# Una base de conocimientos para asistir el aprendizaje de la programación

Juan P. Ucán, Antonio A. Aguileta, Raúl A. Aguilar

Universidad Autónoma de Yucatán, Facultad de Matemáticas, México

{juan.ucan, aaguilet, avera}@correo.uady.mx

**Resumen.** Con el propósito de ampliar la investigación en el ámbito del aprendizaje de la programación, se ha realizado un trabajo de investigación que involucra, como estrategia de análisis de nuevas opciones instruccionales, el desarrollo de un Entorno Virtual Colaborativo Inteligente (EVCI). En este artículo se presenta la arquitectura del EVCI, en particular, se describe la Base de Conocimientos del Sistema Experto, la cual es integrada a manera de componente al EVC. Las principales funcionalidades del Sistema Experto, y por tanto prestaciones del EVCI son también descritas en el artículo.

**Palabras clave:** EVC, EVCI, sistema experto.

# A Knowledge Base to Assist the Learning of the Programming

**Abstract.** In order to expand research about the learning of the programming, it has done a research that involved as strategy of analysis about new instructional options, development of an Intelligent Collaborative Virtual Environment (EVCI). In this paper we describe an EVCI architecture, particularly the Knowledge Base of Expert System, which is integrated to EVC as a component. The main features of Expert System and EVCI are also described in the paper.

**Key Words:** EVC, EVCI, expert system.

## 1. Introducción

El aprendizaje de la programación, a nuestros días aún no resulta una tarea resuelta en el ámbito de la Educación en Informática. El primer lenguaje seleccionado, el primer paradigma abordado, los errores comunes cometidos por los aprendices, así como las actividades y habilidades cognitivas —independientes del lenguaje y/o paradigma— propias del dominio, son algunos de los factores inmersos en dicha actividad educativa. En este artículo se aborda dicha problemática desde una línea de investigación e

innovación conocida como Entornos Virtuales Inteligentes, en particular, Entornos de Aprendizaje basados en la Colaboración y el Conocimiento.

El trabajo reportado se enmarca en un proyecto de investigación en el que estableció como pregunta de investigación, la siguiente:

"…¿Es posible implementar un Modelo para Asistir en el Aprendizaje de la Programación mediante un Entorno Virtual Colaborativo Inteligente, que facilite al aprendiz la detección de errores comunes de la programación? "[1].

Para dar respuesta a la pregunta de investigación, se propuso el desarrollo de un Marco de Trabajo para Asistir el Aprendizaje de la Programación en Espacios de Trabajo Compartidos y experimentar con dicha propuesta utilizando un Entorno Virtual Inteligente (EVI) diseñado exprofeso. En el presente artículo se describe de manera específica el "Componente Inteligente" diseñado e incorporado al Entorno Virtual de Aprendizaje, dicho componente se corresponde con un Sistema Experto Basado en Reglas que utiliza un motor de inferencia instituido en los errores comunes de la programación.

## 2. Antecedentes

### 2.1. Aprendizaje de la programación

La Programación no es una disciplina fácil de aprender y mucho menos de enseñar, ha resultado compleja en su aprendizaje debido a la variedad de sub-tareas y tipos de conocimiento especializados que involucra, y que son necesarios de ejecutar de manera efectiva [2], por otro lado, es una disciplina que debe ser aprendida bajo un enfoque constructivista, es decir, en buena medida se aprende haciendo, y en muchas ocasiones, requiere de un enfoque por aproximaciones sucesivas para poder generar aprendizajes significativos.

En el caso de su enseñanza, la selección del primer lenguaje de programación [3] ha generado discusiones continuas entre los especialistas; vinculado con el lenguaje, el paradigma abordado en los primeros cursos de Informática también ha generado posiciones encontradas, no obstante, y sin un análisis suficientemente discutido, la aparición de enfoques orientados a objetos ha venido desplazando al paradigma estructurado; otros autores por su parte, se han enfocado al aspecto didáctico, y han propuesto algunos modelos para enseñar a programar [4].

Independientemente del proceso educativo a desarrollar, la Programación, como actividad socio-técnica, involucra actividades dependientes del lenguaje, las cuales requieren del conocimiento de la sintaxis, semántica y de la pragmática del lenguaje de programación [5]; por otro lado, requiere de un amplio rango de actividades vinculadas con el proceso de desarrollo software: análisis, diseño, codificación, comprensión de programas, depuración, mantenimiento y documentación.

En el ámbito de la Informática Educativa se han desarrollado diversos sistemas software para asistir el aprendizaje y/o enseñanza de la Programación [6], [7] y [8], sin embargo, para las actividades vinculadas con la depuración y comprensión de programas, tareas a las que no siempre se les otorga la importancia debida, los sistemas software desarrollados, generalmente se orientan al entendimiento del lenguaje, y resultan poco útiles para el entrenamiento de programadores novatos.

## 2.2. Errores comunes en programación

De acuerdo con el estándar IEEE 610.12-1990 [9], un fallo en el sistema ocurre cuando la salida generada por un programa no coincide con los requisitos establecidos en el proyecto. De ahí que un fallo puede ser resultado de una falta, la cual representa un estado en tiempo de ejecución de un programa, que parece ser incorrecto. Las faltas ocurren como resultado de los errores, los cuales son fragmentos del programa que no cumplen con las especificaciones documentadas, o con las especificaciones del modelo mental de los programadores.

Es importante mencionar, para efectos de la tarea de depuración, que aunque los fallos en la ejecución de los programas garantizan que una o más faltas se encuentran en el código —ya sea por omisión o por comisión del programador—, y que las faltas garantizan que uno o más errores existen, los errores no siempre causan faltas, y las faltas no siempre generan fallos.

En la literatura es posible encontrar una variedad de clasificaciones sobre tipos y causas de errores que pueden ser realizados, principalmente por programadores novatos; dichas clasificaciones han sido generadas a raíz de estudios empíricos sobre patrones encontrados en la actividad de programación. En [10] por ejemplo, se llegó a la conclusión de que una causa de error en la programación, se debe a que los programadores se forman modelos mentales de las especificaciones documentadas, no obstante, dichos modelos resultan insuficientes por las especificaciones inadecuadas, o por el desconocimiento del dominio de la tarea a ser implementada.

Un segundo tipo de errores comunes está relacionado con la sintaxis propia de algunos lenguajes, por ejemplo: el clásico ";" al final de una sentencia en C, la falta de una "}" para delimitar un ciclo o una función, la confusión de los símbolos para realizar asignaciones en lugar de comparaciones, etc.

```
if (num1 == (num2+num3))….        Correcto
if (num1 = (num2+num3))….                 Incorrecto
```

Otros errores tienen relación con la lógica de las estructuras algorítmicas, en estos, es común encontrarse en las estructuras repetitivas con errores en el número de iteraciones deseadas, p.ej., si se desea ejecutar el contenido del ciclo *for* diez veces.

```
for (i=0; i < 10; i++)…….        Correcto
for (i=0; i <=10; i++)…….        Incorrecto (ejecuta once veces)
```

En [11] se presenta una revisión exhaustiva de las principales clasificaciones de errores en el ámbito de la programación y se profundiza en torno a nuevas maneras de entender la ejecución de un programa cuestionando las salidas del programa, también se realiza un análisis de algunos IDEs y lenguajes de programación usados en la actualidad y aporta una taxonomía de defectos comunes de la programación.

## 2.3. Sistemas expertos

Como una aplicación exitosa del área de Inteligencia Artificial (IA), surgieron los en la década de los 60 los denominados Sistemas Expertos; al principio, este tipo de sistemas se desarrolló como herramienta de investigación para tratar problemas

complejos en un dominio estrecho, por ejemplo, el diagnóstico médico de enfermedades en ese entonces, pero desde su introducción comercial a principios de los 80, incrementó su popularidad. Actualmente los sistemas expertos son usados en los negocios, ciencia, ingeniería, manufactura y muchos otros campos en los cuales existe un problema en un área o en un dominio bien definido. Feingenbaum, profesor de la Universidad de Stanford, y considerado como el padre de los sistemas expertos, presentó la siguiente definición para un sistema experto:

"…es un programa de cómputo inteligente que usa el conocimiento y procedimientos de inferencia para resolver problemas suficientemente complicados que requiere significativamente de un experto humano para su solución" [12].

De acuerdo con [13], en un sistema experto se pueden identificar tres componentes principales: base de conocimientos, motor de inferencia e interface de usuario. La figura 1 ilustra una posible arquitectura [14].

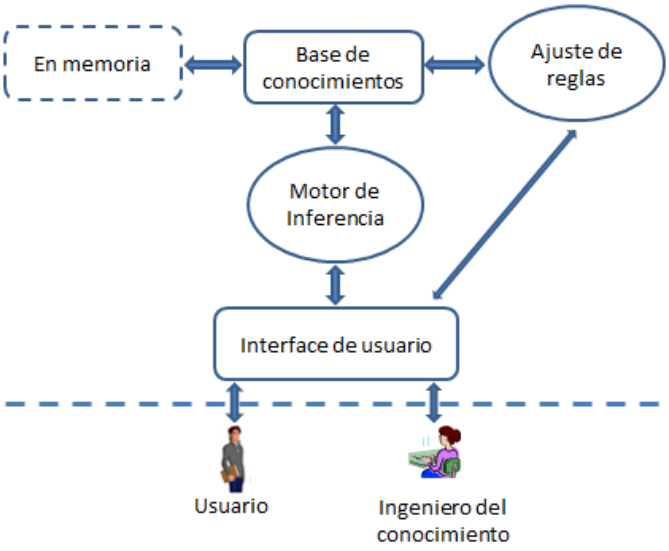


**Fig. 1.** Estructura de un sistema experto.

Base de conocimientos. Es la base de un sistema experto, contiene el conocimiento, datos del problema necesarios para entender, formular y suministrar propuestas de solución. Existen diferentes formas de representar el conocimiento de un sistema experto, incluyendo reglas de producción, redes semánticas y sentencias lógicas. Una base de conocimientos típica debe incluir dos elementos básicos:

- Hechos tal como la situación del problema y la teoría del área del problema.
- Heurísticas especiales o reglas que dirigen el uso del conocimiento para resolver los problemas específicos en un dominio particular.

Motor de Inferencia. Es el cerebro de un sistema experto, también se le conoce como estructura de control o el intérprete de reglas. Este componente es esencialmente un programa de computadora que proporciona una metodología para el razonamiento sobre la información en la base de conocimientos, en el área de memoria y para la formulación de conclusiones.

Interface de usuario. Este componente establece la comunicación orientada al problema entre el usuario y la computadora, dicha comunicación se realiza en un lenguaje natural. La interface de usuario en este tipo de sistemas, para su interacción, en su mayoría usa preguntas y respuestas. En la Figura 1 se visualizan dos tipos de usuarios:

- El usuario, quien utiliza el sistema experto para hacer consultas y ayudarlo a tomar decisiones.
- El ingeniero del conocimiento, quien se encarga del mantenimiento de la base de conocimientos, es la interface entre el humano experto y el sistema experto.

Como se observa en la figura 1, en [14] en dicha arquitectura se consideran dos elementos adicionales: lo que se mantiene en Memoria, y las reglas ajustadas durante la operación del sistema experto.

Memoria. El contenido de este elemento consiste de los hechos generados de acuerdo a un problema específico en una sesión de consulta, es decir los nuevos hechos que resultaron de un proceso de inferencia.

Ajuste de reglas. Se trata del ajustador de reglas, es decir la entrada de reglas especificadas por el ingeniero del conocimiento dentro de la base de conocimientos durante el desarrollo del sistema experto o mantenimiento de la base de conocimientos.

## 3. Metodología

Como parte de la investigación antes citada [1], se desarrolló un sistema experto basado en reglas, capaz de identificar los tipos de defectos comunes en la programación; dicho sistema experto se integró como un componente al Entorno Virtual Colaborativo (EVC) que fue diseñado exprofeso para la investigación, en particular para el proceso de experimentación. En términos generales, las actividades que se planificaron como parte de la metodología para la construcción del Entorno Virtual Colaborativo Inteligente (EVCI) son:

1. Diseño una Arquitectura para un EVCI.
2. Implementación del Entorno Virtual Colaborativo.
3. Formalización de las reglas que permitirán incorporar al EVC una Base de Conocimiento sobre los defectos comunes en programación.
4. Implementar el componente inteligente (Sistema Experto) al EVC.
5. Realizar estudios con experimentos controlados en ingeniería de software utilizado el EVCI.
6. Reporte de resultados.

## 4. Desarrollo

### 4.1. Arquitectura propuesta para el EVCI

Un Entorno Virtual Colaborativo Inteligente (EVCI) es un sistema computacional diseñado exprofeso como un espacio conceptual para que el usuario, en condiciones

espacio-temporales distintas, interaccione con otros usuarios o con elementos del entorno para construir su aprendizaje; el componente inteligente es un elemento del entorno provisto de una base de conocimientos y de una estrategia pedagógica basada en la formulación de consultas por parte del aprendiz [1].

La Figura 2 ilustra mediante un diagrama la arquitectura que se diseñó y utilizó para la implementación del EVCI presentado en esta investigación.

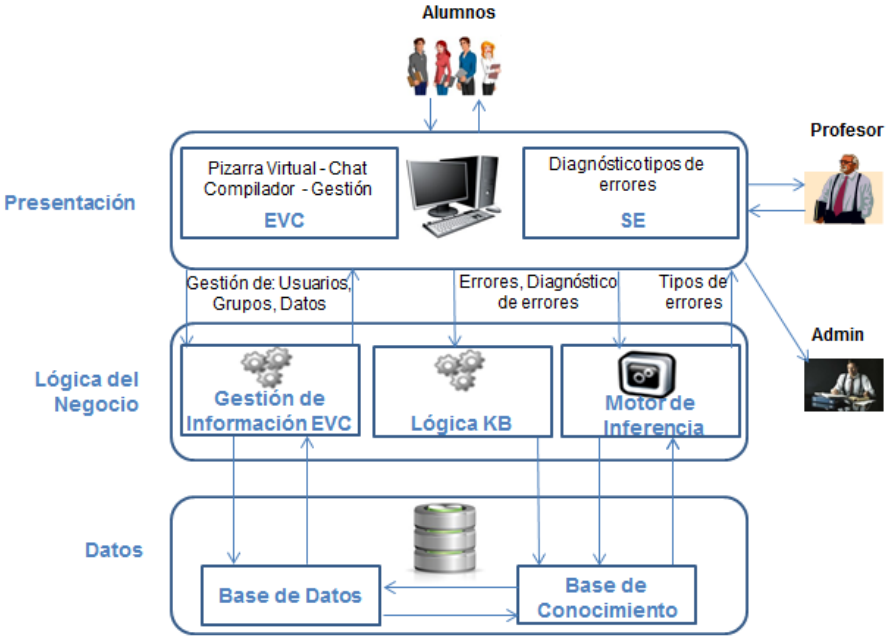


**Fig. 2.** Arquitectura del EVCI.

La arquitectura del EVCI consta de la integración de los ambientes de un entorno virtual colaborativo (EVC) y de un Sistema Experto (SE). Esta arquitectura se puede visualizar en una arquitectura por capas [15], donde los usuarios principales del sistema son los alumnos; los alumnos pueden visualizar en el EVC trozos de código de programas implementados bajo un paradigma de programación seleccionado (p.e. el paradigma estructurado) y deben encontrar los defectos del programa que se les presente, lo podrán realizar en grupos trabajo, consultando con el profesor o con un SE integrado al EVCI que permite identificar los tipos de defectos comunes de la programación.

### 4.2. Base de conocimiento

En esta sección, se establece la base de conocimiento sobre los tipos de defectos más comunes de la programación que se implementó en el sistema experto, componente inteligente del EVCI. En este trabajo se abordó el problema del aprendizaje y/o enseñanza de la programación mediante entornos virtuales, utilizando como parte del modelo del dominio, un modelo de perturbaciones o de defectos clásicos en el dominio del aprendizaje [16].

Para representar la base de conocimiento en esta investigación se utilizaron reglas de producción, estas reglas se generan relacionando los datos de la base de hechos. Cada regla está formada de una parte denominada premisa y de una parte denominada conclusión y tendrá la siguiente forma:

– Si premisa Entonces conclusión.

El sistema EVCI ha sido concebido para el entrenamiento de los aprendices en la identificación de defectos, en este sentido, debido a que la relación entre los tipos de defectos cometidos por los aprendices y el diagnóstico no tiene un comportamiento determinístico, se ha decidido considerar un modelo probabilístico para el sistema experto, es decir, su motor de inferencia se basa en probabilidades condicionales. En este esquema se consideran todas las evidencias, por lo que también se pueden modelar reglas de la forma:

– Si premisa1 And premisa2 Or premisa3 Entonces conclusión.

Los sistemas basados en reglas surgieron de sistemas prácticos e intuitivos para la inferencia lógica. Estos métodos fueron desarrollados a mediados de los setentas y han formado las bases de un gran número de sistemas expertos en medicina y en muchas otras aplicaciones [15].

**Tabla 1.** Atributos y valores de los defectos (errores) sistemáticos de la programación.

| | Tipos de errores comunes o sistemáticos | | | | | |
|---|---|---|---|---|---|---|
| **Diagnósticos de error (DxE)** | **Inicialización** | **Cómputo** | **Control** | **Datos** | **Interface** | **Cosmético** |
| Diseño lógico | | Caso imprevisto, comisión | Terminación abrupta | | | Omisión |
| | | Error en el algoritmo, comisión | | | | |
| Sobrecarga de memoria | Subíndice fuera de límite, comisión | Liberación de memoria, omisión | Validación de memoria, omisión. | | Terminación abrupta | |
| Léxico | | Análisis inadecuado, comisión | Síntaxis ambigua, comisión | | | |
| Variable | Errores por tipos diferentes, comisión | Resultados incorrectos, comisión | | Resultados incorrectos, comisión | | |
| | Inicialización inapropiada de una estructura de datos comisión | | | | | |
| Lenguaje | | Interpretaciones erroneas en la semántica del lenguaje | | | | |

Una de las actividades primordiales para el desarrollo de un sistema experto, es la adquisición del conocimiento posteriormente codificado en la base de conocimientos,

una base de datos existente y apropiada dentro de un conjunto de reglas de producción. La fuente de este conocimiento son los defectos comunes de la programación, estudiado en el marco teórico.

Debido a la gran cantidad de lenguajes de programación, para poder realizar la experimentación con el EVCI, se decidió una versión reducida de la estructura identificada como fuente de conocimientos, en específico, en el lenguaje de programación C, los objetos son los seis tipos de defectos comunes de la programación propuestos en [17] y los atributos y valores de dichos objetos son los investigados en [11]. En la Tabla 1 se presenta una estructura utilizada para la construcción de la base de conocimientos.

Los defectos representan el aspecto más importante ya que son estos elementos los que en mayor medida permiten encontrar un tipo de defecto específico basado en la presencia o ausencia de ciertos indicadores.

Los diagnósticos de defecto son indicadores que también forman parte del proceso de obtención de tipos de defectos diferenciales. Los diagnósticos de defecto permiten a los profesores de programación o expertos, confirmar un tipo de defecto a través de los resultados reportados, es por eso que se han tomado en cuenta como un tipo de indicador.

En la siguiente lista se presenta la base de datos que almacena la base de conocimiento del sistema EVCI utilizado para el experimento, es modelado en forma de reglas:

| | |
|---|---|
| Regla 0: | *Si* Error = Subíndice fuera de límite<br>*And* Error = Comisión<br>*And* DxE = Sobrecarga de memoria<br>*Entonces* Tipo de error = Inicialización. |
| Regla 1: | *Si* Error = Caso imprevisto<br>*And* Error = Comisión<br>*And* DxE = Diseño lógico<br>*Entonces* Tipo de error = Cómputo. |
| Regla 2: | *Si* Error = Error en el algoritmo<br>*and* Error = Comisión<br>*and* DxE = Diseño lógico<br>*Entonces* Tipo de error = Cómputo. |
| Regla 3: | *Si* Error = Liberación de memoria<br>*And* Error = Omisión<br>*And* DxE = Sobrecarga de memoria<br>*Entonces* Tipo de error = Cómputo. |
| Regla 4: | *Si* Error = Análisis inadecuado<br>*And* Error = Comisión<br>*And* DxE = Léxico<br>*Entonces* Tipo de error = Cómputo. |
| Regla 5: | *Si* Error = Interpretaciones erróneas en la semántica del lenguaje<br>*And* DxE = Lenguaje<br>*Entonces* Tipo de error = Cómputo. |

| | |
|---|---|
| Regla 6: | *Si* Error = Terminación abrupta<br>*And* DxE = Diseño lógico<br>*Entonces* Tipo de error = Control. |
| Regla 7: | *Si* Error = Validación de memoria<br>*And* Error = Omisión<br>*And* DxE = Sobrecarga de memoria<br>*Entonces* Tipo de error = Control. |
| Regla 8: | *Si* Error = Resultados incorrectos<br>*And* Error = Comisión<br>*And* DxE = Variable<br>*Entonces* Tipo de error =Datos. |
| Regla 9: | *Si* Error = Sintaxis ambigua<br>*And* Error = Comisión<br>*And* DxE = Léxico<br>*Entonces* Tipo de error = Control. |
| Regla 10: | *Si* Error = Terminación abrupta<br>*And* DxE = Sobrecarga de memoria<br>*Entonces* Tipo de error = Interface. |
| Regla 11: | *Si* Error = Omisión<br>*And* DxE = Diseño lógico<br>*Entonces* Tipo de error = Cosmético. |

La razón de usar reglas de producción para la representación del conocimiento en esta investigación, es la facilidad para entender y agregar nuevas reglas a la base de conocimiento sin afectar las reglas que ya existan [13].

## 5. Resultados

Como resulta de esta investigación, se presenta uno de los componentes más significativos del Sistema EVCI, éste es el Sistema Experto. En la Figura 3 se visualiza una página de este componente, en esta página el profesor puede ver tres listas.

La primera corresponde a la lista de los tipos de defectos (errores). Cabe mencionar que este EVCI se ha utilizado para realizar estudios con experimentos controlados en Ingeniería de Software [18, 19], para los experimentos se escogió los seis tipos defectos comunes. La segunda lista corresponde a los defectos (errores) de cada una de los tipos de defectos y la tercera lista muestra los diagnósticos de error asociados a cada tipo de defecto.

Para ambos casos, tanto para la lista de tipos, la lista de defectos como para la de diagnósticos de defectos, en el menú inferior de la aplicación se muestran los botones correspondientes para agregar un registro nuevo a la lista correspondiente, también para cada registro es posible seleccionar y eliminar el dato correspondiente. Las opciones de agregar o eliminar datos solo son posibles con el rol de administrador o tutor, donde

este último es el experto que puede ser el profesor, un alumno solo puede realizar operaciones de consulta e inferencia.

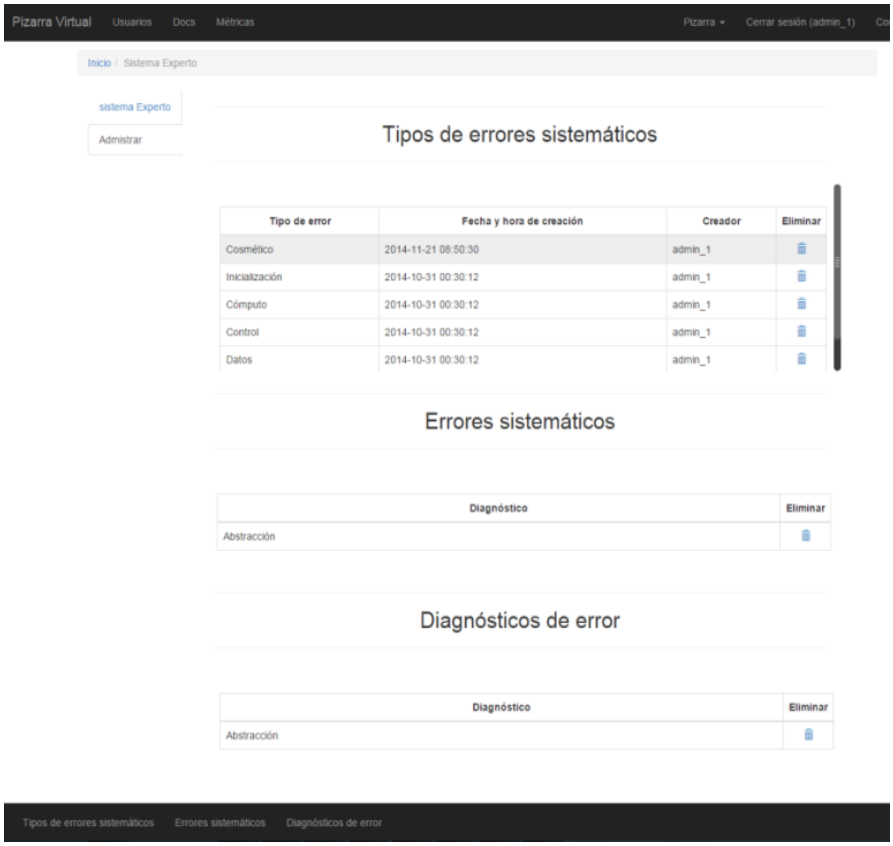


**Fig. 3.** Página para la edición de base del conocimiento.

Al pulsar sobre el botón de errores sistemáticos sobre la misma página, se muestra un diálogo que se ilustra en la Figura 4.

En este diálogo el profesor puede seleccionar de la lista defectos (errores sistemáticos) el elemento que corresponda agregar, como parte del tipo de error que haya seleccionado previamente. Dado que esta lista es larga se ha puesto una caja de texto en donde los profesores escriben una palabra que el sistema utilizará como criterio de búsqueda y entonces reducirá la lista de defectos a solamente aquellos términos que coincidan con el criterio de búsqueda.

Para el caso de los diagnósticos de defectos el sistema funciona de la misma manera. Con el botón Diagnósticos de error el sistema permite a los profesores agregar nuevas pruebas diagnósticas a la lista y relacionarlas con el tipo de errores que se esté modelando. Al pulsar sobre el Diagnósticos de error aparecerá el diálogo que se muestra en la Figura 5 con un listado de todos los diagnósticos de defectos que se encuentran en la base de conocimientos original.

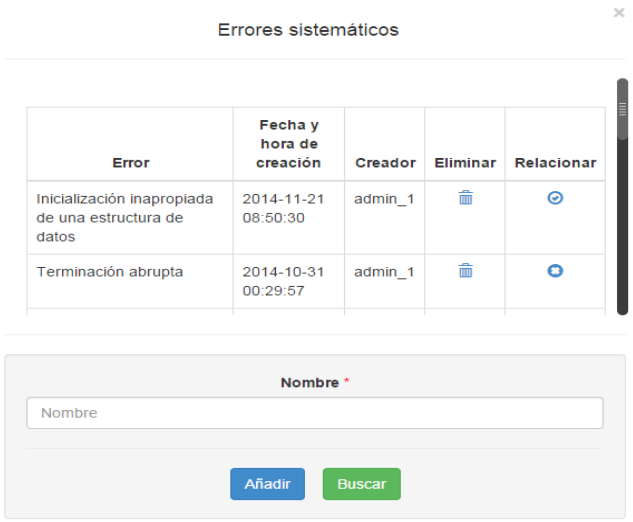


**Fig. 4.** Diálogo para agregar nuevos errores.

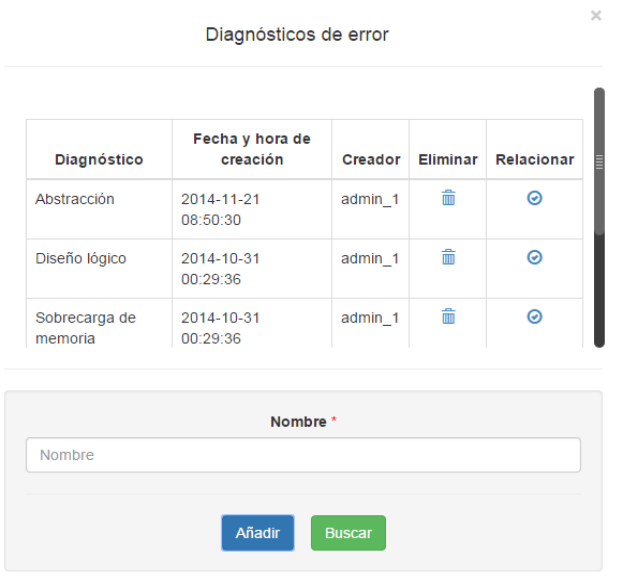


**Fig. 5.** Diálogo para agregar diagnósticos de defectos.

Dentro de este mismo módulo del Sistema EVCI, además de la opción que corresponde a la edición de la base de conocimiento, explicado anteriormente, también tiene una segunda opción que corresponde a la obtención de los tipos de defectos comunes de la programación, aquí tienen permisos tanto el alumno como el profesor y en su caso el administrador.

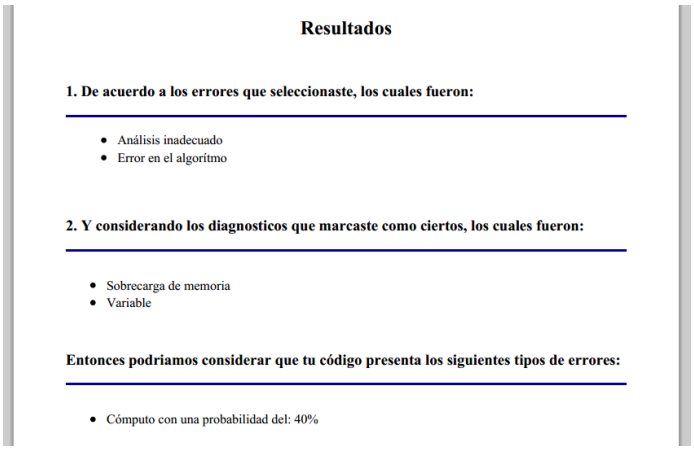


**Fig. 6.** Resultado con lista de los tipos de defectos diferenciales.

Después de haber agregado elementos a dichas listas, el contenido de estas listas se envía al motor de inferencia quien a su vez hace la búsqueda en la base de conocimiento para, finalmente, mostrar la lista de los tipos de defectos diferenciales que se generen como respuesta. Un ejemplo de esto se muestra en la Figura 6.

Como se puede observar, los resultados se imprimen con porcentajes debido a que en el método basado en reglas se consideró un sistema probabilístico y éste se calcula con base al número de atributos y diagnósticos de defectos que tiene un tipo de defecto.

## 6. Trabajos en curso y futuros

De acuerdo con las actividades descritas en la sección 3, el proceso de experimentación con el Entorno Virtual se ha comenzado a realizar en diversos escenarios, tal y como se ha reportado en [18], no obstante, el componente Inteligente (Base de Conocimientos) aún no ha sido validado; al momento de redactar este artículo el grupo de investigación se encuentra diseñando un escenario de aprendizaje que permita validar el uso del Sistema Experto.

Entre las conclusiones parciales de los experimentos realizados, se ha resuelto la conveniencia de experimentar con otros lenguajes de programación; en el diseño original la Base de Conocimientos se utilizó como referente al paradigma estructurado, en específico se utilizó el Lenguaje C; actualmente se está experimentando con otro lenguaje de programación, éste pertenece al paradigma de programación orientada a objetos, se utilizó Java, resultado parcial reportado en [19]. Otro trabajo interesante a futuro, sería adecuar el EVCI para que aprendiera en forma autónoma directamente de la experiencia del estudiante, para convertirla en una herramienta computacional mucho más flexible, sin dependencia del experto o del tutor.

Finalmente, una actividad obligada y que será realizada a mediano plazo, consiste en medir el desempeño del sistema experto del EVCI; lo cual puede ser realizado comparando el sistema propuesto con otros sistemas o con un estándar, y para ello se requerirá un conjunto de expertos.

## Referencias


1. Ucán, J.: Aprendizaje de la Programación Asistido con Entornos Virtuales Colaborativos Inteligentes. Tesis Doctoral, Universidad del Sur (2015)
2. Pennington, N., Grabowski, B.: The tasks of programming. Hoc, J.-M., Green, T. R. G., Samurcay, R., Gilmore, D.J. (Eds), Psychology of programming, Academic Press, UK (1990)
3. Jablonowski, J.: Some Remarks on Teaching of Programming, Proceedings of International Conference on Computer Systems and Technologies (2004)
4. Kaasbøll, J.: Exploring didactic models for programming. Norsk Informatikk-Konferanse, Høgskolen I Agder (1998)
5. Rogalski, J., Samurcay, R.: Adquisition of programming knowledge and skills. Hoc, J.-M., Green, T. R. G., Samurcay, R., Gilmore, D.J. (Eds), Psychology of programming, Academic Press, UK (1990)
6. Vizcaino, A.: Enhancing Collaborative Learning Using a Simulated Agent. Tesis Doctoral, Universidad de Castilla-La Mancha, España (2002)
7. Guzdial, M.: Programming environments for novices. Fincher, S., Petre, M. (Eds.), Computer Science Education Research (pp. 127–154), Lisse, The Netherlands: Taylor & Francis, pp. 127–154 (1994)
8. Kelleher, C., Pausch, R.: Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, Vol. 37, No. 2, pp. 88–137 (2005)
9. IEEE, IEEE Standard Glossary of Software Engineering Terminology. Std. 610.12-1990. Institute of Electrical and Electronics Engineers, NY, USA (1990)
10. Mayrhauser, A., Vans, A.: Program Understanding Behavior During Debugging of Large Scale Software, Empirical Studies of Programmers, 7th Workshop, Alexandria, VA pp. 157–179(1997)
11. Ko, A.: Asking and Answering Questions about the causes of Software Behavior. PhD Thesis, University of British Columbia (2008)
12. Giarratano, J., Riley, G.: Expert systems: principles and programming. Canada: Thomson Technology (2005)
13. Turban, E., Aronson, J.E., Liang, T.P., Sharda, R.: Decision Support System and Business Intelligence. Eighth Edition, Edit Pearson Education, Inc. (2012)
14. Ignizio, J.: Introduction to expert systems. McGraw-Hill (1991)
15. Hernández, G.: Análisis del uso de la inteligencia colaborativa como herramienta para la construcción de bases de conocimiento consensuadas en procesos de diagnóstico médico. Tesis Doctoral, Universidad Carlos III de Madrid (2013)
16. Holt, P., Dubs, S., Jones, M., Greer, J.: The state of Student Modelling. Student Modelling: The Key to Individualized Knowledge-Based Instruction, Springer-Verlag, pp. 3–39, (1994)
17. Basili, V., Perricone, B.: Software errors and complexity: an empirical investigation. Communications of the ACM, Vol. 27, No. 1, pp. 42–52 (1984)
18. Ucán, J., Gómez, O., Castillo, A., Aguilar, A.: Detección de defectos con y sin apoyo de un entorno virtual colaborativo inteligente en cursos introductorios de programación. Gómez, O.S., Arcos, G., Aguirre, L., Villa, E., Rosero. R.H. (eds), Ingeniería de Software e Ingeniería del Conocimiento. Jornadas Iberoamericanas. 11th (JIISIC'2015). Curran Associates, Inc., Riobamba, Elsevier, pp. 65– 78 (2015)
19. Ucán, J., Aguilar, R., Aguileta, A., Díaz, J.: Analizando la efectividad del uso de un EVCI para asistir a estudiantes avanzados en la identificación de faltas en el código: un experimento controlado. Revista Latinoamericana de Ingeniería de Software, Vol. 4, No. 1 (2016)