer semestre de su posgrado, los autores de este trabajo impartieron un curso de Programación Orientada a Aspectos; así como de Programación Naturalística. El estudiante tomó dicho curso para contar con las bases necesarias para realizar su trabajo.

Para la práctica con los estudiantes de licenciatura, se utilizaron tres técnicas principales para la recolección de datos:

1. Observación directa, a través de videollamada con cámara encendida para registrar reacciones espontáneas durante el proceso de codificación.
2. Análisis del desempeño, mediante la elaboración de un programa funcional que calcula la serie de Fibonacci en Cal-4700.
3. Encuesta de percepción, construida con base en una escala Likert y preguntas abiertas, para conocer la experiencia del estudiante respecto al paradigma y el lenguaje utilizado.

Procedimiento

El procedimiento metodológico se dividió en seis etapas:

1. Introducción a la práctica y explicación de objetivos.
2. Descarga y exploración del entorno de Cal-4700 con ejemplos funcionales.
3. Breve presentación teórica del paradigma naturalístico.
4. Ejecución de ejemplos en vivo por parte de los estudiantes.
5. Desarrollo individual del algoritmo de Fibonacci usando Cal-4700.
6. Respuesta a la encuesta de evaluación y reflexión final.

Análisis de resultados

Los resultados fueron interpretados a partir de la observación cualitativa de las reacciones, el análisis del código producido y la sistematización de respuestas en la encuesta. Se consideraron las dimensiones de expresividad, comprensión, facilidad de uso, utilidad potencial y disposición a continuar el aprendizaje con Cal-4700.

Resultados

Desarrollo de una aplicación lúdica

El método de desarrollo de software naturalístico propuesto por Hernández-González y otros (2024) fue aplicado para desarrollar una aplicación lúdica utilizando un lenguaje naturalístico (Juárez-Romero et al., 2022). El sistema consta de una aplicación lúdica que tiene como objetivo que niños de entre 5 y 7 años (los jugadores), aprendan a usar

los elementos de la computadora jugando. El juego es un puzzle o rompecabezas, en el que, a través del uso del ratón, al hacer clic y arrastrar, el jugador podrá ordenar las piezas desordenadas de una imagen dividida en secciones.

Juárez-Romero y otros (2022) generaron los artefactos de análisis y diseño que propone el método de desarrollo para generar código naturalístico utilizando el lenguaje Cal-4700. En la figura 1 se presentan dos pantallas de la aplicación lúdica, la pantalla principal para elegir el nivel de dificultad y la figura 2 con la imagen seleccionada en nivel fácil y las piezas ya divididas, listas para armar. La intención es que el lector visualice el tipo de aplicaciones que se pueden realizar con Cal-4700.

Figura 1 Selección de nivel del rompecabezas (Juárez-Romero et ál., 2022)

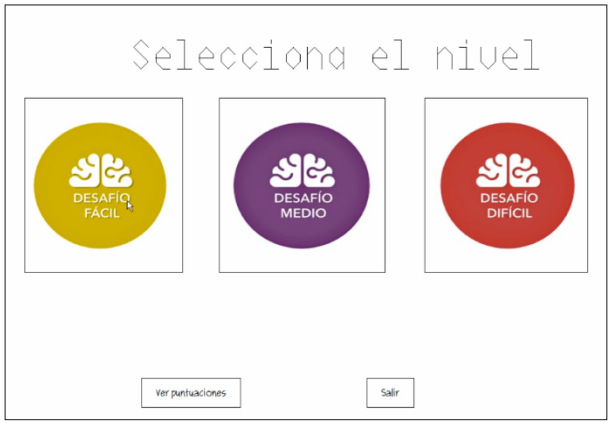
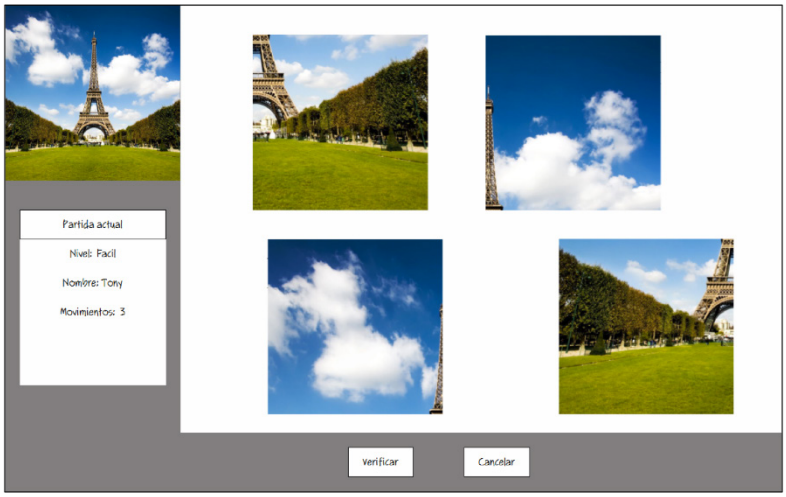


Figura 2 Imagen seleccionada ya dividida en partes (Juárez-Romero et ál., 2022)



El estudio demostró que es posible aplicar tanto el método de desarrollo como el lenguaje naturalístico Cal-4700 en la construcción de software de propósito general.

Pruebas guiadas con Cal-4700

Como se mencionó en el apartado anterior, al finalizar la práctica guiada, se les pidió a los estudiantes contestar una encuesta de forma anónima, para recabar su opinión acerca del paradigma y del lenguaje Cal-4700. La encuesta y sus resultados se observan en la tabla 1.

Sólo dos estudiantes terminaron completamente el programa, el resto comentó que pudo programar y ejecutar algunas funciones, pero no podían permanecer más tiempo en la sesión.

No se observó alguna expresión de molestia o frustración, más bien parecían muy concentrados revisando la documentación y realizando pruebas. En la tabla 1 se observan los resultados de la encuesta, para cada una de las preguntas se contabilizaron las respuestas, las cuales se encuentran en la escala Likert con los siguientes valores:

- Totalmente en desacuerdo
- En desacuerdo
- Neutral
- De acuerdo
- Totalmente de acuerdo

Como se puede observar, cuatro de los cinco estudiantes respondieron favorablemente a cada una de las preguntas. Uno de los estudiantes respondió desfavorablemente; sin embargo contestó que sí utilizaría Cal-4700 para resolver problemas, además de que recomendaría su uso y el de los lenguajes naturalísticos en general.

Los comentarios y sugerencias que se recabaron en la encuesta son los siguientes:

- “Me pareció muy interesante en aprender a utilizar Cal-4700 mediante el lenguaje naturalista, es algo muy nuevo en lo personal para mí, pero me pareció muy práctico, considero que si se requiere algo de practica ya que estamos muy acostumbrados a lenguajes con una sintaxis y estructura muy estricta, con sentencias ya asignadas, y en este lenguaje naturalista podemos implementar nuevas sentencias, me gustó mucho esta platica y por supuesto que me gustaría seguir practicando. No logre terminar la practica realizada durante la plática, pero la terminare ya que me quede con ganas de aprender más”.
- “Es una interfaz fácil de utilizar, en mi caso, la sintaxis es un poco complejo [sic], ya que viene en inglés y me es un poco difícil recordar las sentencias”.
- “El modo de explicación, aprendizaje es muy eficiente al ser algo sencillo y tener conocimientos previos”.
- “Me pareció muy interesante la forma de programar, y sin duda me gusto [sic] y aprendería más sobre Cal-4700”.

El análisis evidenció una actitud favorable hacia el paradigma naturalístico, destacando la expresividad del lenguaje y su potencial educativo. Las principales limitantes percibidas fueron el idioma inglés y la novedad del enfoque. A pesar de ello, todos los participantes manifestaron interés en seguir utilizando este tipo de lenguaje en contextos reales de programación.

Tabla 1 Encuesta de evaluación de Cal-4700 con resultados.

	Totalmente en desacuerdo	En desacuerdo	Neutral	De acuerdo	Totalmente de acuerdo
1. Considero que Cal-4700:					
a. Es fácil de utilizar.	1		1		3
b. Es fácil de recordar.	1			2	2
c. Es eficiente.	1			2	2
d. Es flexible.	1			1	3
e. Su interfaz tiene manejo adecuado de errores.	1			2	2
f. Su interfaz minimalista es práctica.	1			3	1
g. Su interfaz es fácil de usar.	1			3	1
h. Su interfaz es fácil de recordar.	1			1	3
i. Es consistente con el paradigma.	1			1	3
j. Proporciona ayuda y documentación suficiente.	1			1	3
	SÍ			NO	
2. Utilizaría Cal-4700 para resolver problemas.	5				
3. Recomendaría utilizar Cal-4700.	5				
4. Recomendaría utilizar lenguajes naturalísticos.	5				

Discusión y conclusiones

La programación naturalística busca reducir la brecha entre el dominio del problema y el dominio de la solución, incorporando elementos del lenguaje natural como la reflexividad en los lenguajes de programación. Los lenguajes naturalísticos presentados muestran que la utilización de un subconjunto controlado del idioma inglés, como lenguaje de programación, permite alcanzar los objetivos de reducir la complejidad y ofrecer mayor expresividad al código fuente. Usar los lenguajes naturalísticos favorece que el programador se enfoque en resolver el problema del cliente, más que en resolver los problemas que la propia tecnología le presenta. La traducción de las ideas sería transparente, sin deformarlas.

Cal-4700 es lo suficientemente robusto para aplicaciones pequeñas a medianas de propósito general, tal como se observó en el estudio de caso lúdico. Su descarga es gratuita y cuenta con la documentación y ejemplos suficientes. Su aceptación en la primera prueba es alentadora, por lo que promover su uso en la enseñanza de la programación se muestra viable. Los estudiantes se enfocarían en diseñar los problemas a través de pseudocódigos, utilizando una gramática reducida del inglés, con un lenguaje que puede evolucionar de acuerdo a sus necesidades. Los códigos se crearían autodocumentados, siendo más comprensibles para los programadores.

Durante la formación profesional, las asignaturas de programación suelen tener índices de reprobación altos, por lo que esta podría ser una buena propuesta a considerar, sin descartar, por supuesto, la enseñanza tradicional.

Esta experiencia demuestra que, mediante una estructura metodológica adecuada, es posible evaluar con rigor el impacto educativo de lenguajes emergentes como Cal-4700, aportando evidencia útil para futuras aplicaciones y adaptaciones curriculares en el ámbito de la programación.

Referencias

Cozzie, A., Finnicum, M. y King, S. (2011). Macho: programming with man pages. In Proceedings of the 13th USENIX conference on Hot topics in operating systems, pp. 7–7.

Hernández-González, L. A., Juárez-Martínez, U. y Alducin-Francisco, L. M. (2021). Evolution of Naturalistic Programming: A Need. In J. Mejia, M. Muñoz, l. Rocha, y Y. Quiñonez (Eds.), New Perspectives in Software Engineering, Advances in Intelligent Systems y Computing, Cham, pp. 185–198. Springer International Publishing.

Hernández-González, L. A., Juárez-Martínez, U., Mejía, J. y Aguilar-Laserre, A. (2024). A Proposal of Naturalistic Software Development Method. JUCS - Journal of Universal Computer Science 30(2), 179–203.

Juárez-Romero, J. A., Juárez-Martínez, U., Hernández-González, L. A., Rodríguez-Mazahua, L. y Abud-Figueroa, M. A. (2022). Implementación práctica del paradigma naturalístico mediante un caso de estudio lúdico. Revista Aristas: Investigación Básica y Aplicada 9(17), 1–6.

Knöll, R. (2019). Pegasus Natural Programming. TU Darmstadt. http://www.pegasus-project.org/en/Welcome.html.

Knöll, R., Gasiunas, V. y Mezini, M. (2011). Naturalistic Types. In Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, y Reflections on Programming y Software, Onward! 2011, New York, NY, USA, pp. 33–48. ACM. event-place: Portly, Oregon, USA.

Knöll, R. y Mezini, M. (2006). Pegasus: First Steps Toward a Naturalistic Programming Language. In Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, y Applications, OOPSLA '06, New York, NY, USA, pp. 542–559. ACM. event-place: Portly, Oregon, USA.

Liu, H. y Lieberman, H. (2005). Metafor: visualizing stories as code. IUI '05, San Diego, California, USA, pp. 305–307. Association for Computing Machinery.

Liu, X. y Wu, D. (2018). From Natural Language to Programming Language, Chapter 4, pp. 110–130. IGI Global.

Lopes, C. V., Dourish, P., Lorenz, D. H. y Lieberherr, K. (2003). Beyond AOP: Toward Naturalistic Programming. SIGPLAN Notices 38(12), 34–43.

Mefteh, M., Bouassida, N. y Ben-Abdallah, H. (2018). Towards naturalistic programming: Mapping language-independent requirements to constrained language specifications. Science of Computer Programming 166, 89–119.

Pulido-Prieto, O. y Juárez-Martínez, U. (2017). A Survey of Naturalistic Programming Technologies. ACM Comput. Surv. 50(5), 70:1–70:35.

Pulido-Prieto, O. y Juárez-Martínez, U. (2019). A Model for Naturalistic Programming with Implementation. Applied Sciences 9(18), 3936.

Pulido-Prieto y Juárez-Martínez (2020). Naturalistic Programming: Model y Implementation. IEEE Latin America Transactions 18(07), 1230–1237.

Rzeppa, G. y Rzeppa, D. (2019). The Osmosian Order of Plain English Programmers

Blog. https://osmosianplainenglishprogramming.blog.

Yin, P. (2010). Natural Language Programming Based on Knowledge. In 2010 International Conference on Artificial Intelligence y Computational Intelligence, Volume 2, pp. 69–73.