ISBN: 978-959-7213-02-4

ENSEÑANZA DE C++ AL ESTUDIANTE JAVA
TEACHING C++ TO THE JAVA STUDENT

Adolfo Di Mare

Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica

adolfo.dimare@ecci.ucr.ac.cr

RESUMEN: *Se describe como minimizar el impacto de pasar al uso del lenguaje C++ si los estudiantes han aprendido primero el lenguaje Java. También se discute cómo organizar el segundo cursos de programación.*

Palabras Clave: Transición Java C++, Programación, Abstracción, Especificación.

ABSTRACT: *We describe how to minimize the impact of changing to using C++ if students have learned the Java language first. We also discuss how to organize the second programming courses.*

KeyWords: C++ to Java transition, Programming Abstraction, Specification.

1. INTRODUCCIÓN

La escogencia de los lenguajes de programación usados en las carreras de computación frecuentemente es controversial. Sin embargo, cuando ya se ha decidido que se usará primero Java y luego C++, es provechoso contar con un enfoque didáctica que permita hacer la transición al nuevo lenguaje pues, de lo contrario, hay que usar demasiado tiempo explicando los detalles sintácticos de C++. Las ideas aquí expuestas sirven para mitigar el impacto del cambio de lenguaje.

2. ESPECIFICACIÓN Y REUTILIZACIÓN

En algunas carreras los cursos de programación se limitan a entrenar a los estudiantes en el uso de un lenguaje. Por ejemplo, si se ha decidido que los alumnos deben aprender Python, Java, C++ y PHP la estrategia es usar un curso para cada lenguaje. Esto aumenta el menú de lenguajes pero requiere muchos cursos, y siempre existe el peligro de que algunos conceptos teóricos de programación no sean cubiertos.

En la Universidad de Costa Rica [UCR] hemos optado por una estrategia diferente, que además usa menos cursos: enseñamos los principios de programación en el primer curso y para el segundo nos concentramos en la especificación y reutilización de módulos y programas. Posteriormente en la carrera hay otros cursos especializados en diferentes paradigmas de programación, incluyendo programación lógica, programación concurrente, programación internet, etc. Muchas veces los estudiantes aprenden lenguajes solos, sobre la marcha, cuando tienen que producir soluciones para tareas o proyectos en cursos posteriores.

Primero por tradición, y luego por escogencia, en la UCR hemos usado en el segundo curso de programación el libro de Abstracción y Especificación escrito por Liskov y Gutag [1] (cuya primera edición obtuvimos, paradójicamente, cuando uno de nuestros profesores lo rescató del basurero en una visita que hizo a otra universidad). Aunque este libro está escrito para Java, como fundamente teórico se puede usar también para C++ si se adopta nuestra estrategia de usar el segundo curso de programación para que los alumnos comprendan cómo especificar módulos para reutilizarlos. Estas 3 palabras son fundamentales en la construcción de programas y por eso es necesario definirlas [2]:

Módulo

Sección de un programa bien construida, con un fin específico, y que puede ser reutilizado.

Especificación

La especificación puede verse como un contrato en el que están definidos todos los servicios que la implementación del módulo es capaz de dar. Toda especificación debe tener estas tres cualidades: debe ser completa (debe decirse todo lo que hay que decir), debe ser correcta (debe omitirse lo que no hay que decir) y no puede ser ambigua.

Reutilización

Reutilizar significa no inventar de nuevo la rueda, aprovechando lo que otros hicieron para construir nuevas soluciones [3].

Como la reutilización generalmente es "reutilización de algoritmos" casi siempre es necesario usar ocultamiento de datos al construir programas, para evitar que la representación interna de cada objeto pueda ser cambiada por un programador que usa una biblioteca de clases y algoritmos. Es de gran importancia que los estudiantes conozcan y apliquen este principio de construcción de sistemas [4].

Aunque sí es importante que el graduado universitario desarrolle las destrezas necesarias para aplicar tecnología, también debe ser formado para que pueda formar parte de esa minoría selecta o rectora que dirige a otros, pues una de las misiones primordiales de una universidad es formar a la élite directora: no basta formar albañiles porque también se necesitan ingenieros y arquitectos.

El primer curso de programación puede verse como un curso de albañilería, en que el futuro graduado aprende los fundamentos de computación. El siguiente paso es el aprendizaje de las teorías y técnicas necesarias para liderar equipos de programación: es necesario que el futuro graduado aprende a especificar módulos para que pueda definir el trabajo que otros realizarán.

Siempre es necesario encontrar maneras de cumplir con los objetivos de enseñanza – aprendizaje con mayor eficiencia y eficacia: hacer lo mismo con menos recursos es una constante en el avance científico y tecnológico. Desde esta perspectiva, la escogencia del lenguaje para Programación 2 debe hacerse en función de las ventajas didácticas respecto a los objetivos del curso. En este contexto, C++ se destaca sobre los demás lenguajes pues, además de asegurar que el futuro graduado conocerá los lenguajes más populares, también conocerá un lenguaje que incluye la mayor parte de los avances tecnológicos en programación.

3. C++ PARA PROGRAMACIÓN 2

C++ incluye un fuerte soporte para parametrización, herencia y polimorfismo, que son los vehículos principales para lograr la reutilización de algoritmos. Sin reutilización no hay progreso, pues sería necesario reescribir desde cero todas las aplicaciones, lo que transformaría en prohibitivo el desarrollo de programas. Ya están disponibles muchas bibliotecas C++ eficientes que se pueden usar para desarrollar aplicaciones prácticamente en cualquier contexto [5].

Debido a que C++ surgió a partir de C, que es un lenguaje que algunos consideramos arcaico [6], contiene muchas complicaciones que no existen en otros lenguajes. Por eso, en la práctica muchos proyectos se efectúan usando lenguajes de guiones de comandos (*scripting languages*) que permiten expresar algoritmos de manera sucinta y muy directa. En muchos contextos, el desarrollo C++ es más caro, por lo que es fácil argumentar que es mejor que el futuro graduado en computación aprenda otros lenguajes que se usan en el mercado para implementar aplicaciones (por ejemplo: PHP, Visual Basic, C#, Java, Phyton, Ruby). Sin embargo, una persona que conoce C++ puede asimilar cualquier otro lenguaje con relativa facilidad, mientras que lo contrario no es cierto. La razón principal para usar C++ no es su popularidad sino su cobertura: C++ incorpora la mayor parte de los avances en programación, sin sacrificar eficiencia [7]. Por eso en los ambientes universitarios se le usa como componente didáctico en muchos de los cursos.

Otra de las grandes ventajas de los lenguajes gordos (como C++ o Java), en contraposición a los simples y directos (*lean and mean*: PHP, Ruby) es que sus compiladores proveen el servicio de verificación de tipos (*type checking*). Este técnica no es nueva, pues no había terminada la década de los años sesenta cuando ya se conocía (pues Pascal y su predecesor Algol son lenguajes cuyos compiladores ofrecían esos servicios a principio de 1970). Si el compilador verifica concordancia de tipos se encarga de encontrar errores de interfaz entre módulos cuando argumentos y parámetros no calzan: así se logra que la máquina haga trabajo que de otra forma debería ser realizado por los programadores [8], lo que los hace más productivos. Si los programadores son más productivos el costo total del proyecto disminuye mucho, pues el costo de la planilla es uno de los rubros más gruesos en cualquier proyecto. La tecnología de programación busca siempre reducir el costo aumentando la productividad de todos los programadores.

Existen muchas herramientas que complementan un ambiente de desarrollo C++. Estas destacan sobre las demás:

Doxygen

Permite extraer anotaciones del código fuente para producir la especificación de módulos e interfaces. Tiene la ventaja de que funciona tanto para C++ como para sus lenguajes derivados: Java, C#, Objective-C, etc.

`http://Doxygen.org`

Code::Blocks

Ambiente de programación para C++ multiplataforma que puede usar múltiples compiladores, en especial los de GNU. Existen versiones compactas que pueden ejecutarse desde una memoria USB en máquinas Linux, Windows o MAC.

`http://CodeBlocks.org`

Boost

Biblioteca portable de algoritmos C++.

http://Boost.org

FLTK

Biblioteca para escribir interfaces gráficas GUI (*Graphic User Interface*) para programas C++.

http://FLTK.org

SQLite

Implementación SQL completa y portátil que permite incorporar como una subrutina el motor de base de datos a programas C o C++.

http://SQLite.org

4. CONVENCIONES DE PROGRAMACIÓN

Es importante que los estudiantes comprendan que no basta que los algoritmos sean efectivos pues es necesario que el código fuente de los programas sea legible, para así se logra disminuir un poco el costo de darle mantenimiento al código. Por eso, el uso de convenciones de programación debe ser uno de las destrezas quienquiera que estudie computación [9].

Muchas veces la reticencia de adoptar convenciones nace de la rebeldía natural de los jóvenes que los lleva a rechazar reglas que han sido definidas por otros. Sin embargo, la disciplina de utilizar convenciones ayuda a mejorar la calidad del programa y luego le permite al estudiante desarrollar su propio estilo o adoptar el de su empleador.

Existen varias herramientas que sirven para darle formato al programa usando plantillas que sigan algún patrón o convención. La herramienta Uncrustify [10] es muy potente y completa, aunque puede resultar más cómodo usar AStyle porque funciona como un programa adherido al ambiente de programación Code::Blocks [11].

Convenciones de Programación
• Especificación correcta, completa y no ambigua.
• Correcta indentación del código fuente.
• Correcto espaciado del código fuente.
• Código fuente escrito de manera legible y clara.
• Uso de identificadores significativos.

Figura 1

El autor ha experimentado con el uso de muchos tipos de convenciones de programación, desde usar cientos de reglas hasta examinar con programas verificadores de convenciones el código implementado por los estudiantes, pero a fin de cuentas lo que ha resultado más eficaz es usar reglas que sean muy simples de recordar y aplicar, como las que se muestran en la Figura 1.

5. TRANSICIÓN DE JAVA A C++

C++ es un lenguaje más complicado que Java y por eso, para cualquier docente, la tentación de utilizar muchas lecciones explicando la sintaxis del lenguaje es difícil de vencer.

```
#ifndef Acumulador_h
#define Acumulador_h ///< Evita inclusión múltiple
class Acumulador{private:long m_total;long m_cantidad;
long m_menor;long m_mayor;public:Acumulador():m_total(
0),m_cantidad(0),m_menor(0),m_mayor(0){/*borre();*/}~
Acumulador(){}Acumulador(const Acumulador&o){*this=o;
}const Acumulador& operator=(const Acumulador&o){
m_total=o.m_total;m_cantidad=o.m_cantidad;m_menor=o.
m_menor;m_mayor=o.m_mayor;return*this;}void borre()
{m_total = m_cantidad = m_menor = m_mayor = 0;}long
total()const{return((m_cantidad <= 2) ? 0 : m_total-
m_menor-m_mayor);}long cantidad() const{return
m_cantidad;}void acumule(long val){m_total += val;
m_cantidad++;if(m_cantidad > 1){ m_menor =( val <
m_menor ? val : m_menor);m_mayor =( val > m_mayor?val:
m_mayor);}else{m_menor = m_mayor = val;}}void acumule(
unsigned n, const long * val){for(unsigned i=0; i<n;++
i){acumule(val[i]);}}friend bool operator==(const
Acumulador&A,const Acumulador&B){return(A.m_total==B.
m_total&&A.m_cantidad==B.m_cantidad&&A.m_menor==B
.m_menor&&A.m_mayor==B.m_mayor);}friend bool operator!=
(const Acumulador&A,const Acumulador&B){return!(A==B);}
friend bool check_ok(const Acumulador&A);friend class
test_Acumulador;}; /* bool */
#endif
```

Figura 2

En la Figura 2 se muestra una forma de evitar usar mucho tiempo para explicar cómo es un programa C++: el trabajo que tiene que hacer cada estudiante es darle formato a la clase Acumulador, que sirve para sumar un grupo de valores numéricos, pero eliminando del total tanto al más grande como al más pequeño de todos los valores. Por ejemplo, el acumulador aplicado a los valores (1,5,2,2,3,3,4,5) sólo sumaría 5+2+2+3+3+4 y eliminaría el 1 y alguno de los 5's, que son los valores menor y mayor de toda la secuencia. La descripción completa de este proyecto programado está aquí:

http://www.di-mare.com/adolfo/cursos/2010-2/p2-ta-1.htm

El truco pedagógico consiste en que el profesor le entregue a sus alumnos el resultado de incorporar las anotaciones Doxygen a la implementación de esta clase cuando les muestra cómo resolver esta asignación. Al cabo de una semana, ya los muchachos conocerán muchos de los detalles del lenguaje, lo que les permitirá enfrascarse en la construcción de programas más complejos a partir de la segunda asignación. El uso del embellecedor de

código AStyle aminora mucho el trabajo que hay que realizar.

Uno de los inconvenientes de mostrar programas completos es que los muchachos muchas veces sienten que el trabajo es enorme, abrumador. Por eso es necesario crear soluciones pequeñas pero completas en las que los estudiantes perciban el proceso constructivo desde el principio: por ejemplo, ayuda implementar un programa completo que lea números y los ordene en un vector.

6. BIBLIOTECAS DE PROGRAMAS

Además de la especificación de módulos reutilizables, en el segundo curso de programación hay que mostrar cómo usar bibliotecas, pues quien no las haya usado tampoco puede construirlas. Para C++ hay 2 bibliotecas muy importantes: primero está la biblioteca estándar, que ofrece muchos algoritmos y contenedores y apoya la construcción de programas concurrentes [12]. Además, la biblioteca Boost incorpora algoritmos aún para aplicaciones de avanzada [13].

Gracias a la potente forma de parametrización que incorpora C++ es posible reutilizar muchos algoritmos y, en algunos casos, resulta más potente que la de otros lenguajes. Por ejemplo, al comparar la parametrización Java con la de C++ se puede notar que, debido que el programador Java siempre debe usar objetos, las plantillas Java se usan más que todo para parametrizar contenedores y no para parametrizar algoritmos. Resulta que a pesar de que la biblioteca Java ya cuenta con el contenedor **`List<>`** pero no cuenta con un algoritmo genérico **`sort<>`** que permita ordenar cualquier contenedor. Para solventar esta carencia, los contenedores Java incluyen un método de ordenamiento, lo que aumenta la cohesión entre el contenedor y el algoritmo de ordenamiento (aunque es cierto que en la mayor parte de las aplicaciones cualquier algoritmo de ordenamiento es adecuado, más si se toma en cuenta que ordenar 10 millones de números en promedio toma menos de 5 segundos en un computador personal, aún si para el caso peor dura varias horas).

7. PRUEBA DE PROGRAMAS

La prueba de programas es un campo del conocimiento muy amplio y merece un tratamiento exhaustivo en uno o más cursos universitarios [14]. Sin embargo, es fácil hacer prueba unitaria de módulos usando como base el verbo **`assert-True()`** (o cualquier otra variante del mismo concepto), que está disponible en muchas plataformas.

```
/// Prueba unitaria de programas.
#define assertTrue(cond) if (!(cond))     \
{ std::cout << "error [" << #cond << "] " \
            <<   LINE    << std::endl; }
```

Figura 3

En la Figura 3 se muestra una implementación mínima de **`assertTrue()`**, que el profesor puede usar para mostrar cómo incorporarlas para complementar las especificaciones de programas desde el principio, cuando trabajan en la clase Acumulador [2].

```
Acumulador::Acumulador() [inline]
Constructor de vector.
{{  // test::constructor()
    Acumulador A;
    assertTrue( 0 == A.total() );
    assertTrue( 0 == A.cantidad() );
    long vec[] = { 1, 2, 3, 4,  5 };
    A.acumule( DIM(vec) , vec );
    assertTrue( 2+3+4 == A.total() );
    A.borre();
    assertTrue( 0 == A.total() );
}}
```

Figura 4

En la Figura 4 se muestra un ejemplo del uso de datos de prueba para complementar la especificación de un módulo. Para muchos programadores los ejemplo de uso como este son mucho más convincentes y apropiados que una documentación exhaustiva, razón por la que es importante que los estudiantes los usen pronto, cuando producen sus especificaciones de módulos [2].

8. EL CURSO PROGRAMACIÓN 2

El temario del segundo curso debe ser una continuación natural del primero. Si se aceptan las ideas aquí compartidas es posible definir con más precisión el contenido del curso.

Programación 2	
SDF/Algorithms and Design	11
SDF/Development Methods	10
SE/Software Verification Validation	3
Horas totales	24

Figura 5

Una estrategia adecuada para definir cuáles temas debe dominar quien terminó Programación 1 es usar la taxonomía del conocimiento publicada la *Association for Computing Machinery* [ACM] cuya versión preliminar 2013 ya está disponible [15]. En la Figura 5 se encuentra el temario para el curso, que implica 24 horas de instrucción presencial. Un curso usualmente tiene hasta 60 horas lectivas, por lo que es posible extender este temario, incorporando algunas otras áreas del conocimiento:

- PD/Parallel Decomposition
- PD/Parallelism Fundamentals
- PBD/Mobile Platforms

Sin embargo, agregar muchos contenidos puede ser contraproducente pues los alumnos universitarios necesitan más de un año de tiempo para aprender a estudiar, lo que en la práctica limita la cantidad de materia que pueden asimilar. También hay que dar espacio para que los muchachos puedan asimilar C++ pues, independientemente de los trucos que uno use para acelerar la velocidad del aprendizaje, siempre es necesario darles tiempo para que maduren sus conocimientos.

En algunos proyectos programados el autor ha usado SQLite [16] como herramienta complementaria, pues es saludable introducir pronto tópicos de bases de datos. Sin embargo, usar varias tablas en el proyecto tiene como resultado que el mismo se complica y alarga. Si el estilo pedagógico del profesor es usar proyectos grandes, las bases de datos complicadas pueden ser la mejor opción.

9. JAVA ES UNA MEJOR ELECCIÓN

El enfoque de la carrera de computación determina el tipo de lenguaje que se imparte en los cursos. Aquellas carreras que tienen una formación que está más influenciada por los cimientos históricos de la computación, en donde la programación en lenguaje de máquina y el estudio matemático de los algoritmos era fundamental, posiblemente incluyan a C++ como su lenguaje predilecto porque le da un mayor control al programador. Algunas de las cualidades que hacen de C++ un lenguaje preferido en la academia son estas:

- Plantillas o programación genérica incorporada en el lenguaje.
- Programación “pegada al hierro”.
- Apoyo para la programación orientada a objetos en la forma de parametrización y polimorfismo.
- Disponibilidad adecuada de compiladores y ambientes de compilación.
- Apoyo para separar la abstracción de cada módulo de su implementación efectiva.
- Disponibilidad de herramientas para la generación de documentación.

Las carreras de computación que están más inclinadas a la construcción de soluciones computacionales para empresas o instituciones, posiblemente usen el lenguaje Java como su caballo de trabajo. Por supuesto, ambos lenguajes son adecuados, y también es posible mostrar sus defectos para justificar el sesgo de una persona o de una academia a favor de alguno de ellos.

La polémica Java vs C++ es interesante, pero a fin de cuentas la decisión que se tome depende mucho del contexto en que deberán desenvolverse los graduados del programa académico.

1	C	19.9%
2	Java	17.2%
3	Objective-C	9.5%
4	C++	9.3%
5	C#	6.5%

Figura 6

La popularidad de un lenguaje a veces es un criterio relevante para incluirlo en el temario de un curso. En el caso de C++, su popularidad es relativamente pequeña según el índice TIOBE [17]. Sin embargo, si se considera a C y Objective-C como versiones reducidas de C++ y a C# como un dialecto de Java, quien primero aprenda Java y luego practique C++ podrá trabajar en más del 60% de los proyectos de construcción de sistemas. Aunque este criterio no es determinante, sí puede ser importante para hacer la transición de Java a C++ en un carrera universitaria (posiblemente la popularidad de Objective-C se debe a su uso para implementar soluciones iPhone; Android usa Java).

Java funciona con apoyo de su recolector de basura que le quita al programador la responsabilidad de retornar la memoria dinámica que usa. Si los programas C++ son implementados empacando en clases aquellos objetos que requieren memoria dinámica, basta que los destructores estén correctamente implementados para que los programas funcionen correctamente. También es posible usar punteros inteligentes para mostrar cómo se pueden transliterar de Java a C++ algunos programas [18].

10. CONCLUSIONES

Al usar programas completos como el que incluye el módulo Acumulador de la Figura 2, un profesor puede evitar el engorroso trabajo de concentrarse en la sintaxis de C++ si sus alumnos aprendieron Java primero. En este artículo se presentan algunas ideas para lograr esta transición, en el contexto de lograr que los estudiantes adquieran pronto los conceptos mínimos para producir programas de calidad. Por eso, aquí se sugiere que en la carrera de computación se incorpore la especificación y reutilización de módulos, incluyendo el uso de ejemplos de prueba unitaria para complementar la documentación.

En el segundo curso de programación los estudiantes pueden trabajar en equipos. El objetivo de Programación 2 se puede definir así: “Cada estudiante debe aprender a especificar módulos para reutili-

zarlos, usando prueba unitaria de programas para completar las especificaciones".

11. AGRADECIMIENTOS

Alejandro Di Mare aportó varias sugerencias para exponer las ideas expresadas en este artículo. Además, varios docentes de la Escuela de Ciencias de la Computación e Informática también hicieron observaciones relevantes.

12. REFERENCIAS BIBLIOGRÁFICAS

1. Liskov, Barbara & Guttag, John: Program Development in Java: Abstraction, Specification, and Object-Oriented Design, Addison-Wesley Professional, 2000.

2. Di Mare, Adolfo: Especificación de módulos con ejemplos y casos de prueba, Artículo CAL002 del V Taller de Calidad en las Tecnologías de la Información y las Comunicaciones (TCTIC-2011), realizado del 7 al 11 de febrero de 2011 en el Palacio de Convenciones de la Habana, en la Habana, Cuba.

http://www.di-mare.com/adolfo/p/BUnitXP.htm

3. Krueger, Charles W.: Software Reuse, ACM Computing Surveys, Vol.24 No.2, pp 131-183, Junio 1992.

http://www.biglever.com/papers/Krueger_AcmReuseSurvey.pdf

4. Di Mare, Adolfo: ¡No se le meta al *Rep*!, Reporte Técnico ECCI-2007-01, Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, 2007.

http://www.di-mare.com/adolfo/p/Rep.htm

5. Boost.: "Boost libraries for C++", 2012.

http://www.boost.org/.

6. Bjarne Stroustrup: Why the Programming Language C Is Obsolete, 2011.

http://youtube.com/watch?v=KlPC3O1DVcg

7. Stroustrup, Bjarne: Why C++ is not just an Object-Oriented Programming Language, OOPSLA 1995.

http://www2.research.att.com/~bs/oopsla.pdf

8. Prechelt, Lutz & Tichy Walter F.: A Controlled Experiment to Assess the Benefits of Procedure Argument Type Checking, IEEE Transactions on Software Engineering 1998.

9. Weinberger, Benjy & Silverstein, Craig & Eitzmann, Gregory & Mentovai, Mark & Landray, Tashana: Google C++ Style Guide, Revision 3.199, 2012.

http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

10. Uncrustify: Source Code Beautifier for C, C++, C#, ObjectiveC, D, Java, Pawn and VALA.

http://uncrustify.sourceforge.net/

11. Artistic Style: A Free, Fast and Small Automatic Formatter for C, C++, C#, and Java Source Code.

http://astyle.sourceforge.net/

12. Stroustrup, Bjarne: What is C++0x? 2009.

https://www2.research.att.com/~bs/what-is-2009.pdf

13. Boost: Boost libraries for C++, 2012.

http://www.boost.org/

14. Myers, Glenford J.: Three- The Art of Software Testing 2nd Ed, John Wiley & Sons, Inc.,2009.

15. Association for Computing Machinery: Computer Science Curricula 2013 Strawman Draft (February 2012), 2012.

http://ai.stanford.edu/users/sahami/CS2013/strawman-draft/cs2013-strawman.pdf

16. SQLite: A software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine, 2012.

http://SQLite.org/

17. TIOBE: Programming Community Index, 2012.

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

18. Di Mare, Adolfo: lkptr.h: Programación por Referencia Java para C++, XVI Congreso Iberoamericano de Educación Superior en Computación (CIESC-2008), ISBN 978-950-9770-02-7, pp163-172, setiembre 2008.

http://www.di-mare.com/adolfo/p/lkptr.htm

ACERCA DEL AUTOR

Adolfo Di Mare: Investigador costarricense en la Escuela de Ciencias de la Computación e Informática [ECCI] de la Universidad de Costa Rica [UCR], en donde ostenta el rango de Profesor Catedrático. Trabaja en las tecnologías de Programación e Internet. También es Catedrático de la Universidad Autónoma de Centro América [UACA]. Obtuvo la Licenciatura en la Universidad de Costa Rica, la Maestría en Ciencias en la Universidad de California, Los Angeles [UCLA], y el Doctorado (Ph.D.) en la Universidad Autónoma de Centro América.