PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO

Estudio del lenguaje de programación Haskell, ventajas y desventajas con respecto a otros lenguajes de programación

Tesis para optar el título de Magíster en Informática con mención en Ciencias de la Computación, presentado por:

Ing. Julita Inca Chiroque

ASESOR: Dr. Maynard Kong Wong

JURADO:

Mg. Juan Miguel Angel Guanira Erazo

Mg. Johan Paul Baldeón Medrano

Lima, 21 de Setiembre del 2012

Comienza haciendo lo que es necesario,
después lo que es posible y de repente
estarás haciendo lo imposible.
- San Francisco de Asís

Para mí la programación es más que
un importante arte que requiere práctica.
Es algo que también incluye una gran
gama de fundamentos para su conocimiento.
- Grace Hopper

No hay lenguaje de programación perfecto.
No hay ni siquiera un mejor lenguaje de programación,
sólo hay lenguajes de programación muy adecuados
o tal vez poco adecuados para fines particulares.
- Hebert Meyer

- A mi madre, Lidia Chiroque Silva, por la entrega y amor a cada una de sus hijas.
- A mi padre, Florentino Inca Heredia, por su ejemplo de superación y responsabilidad.
- A mis hermanas, Marylú y Magaly Inca Chiroque, por su comprensión, amor y apoyo en todo momento.
- A mis seres queridos que ya están descansando en el cielo, mis abuelos, demás familiares y amigos por su recuerdo.
- A los profesores, JPs y alumnos de la sección Informática PUCP, Corrado Daly, Felipe Solari, Miguel Guanira, Claudia Zapata, Johan Baldeon, Walter Segama, Daniel Céspedes, Giancarlo Calderón, Humberto Cano y Nelly Castillo.

ÍNDICE

INTRODUCCIÓN

En el campo de las ciencias de la computación, el mayor desafío es diseñar y desarrollar lenguajes de programación que faciliten las labores del programador en la construcción de softwares eficientes y sostenibles en el tiempo.

Según [Sebesta 10], para que el computador pueda "comprender" lo que el ser humano quiere transmitir, es necesario diseñar métodos que traduzcan las estructuras de las frases y su significado al código de máquina. Se requiere de un exclusivo vocabulario y la aplicación de reglas estrictas para alcanzar el proceso de traducción. En este proceso se debe armonizar tanto, las perspectivas del diseñador, del implementador como del usuario que empleará el lenguaje de programación.

La historia de la computación señala que desde que aparecieron las primeras máquinas de cómputo, se crearon lenguajes de programación enfocados a lo que las máquinas puedan "comprender" y "hacer". Estos lenguajes de programación se denominaron de bajo nivel porque, se podía manipular directamente los registros de datos e instrucciones y los accesos de memoria.

Los lenguajes de programación evolucionaron cuando se tradujeron los símbolos de máquinas en formas lógicas y matemáticas, así como la introducción de palabras reservadas en el vocabulario que ya no sólo eran comprendidas por parte de la máquina, sino también por parte del programador.

El enfoque actual es lograr un lenguaje de programación que se aproxime al lenguaje natural del hombre siguiendo una lógica que permita que la máquina ejecute los requerimientos predefinidos para el desarrollo de un software o sistema de manera rápida y confiable, agregando las nuevas características de los lenguajes en ciclos cortos e incrementales, proporcionando y siguiendo la lógica de negocio con acceso seguro, persistencia de datos, comportamiento transaccional, y otras características avanzadas.

Actualmente, se han documentado más de 2500 lenguajes de programación diferentes y aún se siguen diseñando otros, debido a que las necesidades de desarrollo de software aumentan, cambian, se combinan o mueren.

Uno de los últimos lenguajes de programación creados en el año 2009, es el lenguaje GOlang. GO es un proyecto soportado por la empresa Google Inc., bajo la dirección del Sr. Rob Pike, el cual es normado bajo los paradigmas imperativo y concurrente.

La historia de los lenguajes de programación demuestra que los lenguajes que se crearon a partir de los años 50 y que imperaron por más de dos décadas con gran popularidad, se basaron en los paradigmas imperativo, orientado a objetos (OO) y funcional.

Los lenguajes de programación que se rigen bajo el paradigma imperativo, establecen una secuencia de comandos o sentencias una debajo de otra, donde el programador hace uso de estructuras iterativas como los bucles para trabajar procesos cuantas veces se indiquen, describiendo paso a paso o mediante procedimientos cómo es que el programa se debe comportar o cómo lo debe hacer. Un lenguaje representantivo de este paradigma, es el lenguaje de programación C.

Los lenguajes de programación que se rigen bajo el paradigma orientado a objetos abstraen la forma de cómo el ser humano concibe las entidades en el mundo real y la considera al momento de declarar y manipular de las estructuras de datos, a través de las denominadas clases. Las clases al igual que las entidades en el mundo real, tienen métodos (comportamiento de las clases) y atributos (estado de la clase), que facilitan la programación cuando se presentan características como herencia y polimorfismo[1]. Un lenguaje representantivo de este paradigma, es el lenguaje de programación Java.

Los lenguajes de programación que se rigen bajo el paradigma funcional plantean cosas diferentes. No se dan sentencias o comandos paso a paso, se define qué es lo que hay que hacer y se definen las tareas a través de funciones. Las funciones se pueden reutilizar de manera recursiva a partir de sentencias condicionales, en lugar de hacer uso de los bucles. Un lenguaje representantivo de este paradigma, es el lenguaje de programación LISP, pero desde hace 20 años, un grupo de científicos con grados de doctorado y postdoctorado que se unieron para crear un nuevo lenguaje de programación funcional puro, denominado Haskell.

[Van 04] analiza los principales paradigmas de programación, y se puede apreciar en su estudio que nacen nuevos paradigmas, a partir de los paradigmas imperantes mencionados, y de la mezcla de dos o mas paradigmas de programación existentes.

Los lenguajes de programación multiparadigmas fueron creados con la intención de brindar soluciones a problemas actuales como concurrencia, abstracción y persistencia de datos, entre otros. Un lenguaje de programacion multiparadigma representativo es el derivado de C, C++. Otro lenguaje multiparadigma que resuelve estos problemas es GO.

[1]Polimorfismo: funciones que se definen para realizar un mismo evento, aplicables a diferentes objetos de diferentes tipos, obteniendo diferentes resultados.

[Sebesta 10] indica que lo ideal es que al menos el programador pueda conocer las intenciones con las que fueron creadas cada una de las estructuras de datos en los diferentes lenguajes de programación. Si se decide correctamente el lenguaje o la estructura de datos del lenguaje al momento de programar, la lectura del código será más fluida, se optimizará recursos tanto en la etapa de creación como en la de mantenimiento de un software.

En el presente trabajo de tesis, se compararán los lenguajes de programación representativos de los paradigmas mencionados: C, Java, C++ y GO, con respecto al lenguaje de programación Haskell. Haskell ha sido desarrollado y mantenido por una comunidad que tiene como miembros a estudiosos de posgrado de las más prestigiosas universidades del mundo, con la misión de superar errores y mejorar el diseño de lenguajes de programación convencionales, como por ejemplo, presenta soporte para genéricos, que permite al programador utilizar librerías y reutilizar funciones que se aplican para cualquier tipo de datos; consecuencia de ello es la escritura de un programa simple y conciso. Otra fortaleza de Haskell está reflejada en su sistema de tipos, el cual es fuertemente tipificado, que conlleva a la confiabilidad; y la inferencia de tipos, que permite que un programa se pueda escribir sin declarar los tipos de las variables.

La comparación que se realizará entre los lenguajes de programación se basará en los criterios de lectura, escritura, confiabilidad, reflexión y soporte para programación genérica. La metodología que se aplicará es la revisión bibliográfica, en base a la lectura de publicaciones, investigaciones, tesis de posgrado de Ciencias de la Computación para evaluar a través de los cinco aspectos mencionados, el estudio y avance de los lenguajes C, C++, Java, Haskell y GOlang; y una metodología hipotética-deductiva, se presentarán programas que resuelvan soluciones, aplicando los criterios elegidos en cada uno de los lenguajes de programación y se determinará ventajas o desventajas del lenguaje de programación Haskell sobre C/ C++, Java y GOlang.
En el primer capítulo se expone acerca del impacto de los lenguajes de programación elegidos para el estudio en los ámbitos: académico, industrial y científico. En el segundo capítulo se describirán los lenguajes de programación que fueron apareciendo a través de la historia de la programación y sus paradigmas de programación. En el tercer capítulo se expondrán los lenguajes de programación C, Java, Haskell y GOlang. En el cuarto capítulo se expondrán los criterios de evaluación de un lenguaje de programación. En el quinto capítulo se compararán los criterios determinados de acuerdo a estudios predecesores, en este caso se han definido las características de fácil lectura, fácil escritura, confiabilidad, reflexión y soporte para programación genérica.

CAPÍTULO I: ANTECEDENTES

La aplicabilidad de las computadoras para la solución de problemas a diversas situaciones ha sido demostrada y registrada en publicaciones desde los años 60. Aplicaciones que resuleven desde un juego de ajedrez, composiciones musicales, validaciones matemáticas, entre otros, resueltas utilizando un lenguaje de programación. [Goldfinger 61] realiza una investigación para mejorar el diseño de los lenguajes de programación de la época, aplicada al algebra de información que proporciona una notación para identificación y caracterización de los datos que sean objeto de tratamiento y una técnica para expresar las relaciones que existen entre los datos. Con este estudio se mejoró la notación del lenguaje y desarrolló estructuras de alto nivel y se definieron macroinstrucciones para proveer un conjunto de verbos para incluirlos en los lenguajes estructurados.

Pese a los esfuerzos realizados en la época; un estudio en 1979 [Floyd 07] acerca de los paradigmas de programación, cita en su estado del arte un estudio de Balzer, quien indica: "Es bien sabido que la industria del software se encuentra en un estado de depresión. No hay confiabilidad, los software no son entregado a tiempo y no son adaptables o modificables a cambios, sin que ello implique altos costos en el desarrollo". Robert Floyd, recuerda como anécdota en sus escritos el día que ingresó al campo de la computación en 1956 y vio en las paredes de la oficina de graduados de la Universidad de Stanford "Prefiero escribir programas que me ayuden a escribir programas, que escribir programas". Cuando investigó las causas de esta problemática, afirmó que, el avance continuo de la programación como un arte, requiere del desarrollo y la difusión de los lenguajes que soportan los grandes paradigmas en las comunidades de usuarios.
Mezcló paradigmas para desarrollar nuevos lenguajes de programación, naciendo el concepto de “multiparadigma”. Este concepto se consolidó conforme se crearon nuevos lenguajes de programación a largo de la historia de la computación.
Los lenguajes de programación han ido evolucionando, superando errores de programación y cubriendo nuevas necesidades de programación. Año tras año, se han desarrollado tecnologías que aseguren un buen funcionamiento y desempeño con miras de estabilidad en el mantenimiento y sostenibilidad en el tiempo. Cada escuela de programación creció y trabajó nuevas formas, que si bien han servido para generar nuevos lenguajes programación; en otros casos, han nutrido el repertorio de algún otro.
Ejemplo de ello, C# incluyó el concepto de soporte para genéricos en su nueva versión.

Alrededor del auge de la cantidad de lenguajes de programación comienzan a surgir disyuntivas metodológicas en el campo académico, temáticas en el campo de investigación y tecnológicas en el campo empresarial.

Estos tres campos serán desarrollados con mayor detalle a continuación.

1.1 Impacto de los lenguajes de programación a nivel académico

En el año 1995, la revista SIGCSE publica una investigación [Hartel 95] para saber si el uso de los laboratorios es adecuado para reforzar la enseñanza de un nuevo lenguaje de programación o si éste limita la investigación teórica para mejorar los lenguajes de programación existentes. En su estudio, cuestiona las tendencias divergentes en los programas de ciencias de computación, los cursos previos que se deben llevar antes de enseñar algún lenguaje de programación (matemática discreta, algoritmos, entre otros) y los paradigmas de programación que deben incluirse en el proceso de aprendizaje de lenguajes de programación.

Dentro de los conceptos de programación se identifica y da relevancia a la abstracción, recursión y generalización. Finalmente, cita a King y a Luker para destacar que el estudio comparativo de lenguajes de programación es fundamental tanto como una profunda comprensión de los fundamentos de los paradigmas.

[Luker 89] indica los siguientes puntos con respecto a los lenguajes de programación:

- Pequeños lenguajes[2] no son tan atractivos pero son más fáciles de verificar, más predecibles en la implementación, y tienen una mejor base para una programación larga. Cuando un lenguaje es largo, tiende a tener una semántica ambigua y no es fácil de hacer un seguimiento lógico. Por otro lado, la brevedad no es necesariamente una virtud, porque va de la mano con la ilegibilidad.
- El código de un programa debe ser legible para facilitar mantenimiento y desarrollo.
- La fuerte tipificación conduce a menos errores de ejecución y disminuye costos.
- Los lenguajes de programación como FORTRAN y COBOL que nacieron entre los años 1955 y 1960, han servido de patrón para la creación de nuevos lenguajes de programación. Por ello se cuestiona la evolución de los lenguajes de programación porque cada vez se hacían mucho más complicados; sin embargo los principios básicos no cambiaban en absoluto. Al comparar Pascal con FORTRAN, encuentra un

[2] Lenguajes de programación con menos palabras reservadas, con una semántica bien definida que sea dócil a la manipulación lógica. No es regido estrictamente a un paradigma de programación.

mayor número de similitudes que notables diferencias; y si hay diferencias, éstas son superficiales. Concluye que en cierto nivel de abstracción, todos los lenguajes de programación, a la fecha 1981, son los mismos; donde los lenguajes imperativos son secuenciales, y presentan a la asignación como acción básica. Además incita al uso de lenguajes funcionales como base para todo tipo de programación porque están basados en un claro y constante uso del formalismo matemático. Las matemáticas son exactas y no es el único científico que exhorta el uso de lenguajes funcionales. Los investigadores [Henderson 80] [Turner 82] [Glaser 84] también lo hacen.

- Rescata el concepto de integridad de software como producto en lugar de la modularidad, buen estilo y costo.
- Indica la necesidad de desarrollo de herramientas y técnicas de verificación de códigos de programas.
- Plantea la incorporación de más horas de teoría en los cursos de ciencias de la computación, comenzando por "algoritmia" como base de resolución de problemas para luego ser desarrollado con cualquier lenguaje de programación. Asimismo exhorta la promoción de más horas de práctica en la enseñanza de los lenguajes de programación. Dados los fundamentos de programación, los alumnos deben estar capacitados a entender su propio código y entender el código desarrollado por otros. Exige también maestros en lenguajes de programación, en lugar de entrenadores.
- Aunque los educadores lleguen a un acuerdo acerca de los lenguajes de programación que deben ser impartidos en los primeros ciclos, hay un gran abismo entre el punto de vista del empleador y las prácticas que tienen la mayoría de graduados. Entonces, resurge el círculo vicioso: universidad - empresa. El día en que se pueda generar un correcto programa que cumpla las especificaciones requeridas por el usuario, es aún lejano. Tan sólo estamos limitados a evaluar algoritmos codificados individualmente en pequeños módulos.
- Finalmente, aunque el lenguaje funcional no se convierta en el principal paradigma; éste dará a los estudiantes una buena educación en cuestiones importantes en programación, y les servirá hasta que la historia de la computación logre cambiar al paradigma que se producirá por la entrada del lenguaje natural.

La mayoría de estos puntos expuestos son respaldados quince años más adelante, cuando [Vujošević-Janičić 05] realiza una investigación acerca de la decisión crítica de la enseñanza del primer lenguaje de programación y su paradigma, que afecta la formación y futuro de un programador. Indica que esta decisión es crítica ante la gran proliferación

de los diferentes lenguajes de programación en la sociedad que vive apoyada en las tecnologías de la información. El estudio se enfoca en los lenguajes que han sobrevivido más de diez años y evalúa la escritura eficiente, código legible, aplicación de algoritmos elegantes, códigos libres de errores de programación capturados por el compilador, y la no utilización del paradigma funcional. Resalta a los paradigmas de programación: imperativo, orientado a objetos, paradigma funcional y lógico. Además indica que, el primer lenguaje de programación, independiente del paradigma, que debe enseñarse debe ser: simple, con uso sencillo de operaciones de entrada / salida, legible y coherente de sintaxis, conjunto de funciones pequeñas y ortogonales, construcciones de programación sintácticamente diferenciadas entre sí (incluso si son similar en concepto, funcionalidad, o la aplicación).

El estudio de [Vujošević-Janičić 05] logra presentar una estadística acerca del uso de los lenguajes de programación en cursos introductorios en diferentes instituciones académicas y de investigación en Norteamérica, entre los años 1994 y 2001.

A nivel académico en Norteamérica:

Lenguaje de programación	Porcentaje de uso en 1994	Porcentaje de uso en 1996	Porcentaje de uso en 1999	Porcentaje de uso en 2001	Tendencia
Pascal	40.4	30.1	27.1	25	-
C++	3.6	17.1	19.0	26.6	+
Ada	15.2	14.6	18.3	13.7	-
C	8.2	10.0	10.6	12.6	+
Scheme	12.4	10.0	9.7	8.9	-
Modula	13.4	9.6	9.2	4.6	-
Java	0.0	0.0	2.7	6.4	+
Otros	6.9	8.5	3.3	2.2	-
Número de escuelas	388	510	546	372	

Fuente: Traducción del estudio de [Vujošević-Janičić 05]

Por otro lado, un estudio de [De Raadt 02], logra presentar estadísticas del año 2001 en Australia, donde se muestra la siguiente tendencia:

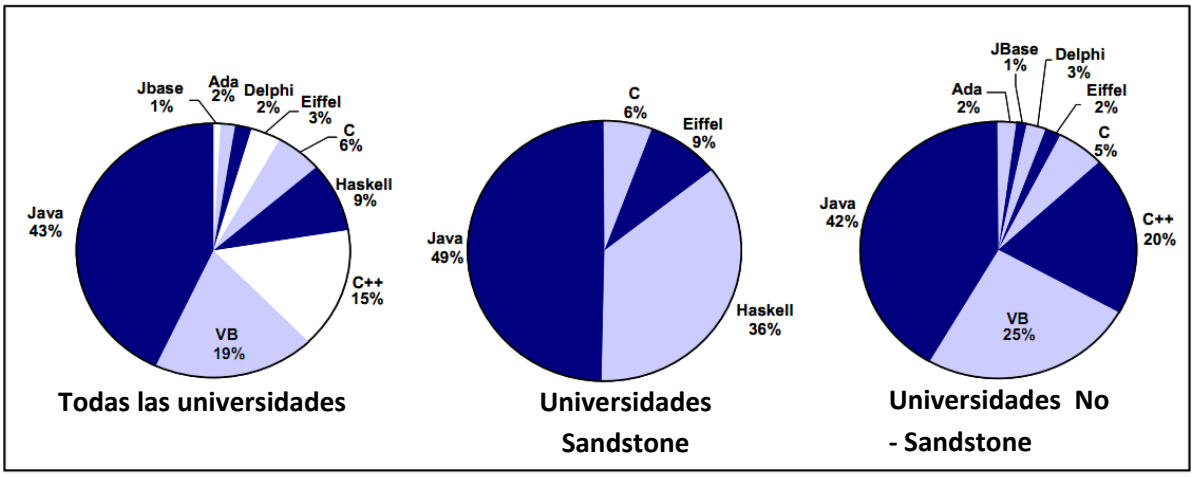
Cursos	57
Universidades	37
Estudiantes(aproximadamente)	19 900
Promedio de estudiantes por curso	349
Promedio de años de uso del lenguaje de programación	4.13

Tabla 1.1: Traducción de los Resultados Generales [De Raadt 02]

Lenguaje de programación	Utilizado actualmente	Previamente estudiado
Java	23	1
Visual Basic	14	2
C++	8	6
C	4	8
Haskell	3	1
Eiffel	2	
Ada	1	4
Delphi	1	1
JBase	1	
Pascal		13
Modula2		3
Smalltalk		3
Miranda		2
Otros (Basic, Blue, Cobol, DBase, Gopher, Turing)		(6)

Tabla 1.2: Traducción de los lenguajes de Programación utilizados y previamente estudiados [De Raadt 02]

Figura 1.1: Tendencia de los lenguajes de programación: Uso ponderado por número de estudiantes.



Relevancia en el industria/ demanda de estudiante	33
Beneficio pedagógico que se obtiene del lenguaje	19
Estructura del plan curricular/ Políticas del departamento	16
OOP lenguaje requerido	15
Interface GUI	6
Disponibilidad/ Costo para estudiantes	5
Fácil de encontrar en textos apropiados	2

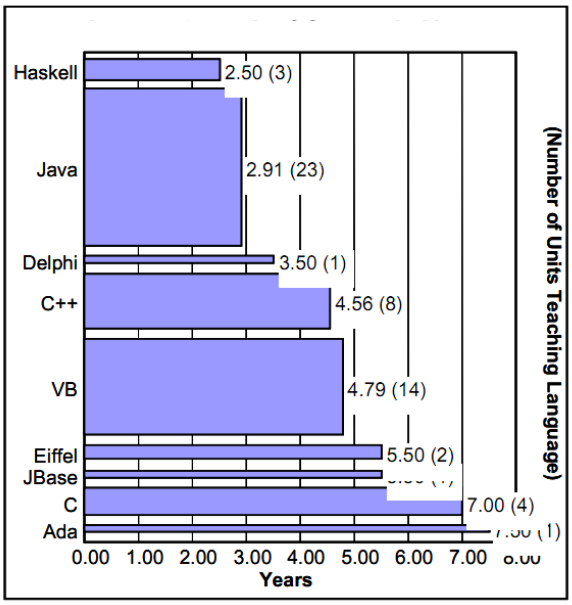
Tabla 1.3: Traducción de las razones por la cual se dictan los lenguajes de programación en universidades [De Raadt 02]

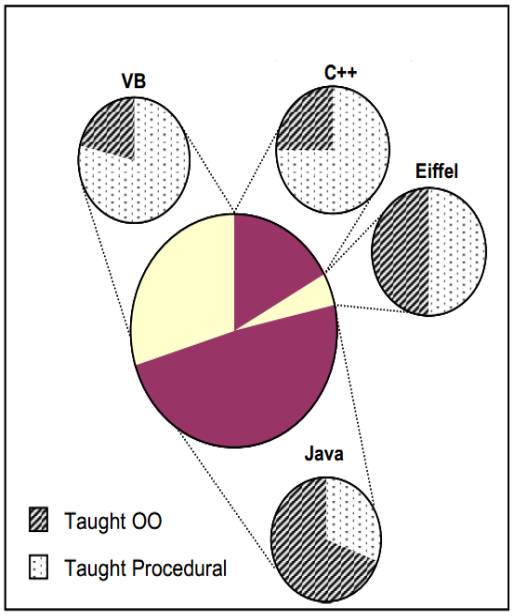
Beneficio pedagógico del lenguaje	6
Relevancia de la industria/ marketing/ demanda del estudiante	4
Estructura del plan curricular/ Políticas del departamento	2
Disponibilidad/ Costo para estudiantes	2
OOP lenguaje requerido	2
Fácil de encontrar en textos apropiados	1

Tabla 1.4: Traducción de las razones por la cual se dictan los lenguajes de programación en las universidades “Sandstone” [De Raadt 02]

Figura 1.2: Lado Izquierdo - Promedio de años de enseñanza de los lenguajes de programación

Figura 1.3: Lado Derecho – Paradigmas utilizados para enseñar lenguajes de programación OO





Frente a la manera de cómo se ha estado llevando la enseñanza de los lenguajes de programación, y cómo éstos deben ser impartidos en las universidades para estar a la vanguardia de las exigencias del mercado, la institución ACM conjuntamente con la institución IEEE elaboran cada 10 años un plan curricular guía para la sección “Ciencias de la computación” y para otras ciencias. El último reporte fue emitido en el año 2001 y los comentarios acerca del impacto teórico-práctico ha sido motivo de otra publicación en el año 2008. [ACM 08]. Aquí dos grandes consideraciones:

- Con respecto a las ciencias de computación, luego del diálogo con la industria, se recibe atención de los siguientes puntos: Seguridad, que incluye acceso a redes, encriptación y escritura de código seguro; problemas de calidad, que incluye pruebas, depuración, seguimiento de errores, comprobación de la legibilidad del código, documentación y revisión de código. Principios de ingeniería de software y técnicas, gestión de versiones, control de código, mejores prácticas para el desarrollo de software en grupo de personas y ajuste del rendimiento, funcionalidad y cálculos en el diseño de los terminales.
- Con respecto a los lenguajes de programación y los paradigmas, se obtuvo un considerable debate en el momento de la consulta pública final. Por una parte se expresa un acuerdo en que los alumnos deben estar expuestos a más de un paradigma de programación porque los profesionales en el área suelen utilizar

diferentes lenguajes de programación para programar diferentes proyectos con diferentes propósitos. Los estudiantes deben ser capaces de aprender nuevos lenguajes durante su carrera, y como resultado, deben reconocer los beneficios de aprender y aplicar nuevos lenguajes de programación cuando estos aparezcan. Por otra parte, el debate se da por la enseñanza del paradigma funcional en cursos de pregrado como primer paradigma a enseñarse. A su vez, el paradigma secundario adecuado dependerá en gran medida de las características específicas y los objetivos educativos de cada institución. Si la universidad busca preparar estudiantes para puestos en el ámbito académico y de investigación; entonces y sólo entonces en ese caso, es conveniente introducir la programación funcional. Con ello, los estudiantes puedan tomar ventaja de la disciplina mental que los lenguajes brindan.

ACM, a través de su revista “CSTA: The Voice of K–12 Computer Science Education and its Educators” ofrece la oportunidad de compartir experiencias, debates, programas de motivación, herramientas y estadísticas. Los artículos publicados son realizados por educadores e investigadores en temas relacionados a las Ciencias de la Computación. En Setiembre del año 2011, con el apoyo de National Science Foundation se lanza una publicación “The Fine Art of Programming” por Karen North, donde muestra las diferentes maneras y programas que se aplican en la enseñanza de lenguajes de programación para mejorar cifras de impacto e interés, desarrollo eficiente, y enfatiza los programas que involucran atracción del sexo femenino en el área. [CSTA 11]

Otro estudio realizado en la Universidad de Copenhagen [Sestoft 08], toma a los lenguajes de programación C# y Scheme como lenguajes instructivos, y a los lenguajes Java y .NET y sus respectivas plataformas para la implementación de software. Asimismo define conceptos referidos al compilador como asignación de registros, optimización de bucle intensivo en código imperativo.

Del estudio se resalta lo siguiente:

- La diferencia entre la “competencia orientada” frente a la “descripción de un tema orientado”. Es decir, no es lo mismo hacer que un estudiante utilice un analizador sintáctico en un curso, ejemplo LL-analizador, y hacer que un estudiante pueda diseñar un generador de analizadores sintácticos y léxicos para implementar un intérprete basado en la continuación para un subconjunto de íconos.
- El estudiante debe ser capaz de explicar el desempeño (tiempo y espacio) de un programa escrito con C o Java, cuando éstos se ejecutan en un hardware moderno, y estimar si una construcción del lenguaje es preferible a otra, dependiendo de la

implementación requerida. Asimismo, debe ser capaz de utilizar herramientas en un nivel superior de sofisticación, ejemplo: manipulación recursiva, utilización de funciones de orden superior o de sus equivalentes. Utilizar herramientas bien establecidas para regular expresión coincidente, léxico y análisis sintáctico, para la entrada de texto y para la construcción de representaciones abstractas de sintaxis, expresividad, diseños y características de rendimiento de los lenguajes de programación. Los estudiantes deben ser capaces de evaluar críticamente las "nuevas" tecnologías que se crean y relacionan, evaluando principios académicos y la evolución a través de la historia de los lenguajes de programación. Ejemplo de las técnicas de implementación que van apareciendo son: la recolección de basura (1960), la programación orientada a objetos (1967), polimórfica los tipos y la inferencia de tipos (1978) y el concepto de reflexión en versiones de Java 5.0 y C # 2.0.

- Los conceptos y construcciones que no son mostrados por Java y deben promoverse son: tipo dinámico, co-rutinas, concurrencia y distribución, concordancia de patrones, evaluación perezosa y tiempo de ejecución de la generación de código.

Se deduce dentro del panorama mundial, que mientras en las universidades de Norteamérica y Europa, se investigan temas de lenguajes de programación, sus respectivos paradigmas en el ámbito académico, y su impacto en la industria del Software; a nivel académico Perú, recién se están formando y consolidando las propuestas para el plan curricular que mejor se adecue a la realidad peruana en lo referente a la industria del software. En el año 2008, [Alvarez, Pow Sang 08] proponen establecer cuatro perfiles profesionales; los cuales permiten definir los programas de las Ciencias de la Computación, Ingeniería de la Computación, Ingeniería Informática y Carrera Técnica en Informática. Los tres primeros como programas de licenciatura en las universidades y el cuarto en una carrera técnica en instituciones de educación técnica. Esta definición surge a partir de la interpretación confusa que se da a la carrera de Ingeniería de Sistemas en las universidades peruanas, que muchas veces influenciadas por programas de otras universidades de Latinoamérica y del exterior, han fusionado planes curriculares y conceptos de la Ingeniería de Sistemas, Ciencias de la Computación e Ingeniería de Software.

1.2 Impacto de los lenguajes de programación a nivel industria

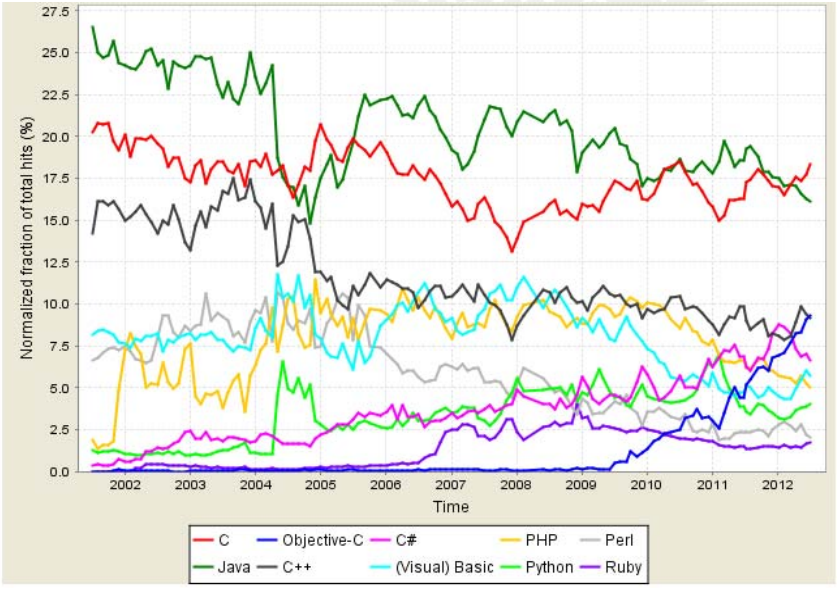
A nivel industria, una publicación de [Vujošević-Janičić 05] presenta un cuadro de proyecciones en el año 2008, basado en el contenido de las páginas Web http://www.langpop.com/ y http://www.tiobe.com:

Posición Noviembre 2008	Posición Noviembre 2007	Variante en Posición	Lenguaje de programación	Clasificación Noviembre 2008	Variante en Noviembre 2007
1	1	=	Java	20.3%	-0.2%
2	2	=	C	15.3%	+1.3%
3	4	+	C++	10.4%	+1.6%
4	3	-	Visual Basic	9.3%	-0.9%
5	5	=	PHP	8.9%	+0.3%
6	7	+	Python	5.1%	+0.9%
7	8	+	C#	4.1%	+0.1%
8	11	+++	Delphi	4.0%	+1.6%
9	6	- - -	Perl	3.9%	-0.9%
10	10	=	JavaScript	2.9%	+0.0%

Fuente: Estudio de [Vujošević-Janičić 05]

* La consulta realizada a la página de la comunidad de programadores Tiobe al año 2012:

Figura 1.4: Indices de la comunidad de programación TIOBE



Fuente: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (al 05 de Julio 2012)

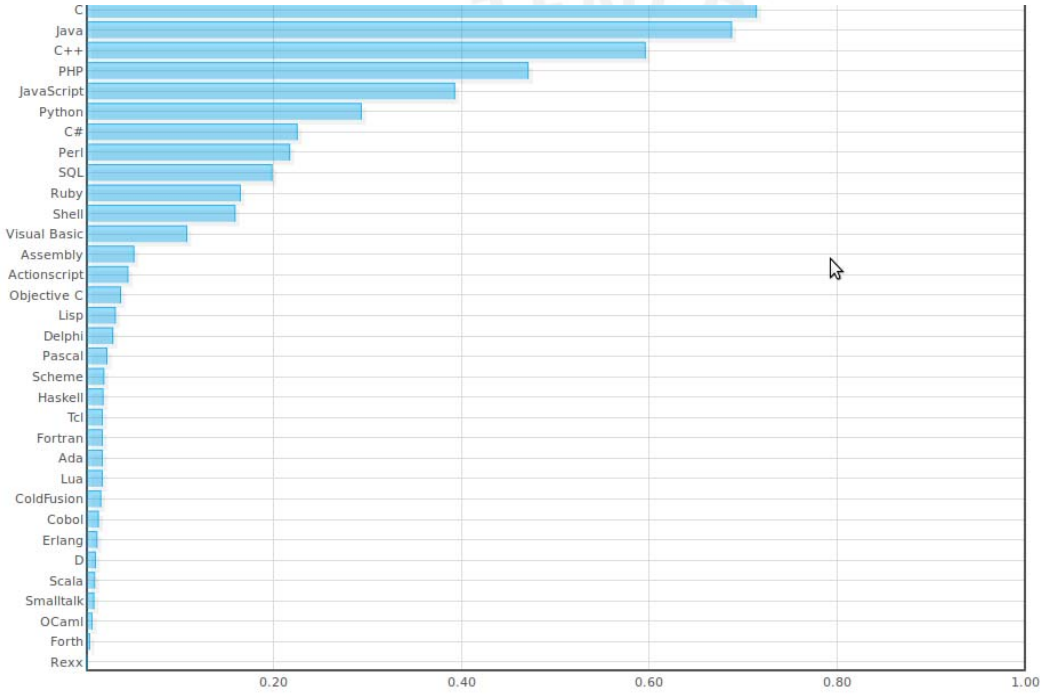
En el gráfico se puede ver que los lenguajes de programación más utilizados por la comunidad de programadores según Tiobe, son Java, C y C++.

No aparecen ni el lenguaje de programación Haskell, ni Go en la estadística.

* La consulta realizada a langpop.com al año 2012 muestra a C, Java y C++ como lenguajes populares y más utilizados en los buscadores como Yahoo, Google Docs, entre otros.

Haskell aparece en la lista dentro de los 20 primeros lenguajes de programación, luego de LISP, Asembler, Pascal y Scheme.

Figura 1.5: Estadística de popularidad de los lenguajes de programación en langpop.com



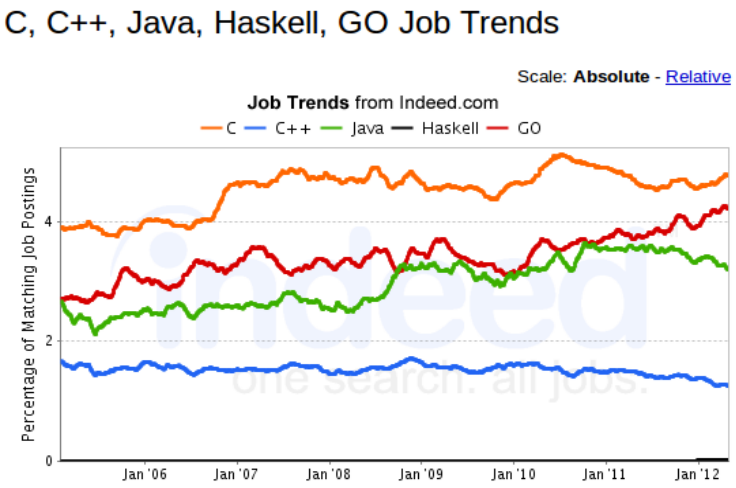
Fuente: http://langpop.com/ (al 05 de Julio 2012)

Por otro lado, la página Web indeed.com muestra indicadores de empleo relacionados con los lenguajes de programación empleados.

A continuación se presenta una estadística de las tendencias y requerimientos de lenguajes de programacion para las empesas.

Hay que tener en cuenta que a pesar de que Haskell y Java fueron creados casi a la par, se tiene un registro de uso de framework desde mediados del año 1999 para Java con Servlet incluido en Java 1.2, mientras que frameworks de Haskell, recién están tomando protagonismo a partir del año 2011 (Ver Anexo 05).

Figura 1.6: Estadistica de demanda de trabajo para desarrolladores en los lenguajes de programación C, C++, Java, Haskell y GO.

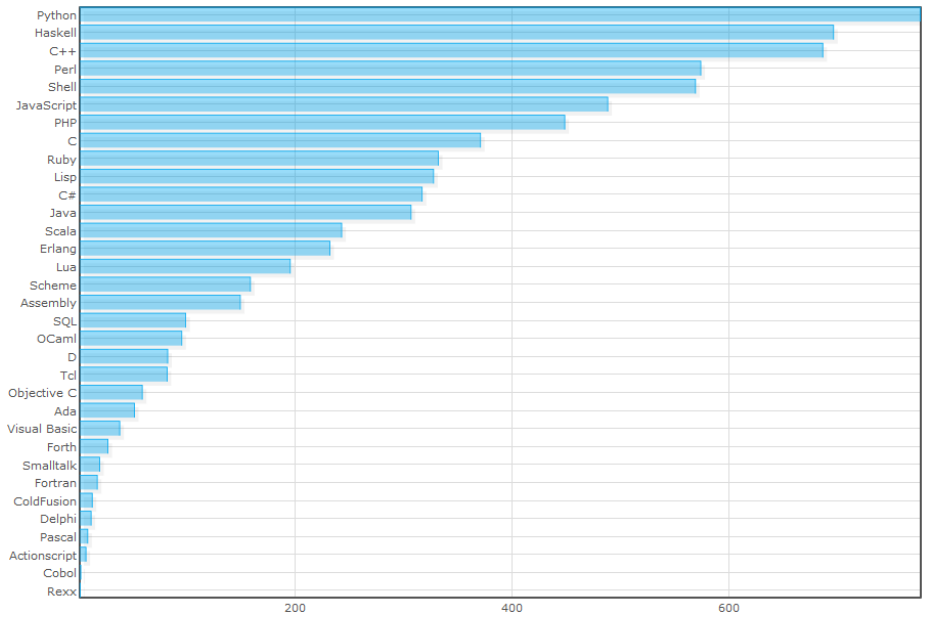


Fuente: http://www.indeed.com/jobtrends?q=C%2C+C%2B%2B%2C+Java%2C+Haskell%2C+GO&l= (al 05 de Julio 2012)

Se C aún lidera la lista de demanda de programadores en la actualidad para trabajos de campo, una de las razones de ello es que C es necesario para transacciones bancarias.

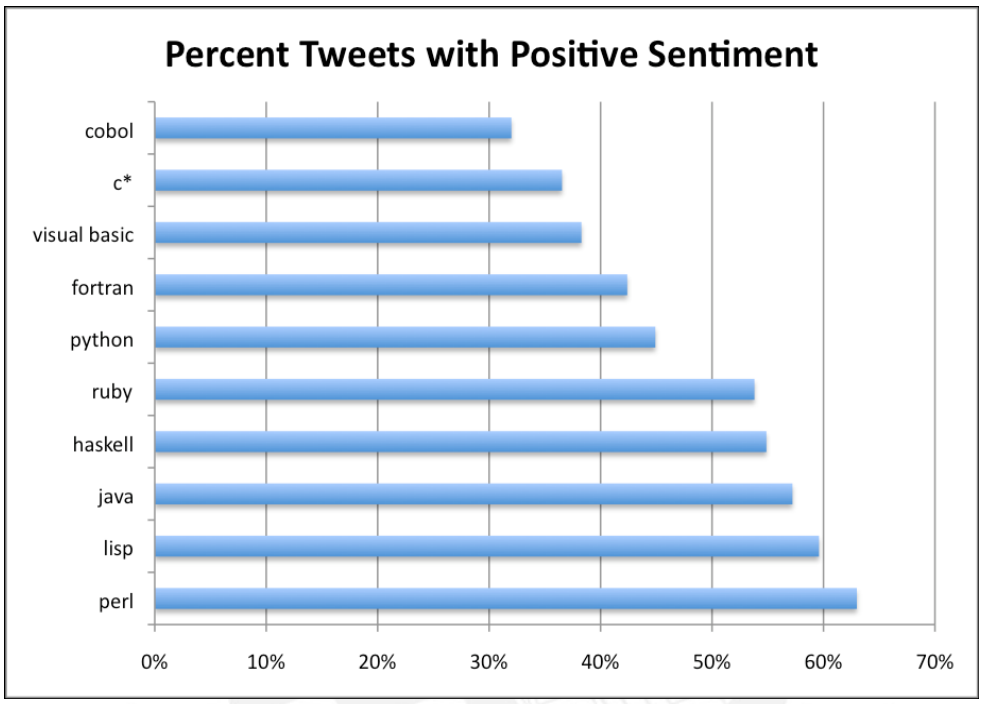
Por otro lado, estudiantes de ciencias de computación y programadores comentan o preguntan y postean en diferentes páginas Web acerca del lenguaje de "moda". Usualmente la respuesta va orientada a los lenguajes interpretados como Python, Ruby, C++ o Perl, lenguajes que superan la complejidad de código de C en manejo de punteros o caídas de sistema por mala asignación de memoria. langpop.com muestra una estadística de los lenguajes de programación que la gente comenta en *FreenodeIRC.*

Figura 1.7: Estadistica de los lenguajes de programación comentados mayormente en FreenodeIRC



Otro estudio acerca de los sentimientos de los programadores, fue realizado en el año 2009. Este estudio se basó en realizar la búsqueda de los lenguajes de programación COBOL, Ruby, Fortran, Python, Visual Basic, Perl, Java, Haskell, Lisp y C utilizando las herramientas twitter y Amazon Mechanical Turk [3].

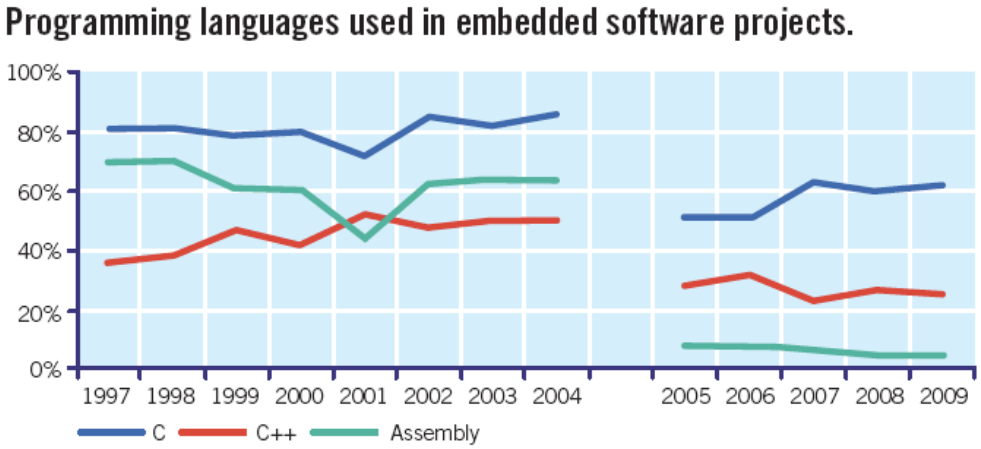
Figura 1.8: Estadística de los lenguajes de programación con sentimiento positivo en twitter



Fuente: http://ebiquity.umbc.edu/blogger/category/twitter/

Pese a que C no ocupe uno de los primeros lugares en los rankings de preferencias por parte de los programadores, ocupa el primer lugar en la demanda para desarrollo.

Figura 1.9: Lenguajes de programación en proyectos de software embebidos.



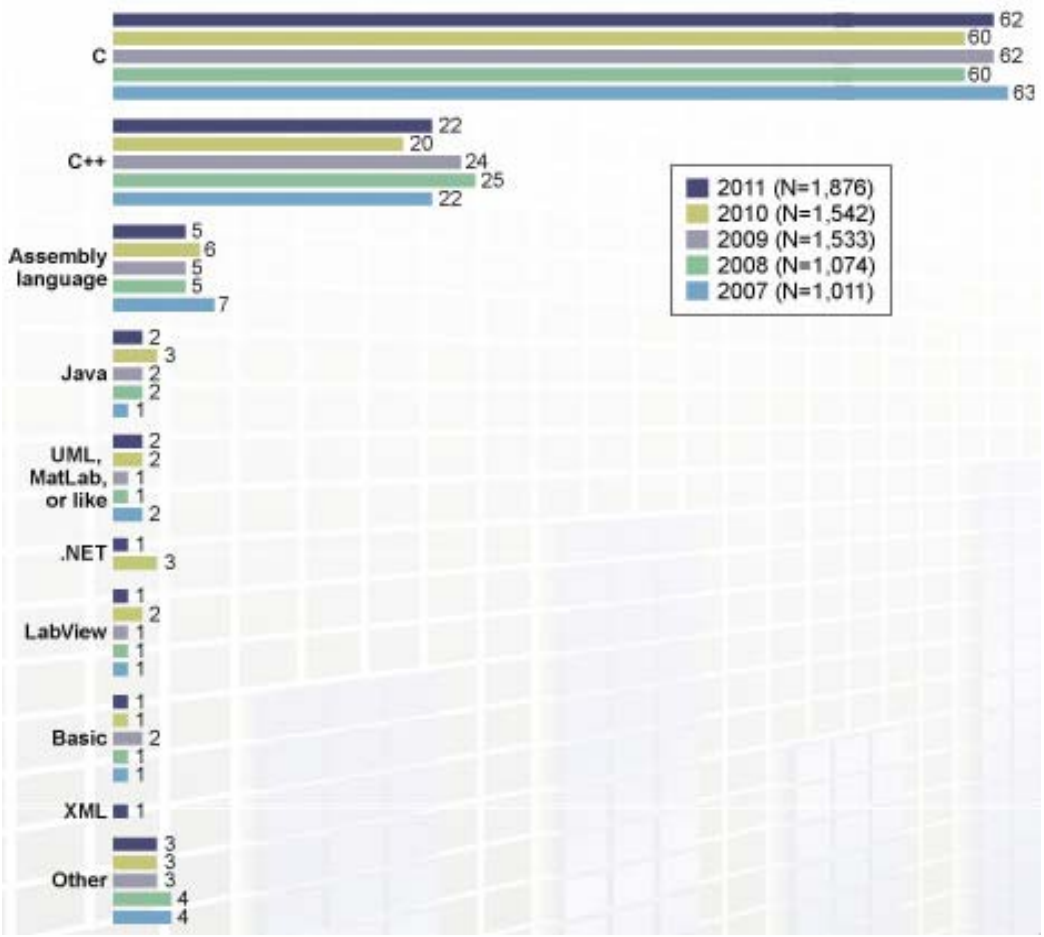
Fuente: http://www.eetimes.com/electronics-blogs/industrial-control-designline-blog/4027479/Real-men-program-in-C

[3] https://www.mturk.com/mturk/welcome

Se puede apreciar en la figura que la mayoría de los proyectos de sistemas embebidos a finales de los años 90s, hasta mediados de los años 2000 eran desarrollados y soportados en un 80% por el lenguaje de programación C, y aunque la mayoría de programadores tengan el sentimiento o tendencia a dejar de aprender C porque existen lenguajes de programación como Python, C++, Java que simplifican muchos códigos de C; aún C está vigente en diferentes proyectos en diferentes librerías como GTK (GIMP ToolKit)[4], GDK (GIM Drawing Kit)[5] por ejemplo.

Un estudio realizado en el año 2011 por Embedded.com[6] reafirma que, C es el lenguaje más utilizado por los sistemas embebidos.

Figura 1.10: Lenguajes de programación en proyectos de software embebidos.



[4] http://en.wikipedia.org/wiki/GTK%2B

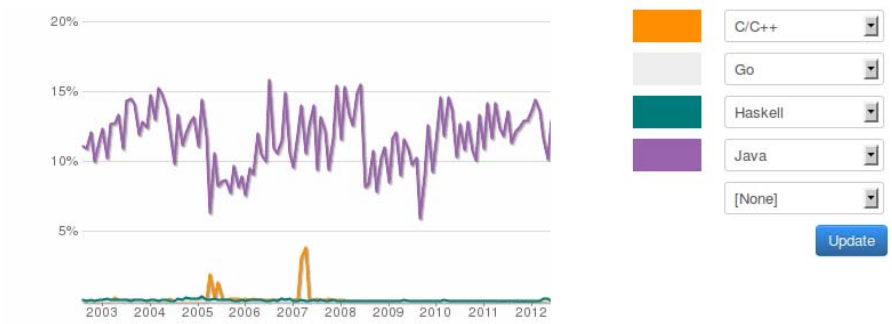
[5] http://en.wikipedia.org/wiki/GDK

[6] http://e.ubmelectronics.com/embeddedstudy/index.html

La figura muestra una encuesta anual a los diseñadores a lo largo del mundo que muestra las tendencias en software y hardware utilizados por varias compañías en el ecosistema de embebidos electrónicos.

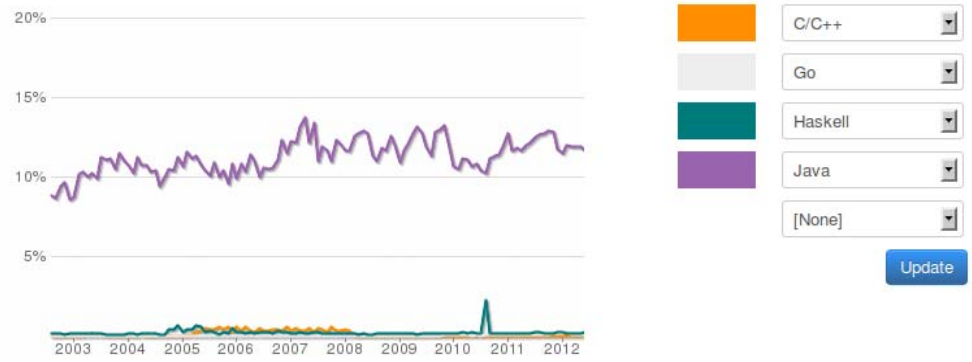
A nivel de proyectos de software libre y de código abierto mundial, se tienen muchos repositorios que registran miles de líneas de código. Una página Web que recopila en tiempo real el registro de los repositorios como GIT, Mercurial y otros, es http://www.ohloh.net/. Se puede consultar estadísticas como la cantidad de líneas modificadas, commits realizados, contribuidores y proyectos activos por mes y por año.

Figura 1.10: Líneas de código cambiadas mensualmente (porcentaje del total):



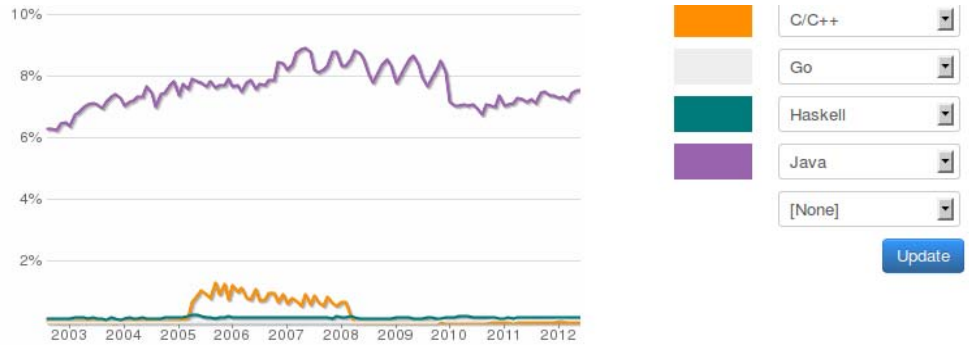
Fuente:http://www.ohloh.net/languages/compare?commit=Update&l0=c&l1=-1&l2=java&l3=haskell&l4=-1&measure=loc_changed&percent=true * al 02 de Julio del 2012

Figura 1.11: Commits realizados mensualemente en proyectos de software libre:



Fuente:http://www.ohloh.net/languages/compare?commit=Update&l0=c&l1=-1&l2=java&l3=haskell&l4=-1&measure=commits&percent=true *al 02 de Julio del 2012

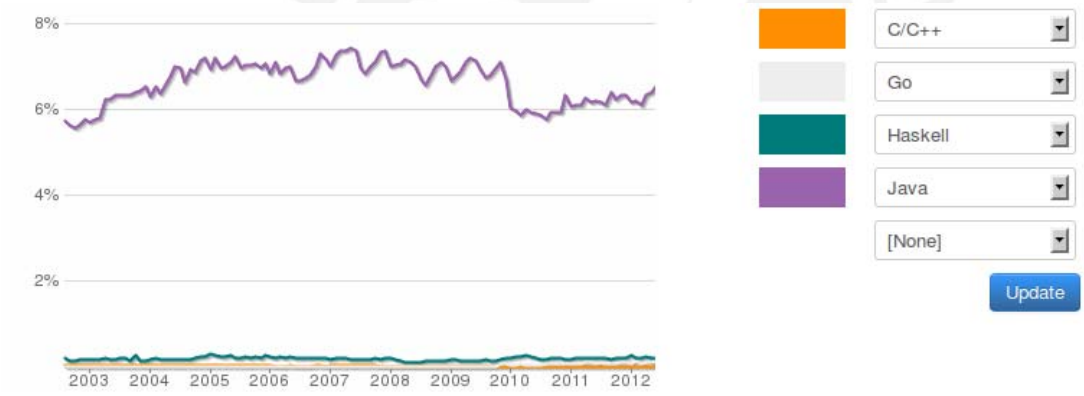
Figura 1.12: Contribuidores activos mensualmente en proyectos de software libre:



Fuente: http://www.ohloh.net/languages/compare?commit=Update&l0=c&l1=-1&l2=java&l3=haskell&l4=-1&measure=contributors&percent=true * al 02 de Julio del 2012

Figura 1.13:

Proyectos mensuales de software libre que utilizan los lenguajes de programación C, Java, Haskell y GO:



Fuente:http://www.ohloh.net/languages/compare?commit=Update&l0=c&l1=-1&l2=java&l3=haskell&l4=-1&measure=projects&percent=true *al 19 de Junio del 2012

En las figuras 1.10, 1.11, 1.12 y 1.13, se muestran las líneas de la suma de líneas de código cambiadas por mes, la cantidad de commits realizados mensualmente por desarrolladores en proyectos de software libre, la cantidad de proyectos con al menos una línea de código cambiada en un mes, y el número de desarrolladores que contribuyeron con al menos una línea de código mensualmente; y se puede apreciar que el lenguaje de programación más utilizado en los proyectos de software libre es el lenguaje de programación Java.

En la figura 1.13 aún se ve una gran diferencia entre Java y los demás proyectos en Haskell, C, C++ y GO, pero se rescata el segundo puesto que ocupa Haskell.

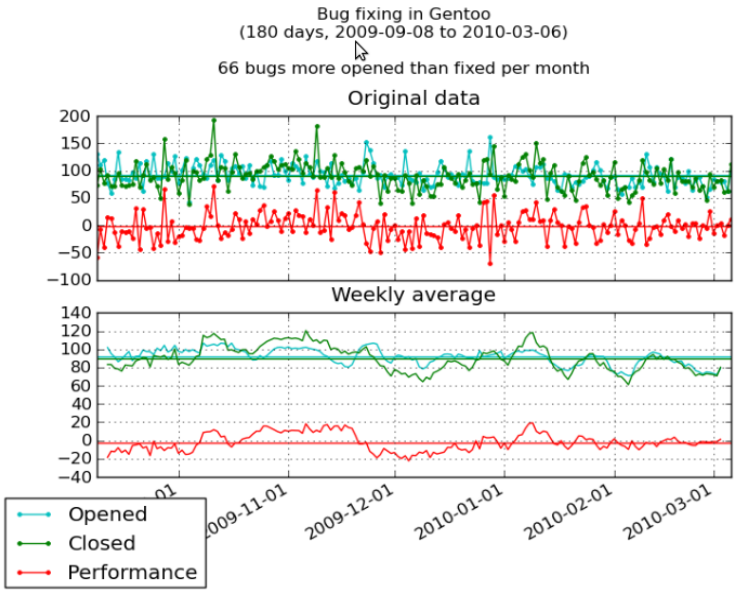
Una explicación de ello, radica en que Java tiene presencia de casi un 100% en la programación para dispositivos móviles, en especial en el mercado Android. Siendo el año 2011, un año donde el mercado de dispositivos móviles creció exponencialmente (Ver anexo 02)

Otra razón para Java, se da en el uso de las tecnologías para aplicaciones web, el uso de Java en los servidores GAE (Google App Engine)[7], los servicios Web de Amazon a través de su producto AWS SDK for Java y la implementación de servidores con JavaEE[8].

El mantenimiento de proyectos de software existentes codificados con Java, y la codificación de las soluciones a bugs o errores de programación también contribuye con el gran porcentaje de actualizaciones y mantenimiento obtenido con Java.

A continuación en la figura se puede apreciar los bugs que fueron abiertos, resueltos y otros que aún están pendientes entre los años 2001 y 2012.

Figura 1.14: Cantidad de bugs se han producido en estos últimos años, causando un fuerte impacto en los costos de mantenimiento de softwares.



Fuente: http://git.goodpoint.de/?p=gentoo-bug-heartbeat.git;a=summary (Octubre 2010)

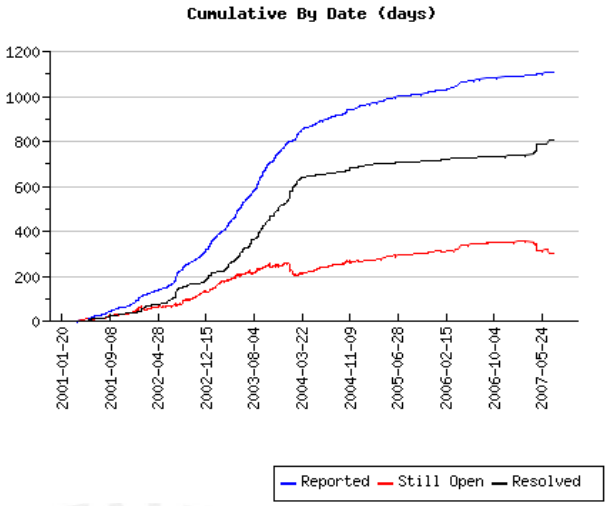
[7] https://developers.google.com/appengine/docs/java/gettingstarted/

[8] http://javajee.com/

Fuente:

http://www.plkr.org/i/tmp/graph_cumulative_bydate.png

(Julio 2011)



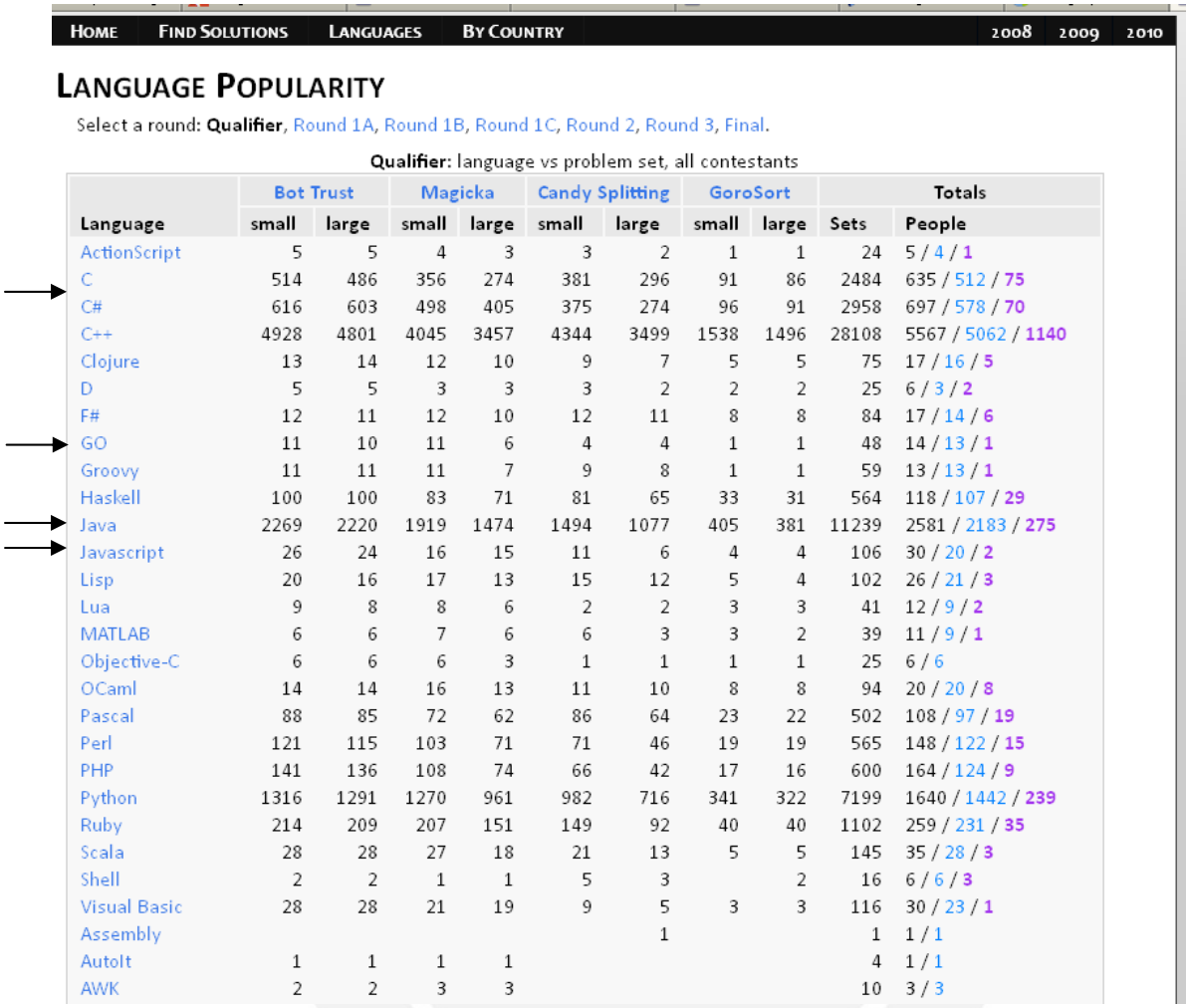
Los lenguajes de programación: C, C++, Java, Haskell y GO están siendo utilizados con gran popularidad actualmente en competencias internacionales que promueve Google para desarrolladores, como el programa "Google Code Jam[9]".

Este programa reúne a programadores profesionales de todo el mundo para resolver problemas difíciles los cuales son vistos muchas veces como "algoritmos rompecabezas". En el año 2011, compitieron 30 000 programadores, y el ganador fue un programador japonés, quien se llevó un premio de $10 000. En la figura 1.15, se muestra el listado de los lenguajes de programación (en orden alfabético) más utilizados en la competencia.

En la parte de total se puede ver en la columna People el total de programadores que participaron, seguido por el número de programadores que pasan a la siguiente ronda, y seguido por los que obtuvieron un puntaje perfecto.

[9] http://code.google.com/codejam

Figura 1.15: Listado de los lenguajes de programación que calificaron para el concurso:



HOME | FIND SOLUTIONS | LANGUAGES | BY COUNTRY | 2008 | 2009 | 2010

LANGUAGE POPULARITY

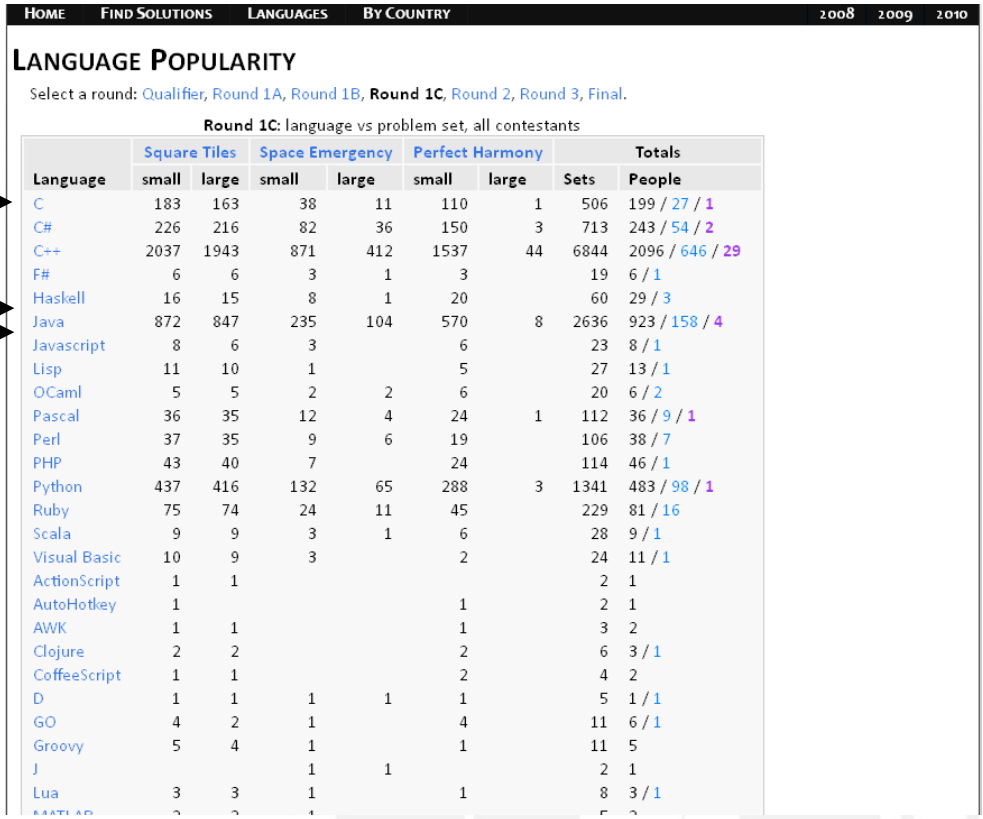
Select a round: **Qualifier**, Round 1A, Round 1B, Round 1C, Round 2, Round 3, Final.

Qualifier: language vs problem set, all contestants

	Bot Trust		Magicka		Candy Splitting		GoroSort		Totals	
Language	small	large	small	large	small	large	small	large	Sets	People
ActionScript	5	5	4	3	3	2	1	1	24	5 / 4 / 1
C	514	486	356	274	381	296	91	86	2484	635 / 512 / 75
C#	616	603	498	405	375	274	96	91	2958	697 / 578 / 70
C++	4928	4801	4045	3457	4344	3499	1538	1496	28108	5567 / 5062 / 1140
Clojure	13	14	12	10	9	7	5	5	75	17 / 16 / 5
D	5	5	3	3	3	2	2	2	25	6 / 3 / 2
F#	12	11	12	10	12	11	8	8	84	17 / 14 / 6
GO	11	10	11	6	4	4	1	1	48	14 / 13 / 1
Groovy	11	11	11	7	9	8	1	1	59	13 / 13 / 1
Haskell	100	100	83	71	81	65	33	31	564	118 / 107 / 29
Java	2269	2220	1919	1474	1494	1077	405	381	11239	2581 / 2183 / 275
Javascript	26	24	16	15	11	6	4	4	106	30 / 20 / 2
Lisp	20	16	17	13	15	12	5	4	102	26 / 21 / 3
Lua	9	8	8	6	2	2	3	3	41	12 / 9 / 2
MATLAB	6	6	7	6	6	3	3	2	39	11 / 9 / 1
Objective-C	6	6	6	3	1	1	1	1	25	6 / 6
OCaml	14	14	16	13	11	10	8	8	94	20 / 20 / 8
Pascal	88	85	72	62	86	64	23	22	502	108 / 97 / 19
Perl	121	115	103	71	71	46	19	19	565	148 / 122 / 15
PHP	141	136	108	74	66	42	17	16	600	164 / 124 / 9
Python	1316	1291	1270	961	982	716	341	322	7199	1640 / 1442 / 239
Ruby	214	209	207	151	149	92	40	40	1102	259 / 231 / 35
Scala	28	28	27	18	21	13	5	5	145	35 / 28 / 3
Shell	2	2	1	1	5	3		2	16	6 / 6 / 3
Visual Basic	28	28	21	19	9	5	3	3	116	30 / 23 / 1
Assembly						1			1	1 / 1
AutoIt	1	1	1	1					4	1 / 1
AWK	2	2	3	3					10	3 / 3

Fuente: http://www.go-hero.net/jam/11/languages/0

Al final de la primera ronda:



HOME FIND SOLUTIONS LANGUAGES BY COUNTRY 2008 2009 2010

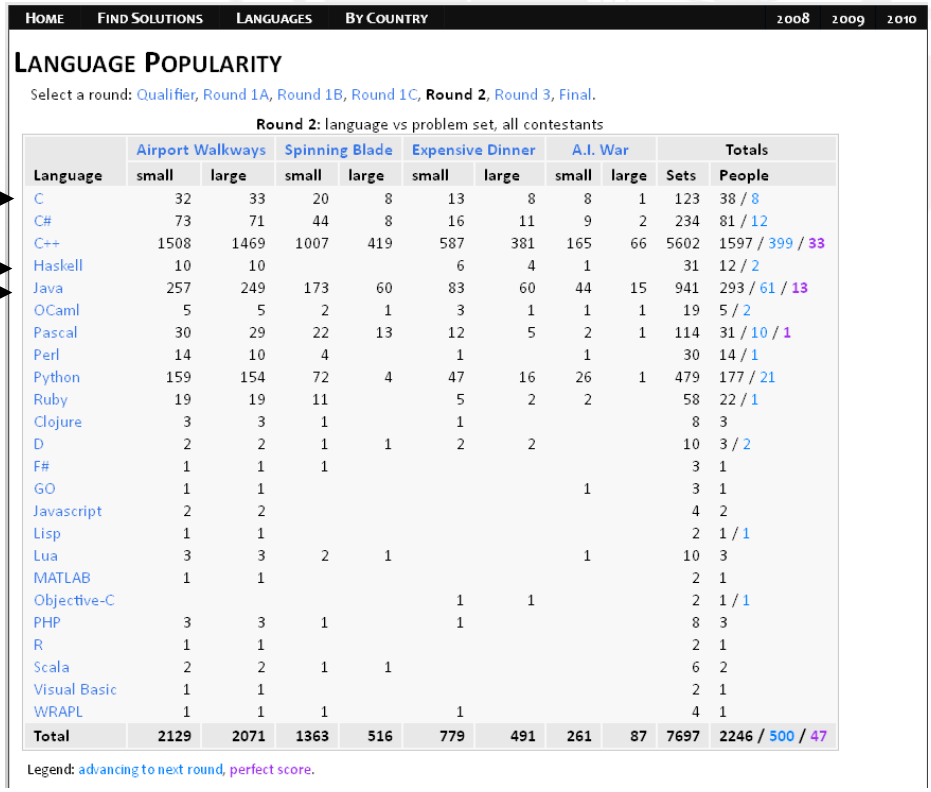
LANGUAGE POPULARITY

Select a round: Qualifier, Round 1A, Round 1B, **Round 1C**, Round 2, Round 3, Final.

Round 1C: language vs problem set, all contestants

	Square Tiles		Space Emergency		Perfect Harmony		Totals	
Language	small	large	small	large	small	large	Sets	People
C	183	163	38	11	110	1	506	199 / 27 / 1
C#	226	216	82	36	150	3	713	243 / 54 / 2
C++	2037	1943	871	412	1537	44	6844	2096 / 646 / 29
F#	6	6	3	1	3		19	6 / 1
Haskell	16	15	8	1	20		60	29 / 3
Java	872	847	235	104	570	8	2636	923 / 158 / 4
Javascript	8	6	3		6		23	8 / 1
Lisp	11	10	1		5		27	13 / 1
OCaml	5	5	2	2	6		20	6 / 2
Pascal	36	35	12	4	24	1	112	36 / 9 / 1
Perl	37	35	9	6	19		106	38 / 7
PHP	43	40	7		24		114	46 / 1
Python	437	416	132	65	288	3	1341	483 / 98 / 1
Ruby	75	74	24	11	45		229	81 / 16
Scala	9	9	3	1	6		28	9 / 1
Visual Basic	10	9	3		2		24	11 / 1
ActionScript	1	1					2	1
AutoHotkey	1				1		2	1
AWK	1	1			1		3	2
Clojure	2	2			2		6	3 / 1
CoffeeScript	1	1			2		4	2
D	1	1	1	1	1		5	1 / 1
GO	4	2	1		4		11	6 / 1
Groovy	5	4	1		1		11	5
J			1	1			2	1
Lua	3	3	1		1		8	3 / 1

En la ronda 2:



HOME FIND SOLUTIONS LANGUAGES BY COUNTRY 2008 2009 2010

LANGUAGE POPULARITY

Select a round: Qualifier, Round 1A, Round 1B, Round 1C, **Round 2**, Round 3, Final.

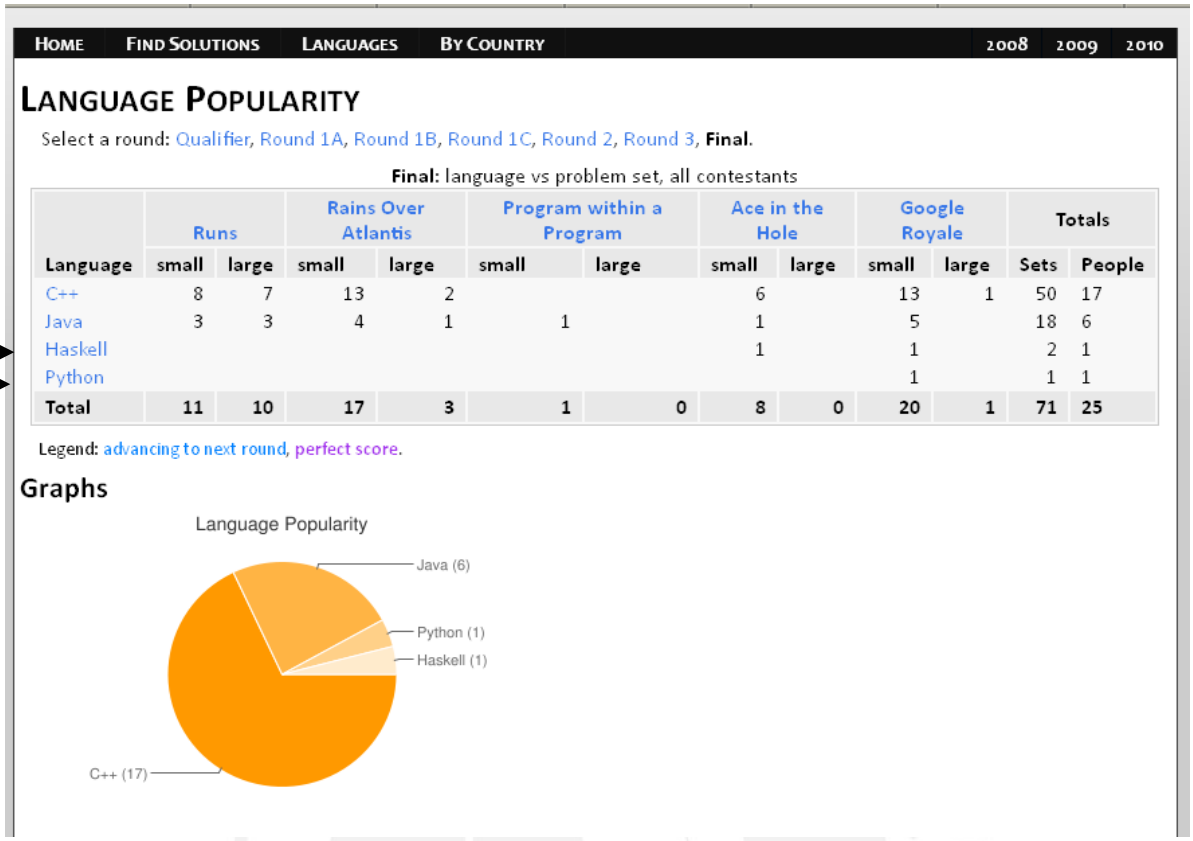
Round 2: language vs problem set, all contestants

	Airport Walkways		Spinning Blade		Expensive Dinner		A.I. War		Totals	
Language	small	large	small	large	small	large	small	large	Sets	People
C	32	33	20	8	13	8	8	1	123	38 / 8
C#	73	71	44	8	16	11	9	2	234	81 / 12
C++	1508	1469	1007	419	587	381	165	66	5602	1597 / 399 / 33
Haskell	10	10			6	4	1		31	12 / 2
Java	257	249	173	60	83	60	44	15	941	293 / 61 / 13
OCaml	5	5	2	1	3	1	1	1	19	5 / 2
Pascal	30	29	22	13	12	5	2	1	114	31 / 10 / 1
Perl	14	10	4		1		1		30	14 / 1
Python	159	154	72	4	47	16	26	1	479	177 / 21
Ruby	19	19	11		5	2	2		58	22 / 1
Clojure	3	3	1		1				8	3
D	2	2	1	1	2	2			10	3 / 2
F#	1	1	1						3	1
GO	1	1					1		3	1
Javascript	2	2							4	2
Lisp	1	1							2	1 / 1
Lua	3	3	2	1			1		10	3
MATLAB	1	1							2	1
Objective-C					1	1			2	1 / 1
PHP	3	3	1		1				8	3
R	1	1							2	1
Scala	2	2	1	1					6	2
Visual Basic	1	1							2	1
WRAPL	1	1	1		1				4	1
Total	**2129**	**2071**	**1363**	**516**	**779**	**491**	**261**	**87**	**7697**	**2246 / 500 / 47**

Legend: advancing to next round, perfect score.

Fuente: http://www.go-hero.net/jam/11/languages/0

Figura 3.5: Listado de los lenguajes de programación que llegaron a la ronda final del concurso:



HOME | FIND SOLUTIONS | LANGUAGES | BY COUNTRY | 2008 | 2009 | 2010

LANGUAGE POPULARITY

Select a round: Qualifier, Round 1A, Round 1B, Round 1C, Round 2, Round 3, **Final**.

Final: language vs problem set, all contestants

	Runs		Rains Over Atlantis		Program within a Program		Ace in the Hole		Google Royale		Totals	
Language	small	large	small	large	small	large	small	large	small	large	Sets	People
C++	8	7	13	2			6		13	1	50	17
Java	3	3	4	1	1		1		5		18	6
Haskell							1		1		2	1
Python									1		1	1
Total	11	10	17	3	1	0	8	0	20	1	71	25

Legend: advancing to next round, perfect score.

Graphs

Fuente: http://www.go-hero.net/jam/11/languages/6

Conforme pasan las rondas de preguntas, se puede ver el listado de lenguajes de programacion que logran llegar a la ronda final los lenguajes de programación.

Se obtuvo el siguiente resultado: C++(17), Java(6), Haskell(1) y Python(1).

Existen publicaciones como la de [Getov 10], donde se indica que es generalmente aceptado que el desarrollo de software basado en componentes y se está convirtiendo en la mejor opción costo-efectividad para la construcción de sistemas complejos y aplicaciones distribuidas. Sin embargo, esto hace propicia la búsqueda del modelo de programación más adecuado y sus entornos de programación correspondientes. Además, de buscar portabilidad y apoyo a propiedades dinámicas, temas críticos para permitir el rápido desarrollo de software de los sistemas informáticos grandes y complejos.

Se refiere principalmente a la construcción sistemática de sistemas de software complejos mediante el uso de un mayor nivel de abstracción de los componentes de software. Los componentes son construidos en bloques, con una granularidad específica, interfaces bien definidas y dependencias explícitas.

Aunque este enfoque reduce significativamente el esfuerzo de desarrollo tradicional, también introduce un nuevo paso de la composición o el acoplamiento de los componentes existentes en función del modelo de componentes adoptados.

Existe además una especificación que propone un modelo genérico, el componente escrito en el que los componentes son entidades que se comunican en tiempo de ejecución exclusivamente a través de interfaces. Tiene como características: soporte a la constitución jerárquica, instalación extensible de reflexión; es decir, cada componente se asocia con un conjunto ampliable de los controladores que permiten inspeccionar y modificar las características internas de los componentes. Los controladores proporcionan un medio para captar los comportamientos extra-funcionales tales como la variación de los sub-componentes de un componente compuesto de forma dinámica o interceptar operaciones de llamadas de entrada y salida. La propuesta de GCM (Grid Component Model) [CoreGrid 07] es una extensión del modelo de componentes fractal que se dirige específicamente a los entornos informáticos distribuidos.

Alrededor de estos modelos de componentes, desarrollo basado en componentes de aplicaciones complejas, se requiere diseñar soporte en forma de métodos, estructura y herramientas. El estudio aplica la teoría del desarrollo basado en componentes en aplicaciones Grid mayormente. Se reutiliza código legado, se envuelve software escrito a mano o generada automáticamente. Además, menciona aplicabilidad a software orientado a objetos, donde se separa las aplicaciones existentes en componentes y sobre eso, desarrollar códigos basados en componentes.

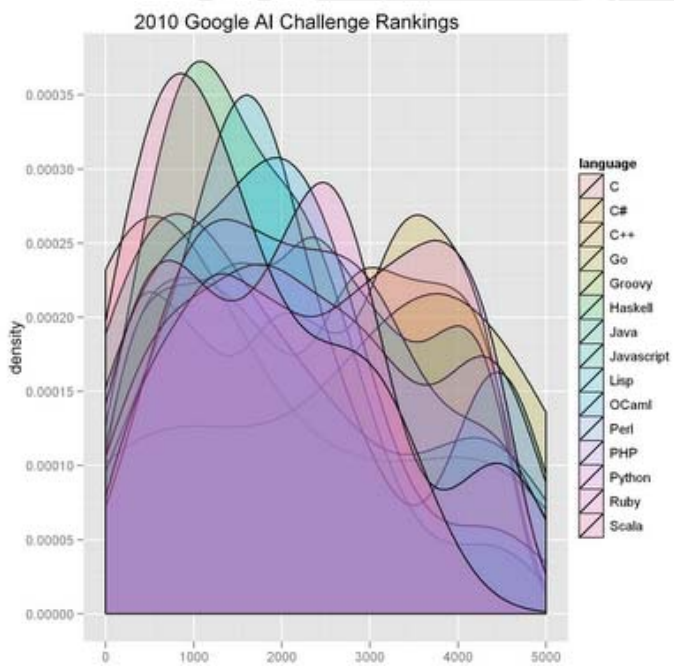
Siguiendo esta tendencia, desarrollo de software basada en componentes; investigaciones realizadas por Ronald Garcia, Jaakko, J¨arvi Andrew Lumsdaine, Jeremy Siek y Jeremiah Willcock: “A Comparative Study of Language Support for Generic Programming” [Garcia 03] y “An extended comparative study of language support for generic programming” [Garcia 07], enfocan el estudio específicamente en la programación genérica. La programación genérica facilita el desarrollo de software con muchos lenguajes de programación modernos con contenedores seguros, con tipos polimórficos y con alta reutilización de algoritmos. Esto provee un gran valor agregado; además que fomenta la construcción de plantillas librerías estándares como es el caso de C++, STL (Stepanov y Lee, 1994; ISO 1998).

1.3 Impacto de los lenguajes de programación a nivel científico

A nivel científico, en el campo de las ciencias de la computación, se realizan programas de cómputo inteligentes, definido como inteligenia artificial (IA). El término fue introducido por John McCarthy en el año 1956, el mismo creador del lenguaje de programacion funcional LISP, un lenguaje de paradigma funcional. Por muchas décadas se ha relacionado a los lenguajes de paradigma funcional con la Inteligencia Artificial.

En el año 2010, la empresa Google Inc. auspició el el programa "Planet Wars Google AI Challengue", el cual tuvo mas de 4000 entradas que utilizaron IA con teoría de juegos para la competencia. R-Chart fue utilizado para analizar los lenguajes de programacion que emplearon los concursantes con algunos resultados interesantes. [10]

Figura 1.15: Ranking de los lenguajes de programacion que se utilizaron en el Google AI Challenge.



Fuente: http://ebiquity.umbc.edu/blogger/category/general/social

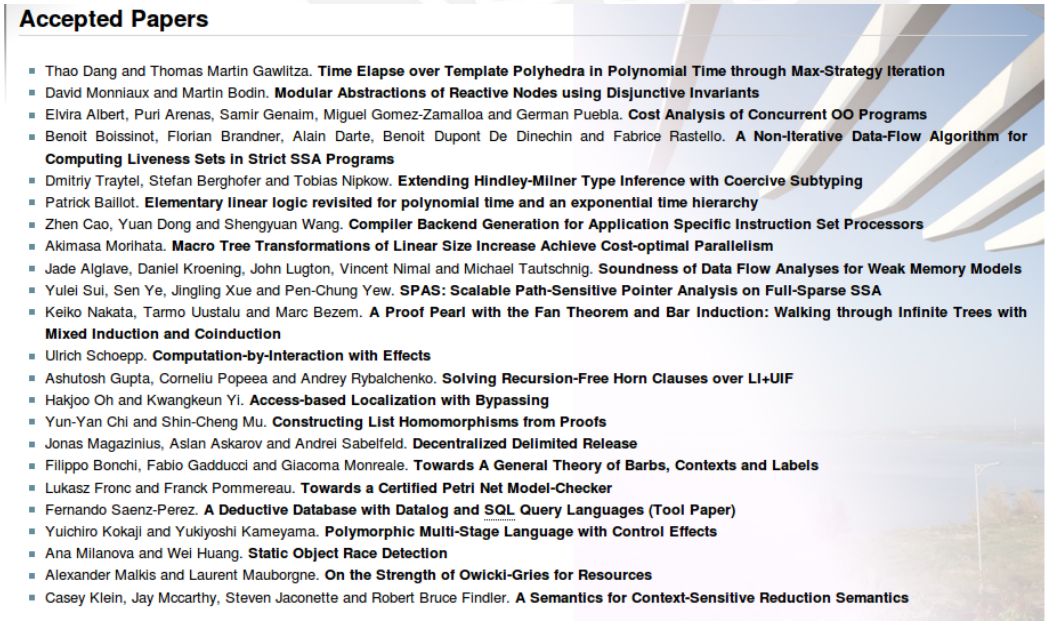
Los lenguajes de programación mas utilizados fueron Java, C++, Pyhton y C#. El ganador Hungaro Gábor Melis empleó LISP, al igual que otros 33 concursantes. La participación del lenguaje C fue menor, los 18 consursantes que lo utilizaron, lo hicieron muy bien.

[10] http://ebiquity.umbc.edu/blogger/category/general/social/

Figura 1.16: Lenguajes de programacion de los primeros lugares en el Google AI Challenge

Conferencias relacionadas a los avances de los lenguajes de programación e investigación acerca de ellos, se han expuesto en el APLAS (Asian Symposium Programming Languages and Systems) en los ultimos años. En el año 2009, se recibieron más de 56 papers o publicaciones de investigaciones a nivel mundial, acerca de los resultados y/ o experiencias concernientes a los lenguajes de programación y sistemas. En el año 2011, se aceptaron los temas que a continuación muestra la figura:

Figura 1.17: Papers o investigaciones aceptadas para la exposición en el APLAS 2011



Fuente: http://flolac.iis.sinica.edu.tw/aplas11/doku.php?id=accepted

Un estudio realizado por [HU 11] resume las cuatro investigaciones transcendentales de este congreso.

- "Weak updates and separation logic" by Gang Tan, Zhong Shao, Xinyu Feng and Hongxu Cai, donde se propone un nuevo enfoque para la combinación de un sistema de tipos con las afirmaciones lógicas separadas con el fin de construir un seguro sistema de tipos donde las referencias débiles y fuertes pueden coexistir. Las referencias débiles son aquellas cuyo tipo no debe cambiar durante la ejecución del programa, mientras que las referencias fuertes apoyan a las actualizaciones por el cambio de sus tipos.

- "A Short Cut to Optimal Sequences" by Akimasa Morihata, donde se muestra cómo usar una fusión de acceso directo a programas derivados sistemáticamente para buscar secuencias óptimas. A partir de un enfoque de enumerar-y-elegir, leyes algebraicas fueron desarrolladas de forma segura se pueden fusionar varias veces sin perder soluciones óptimas. Con esta base, se ha desarrollado una librería de dominio específico, junto con un conjunto de reglas reescritas, que se pueden aplicar automáticamente por el compilador.

- "Classical Natural Deduction for S4 Modal Logic" by Daisuke Kimura and Yoshihiko Kakutani, presentan una extensión del clásico sistema deductivo con necesidad y posibilidad en modalidad S4. El cálculo lambda es extraído del sistema lógico y propiedades como reducción y normalización fueron probadas. Además, una interpretación computacional de la posibilidad en modalidad en el contexto de la escena computacional se da cuando se permiten excepciones. Ver más detalle acerca de S4 Modal Logic en la publicación:[Chung-chieh 05]

- "On the Decidability of Subtyping with Bounded Existential Types and Implementation Constraints" by Stefan Wehr and Peter Thiemann, provee dos resultados:

- Subtipos y tipos de existencia limitada conducen a la no expresividad.
- Subtipos para los tipos existentes con límites inferior y superior son inexpresables.

Ambas pruebas muestran la no expresividad con una traducción de un problema existente ya conocido.

Se analiza ello en lenguajes orientados a objetos como Java.

Finalmente, [Henglein 10] publica un artículo en la revista ACM, con la misma crítica con la cual comenzamos la exposición de los antecedentes (desde los años 60s), acerca las imperfecciones de los lenguajes de programación. Esta vez, conceptos como desbordamiento de búfer, suplantación de identidad, inyección de ataque, reemplazo de fragmento de código por uno más eficiente, excepciones e invariabilidad de valores, aparecen en la investigación. Indica que el análisis que se emplea en la mayoría de los casos cuando queremos saber si el código del programa (la entrada al análisis) tiene una propiedad específica, debe cambiar. Se asume rutinariamente una programación Turing-completa, una propiedad de un programa con comportamiento no trivial, que requiera una respuesta exacta.

Si aún existen problemas luego de haber probado el código generado, entonces es momento de hacer algo diferente. El autor propone olvidar al menos una de las condiciones mencionadas para evitar y/o cazar los impredecibles errores de programación que son detectados luego de un buen tiempo.

Consideraciones Finales

Se puede apreciar e inferir de los diferentes estudios mencionados, que:

1. Las investigaciones citadas en el primer capítulo resaltan que, la teoría de los lenguajes de programación no se desliga a los paradigmas bajo los cuales se sustentan.
2. A nivel académico, la polémica acerca del primer lenguaje de programación a impartirse en las escuelas de ciencias de la computación, asi como el paradigma de programación a enseñarse; se presenta desde los años 60s a la actualidad.

   Se tiene un registro que el lenguaje bandera a enseñarse en las escuelas a comienzos de los años 90s en los Estados Unidos, era Pascal; para los años 2000, la tendencia fue enseñar C/C++ y Java. En Europa, estudios estadísticos encontrados en Australia y Copenhagen, señalan como lenguajes instructivos a Java, C# y .NET.

   En Latinoamerica no se ha encontrado un estudio dedicado al registro estadístico de los lenguajes de programación utilizados en los años 60 a la actualidad, en escuelas de ciencias de computación o informática.

   En el Anexo 01 A, se adjuntan los programas de las universidades y qué lenguajes de programación se enseñan en Lima, Perú. En el Anexo 01B se adjunta la imagen del portal de la universidad de Berkeley con todas las documentaciones respectivas a los cursos, temáticas, horas y fundamentos de los lenguajes de programación enseñados.

   En el anexo 01 C, se puede verificar que online que han publicado las calificaciones obtenidas en programación desde el 2002 al 2011 en EEUU en la Universidad MTF.

   Por otro lado, los miembros dirigentes de la asociación “ACM”, también se reúne para debatir la tendencia de la enseñanza, horas, temática de los cursos en los que se imparten los lenguajes de programación, y plasma su estudio en una curricula cada cuatro años. Ha lanzado a través de su página Web CSTA (Computer Science Teacher Association) un programa denominado “The Voice for K-12 computer science education and its educators”, para que profesores y alumnos tengan la oportunidad de compartir y prepararse mejor en la disciplina de las ciencias de la computación.
3. A nivel industrial, se puede visualizar que los estudios de popularidad de lenguajes de programación desde el año 2008 al año 2012, basados en las páginas Web langpop.com y tiobe.com, muestran a Java, C y C++ en los primeros lugares.

   Los proyectos que emplean mayormente el lenguaje de programacion C con gran demanda en el mercado hasta el año 2009 refieren a los sistemas embebidos.

Los proyectos para aplicaciones web y programacion para móviles empleados en su mayoría el mercado refieren al lenguaje de programacion Java.

Se muestran gráficos de los errores de programacion y del mantenimiento que se deben hacer de los proyectos de software codificados con el lenguaje C y Java. Esta gran cantidad de trabajo, hace que crezca la demanda de programadores en estos lenguajes de programacion.

Otro estudio realizado, presentado en este capítulo refiere a los sentimientos positivos de los programadores hacia los lenguajes de programacion en la actualidad. Estos estudios se basaron en los diversos canales como IRC y en redes sociales como twitter, que también guardan información acerca de los lenguajes de programacion a través de los tags. Aparecen con gran porcentaje de aceptación los lenguajes de programacion Java y Haskell.

Por otro lado, se muestran estadísticas obtenidas por repositorios de código de proyectos de software libre y código abierto, donde se demuestra que entre los lenguajes de programacion C, C++, Java, Haskell y GO, destaca con gran diferencia, la popularidad y gran uso del lenguaje de programacion Java.

Finalmente, se presenta uno de los concursos que la empresa Google promueve en estos últimos años, el Google Code Jam, en donde se puede ver que los lenguajes de programacion C, C++, Java, Haskell y GO están considerados como calificados para la competencia y todo ellos logran pasar hasta la segunda ronda. A la ronda final, lograron participar los lenguajes de programacion C++, Java y Haskell.

4. A nivel científico, se muestra en este capítulo un programa que promueve Google en el campo de la Inteligencia Artificial y se puede ver el listado de los ganadores, asicomo los lenguajes de programacion que se utilizaron en los diferentes proyectos.

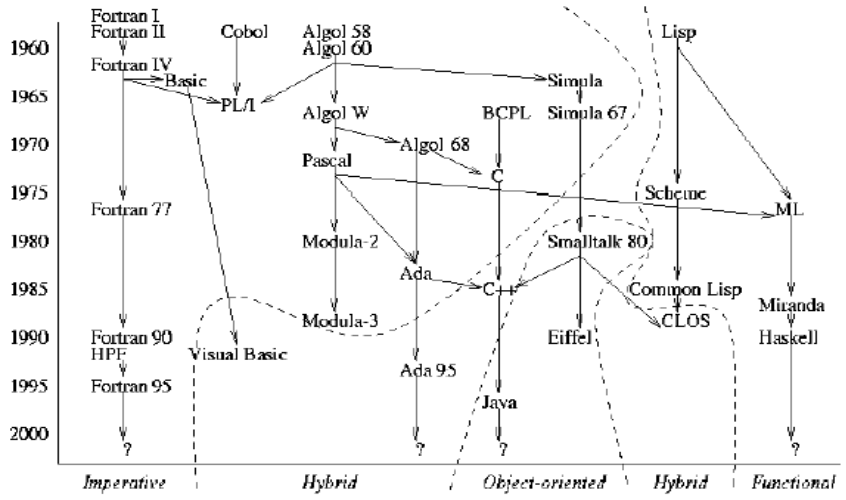
   C, C++, Java y GO fueron los lenguajes de programacion líderes que aparecieron en la lista. Esto fue una sorpresa debido a que usualmente es asociado un lenguaje funcional para poder solucionar temas de inteligencia Artficial. El ganador programo en LISP, pero no aparece en el listado el lenguaje de programacion Haskell.

   Las investigaciones que se realizan para proponer mejoras en los lenguajes de programación de la actualidad son presentadas en los diferentes congresos y conferencias. Se muestra en el capitulo el evento APLAS, y la temática que se presenta a través de las publicaciones. Esto es importante porque los estudios de curva de aprendizajes de los lenguajes de programacion son temas de interés en los últimos años. (Ver anexo 03).

CAPÍTULO II: CATEGORIZACIÓN DE LOS PARADIGMAS DE PROGRAMACIÓN

Los lenguajes de programación evolucionan y se vuelven cada vez más sofisticados, aparecen nuevos lenguajes de programación a partir de los años 60s, y tal como se puede apreciar en la Figura 2.1, se pueden diferenciar por algún "estilo", mejor definido como "paradigma de programación". Actualmente no hay un estándar para la clasificación de los mismos, pero los paradigmas de programación existentes brindan una visión general al programador acerca de la ejecución del código.

Figura 2.1: Lenguajes de programación que aparecen a lo largo del tiempo a partir de los años 60 y su diferenciación a través de los diferentes paradigmas de programación



Fuente: http://www.cs.fsu.edu/~engelen/courses/COP402001/notes1_4.pdf

Muchos estudios se han realizado con referencia a los paradigmas de programación.
A continuación, se resume los aportes de algunos investigadores:

2.1. Según Michael L. Scott

En su libro "Programming Language Pragmatics" se distinguen dos conjuntos de familias de lenguajes de programación: los lenguajes declarativos y lenguajes imperativos. Resalta que los lenguajes declarativos se basan en declarar qué es lo que la computadora hace. Lenguajes funcionales como Lisp, Scheme, ML, Miranda, Haskell;

Dataflow, y lenguajes lógicos como Prolog y Excel son ejemplos de lenguajes declarativos.

El grupo de los lenguajes imperativos se enfocan en cómo la computadora debe realizar las cosas. Lenguajes procedurales como Fortran, Pascal, Basic, C y lenguajes orientados a objetos (OO) como Smalltalk-80, C++ y Java son ejemplos de lenguajes imperativos. Se indica que los lenguajes declarativos son lenguajes de alto nivel que están más sintonizado con el punto de vista del programador; sin embargo, el desempeño es más favorable para lenguajes imperativos. Señala entonces que ahí nace una tensión en el diseño de lenguajes, entre el deseo de deshacerse de irrelevantes detalles de implementación y la necesidad de recordar suficientemente los detalles de al menos control del esquema de un algoritmo.

Además se indica que aún no está claro hasta qué punto podemos esperar que los compiladores descubran buenos algoritmos para problemas planteados en un nivel muy alto de abstracción y que si el compilador no puede encontrar un buen algoritmo, los programadores tienen que ser capaces de especificar una forma explícita.

Esta categorización de los lenguajes de programación no se considera como definitiva. Aún está abierta a debate debido a que es posible que un lenguaje de programación funcional sea considerado orientado a objetos, y/o que otros autores no consideren la programación funcional como declarativa.

A continuación se describe con mayor detalle las agrupaciones de los diferentes lenguajes de programación y la categorización planteada por el autor.

2.1.1. Lenguajes de programación declarativos

- Lenguajes de programación funcional, LISP, ML y Haskell, basados en recursión de funciones que define entradas y salidas, inspiradas en el cálculo lambda.
- Lenguajes de programación de flujo de dato, utiliza nodos que son disparados con la llegada de tokens de entrada y pueden operar concurrentemente. Id, Val y su descendiente Sisal son lenguajes de control de flujo. Este último a veces descrito como lenguaje funcional.
- Lenguajes de programación lógicos o basados en restricciones, inspirados en la lógica de predicados, buscan ciertos valores que satisfacen ciertas especificaciones usando una lista de reglas lógicas: Prolog, SQL database, XSLT script y programación de aspectos Excel.

2.1.2. Lenguajes de programación imperativos

- Lenguajes de programación basados en la arquitectura Von Neumann, los más familiares y exitosos: Fortran, Ada 83, C; que implican modificación de variables. Mientras los lenguajes funcionales se basan en expresiones, los lenguajes Von Neumann se basan en asignaciones con efecto colateral (side-effect).

2.2. Según David A. Watt

En su libro "Conceptos de Diseño de Lenguaje de Programación", le da mayor importancia al paradigma orientado a objetos (OOP), brinda ejemplos de aplicaciones y usos de los lenguajes C, Java, Haskell y otros. Además de sus características, explica su motivación y percepción ante cada uno de estos lenguajes de programación. Aquí, un resumen de los paradigmas que considera:

2.2.1. Lenguaje de programación imperativa

- Imperativo es una palabra que proviene del latín "imperare", el cual significa tpor ende está basado en comandos, los cuales permiten actualizar valor de variables y los almacena en con una simple abstracción de acceso a memoria a través de instrucciones, es eficiente debido a que tiene una estrecha relación con la arquitectura de la computadora. Es el paradigma líder desde los 50's hasta los 90's, que aparecen los lenguajes OOP. Entidades del mundo real pueden ser modeladas por variables y los procesos por comandos que inspeccionen y actualicen estas variables. Para lograr ello, es permitido en los lenguajes imperativos realizar cálculos recursivos o iterativos, en este último se utiliza variables que sostengan resultados intermedios para controlar los cálculos y acumular el producto. Se puede evitar el uso de variables intermedias, con el empleo de un rico repertorio de expresiones condicionales e iterativas como en las funciones recursivas. En la práctica C y ADA no son tan ricos al respecto.
- Un concepto clave en los lenguajes imperativos son los procedimientos porque abstraen sobre los comandos, y se relaciona el comportamiento del procedimiento con la intención del algoritmo aplicado. Ello ha sido determinado por el implementador del lenguaje pensando en cómo usuario del lenguaje abstrae las entidades del mundo real. La abstracción de datos no es soportado por lenguajes imperativos clásicos como C y Pascal, pero si en lenguajes imperativos modernos como ADA porque la observación del comportamiento de las operaciones de los tipos abstractos y los algoritmos aplicados han sido una preocupación del implementador del lenguaje pensando nuevamente en el enfoque del usuario del lenguaje al programar. La

arquitectura de un programa influye en el código, las unidades de un programa en las que se descompone junto con la relación de estas unidades. Esto es importante en la ingeniería de software porque afecta costo de implementación y mantenimiento del programa. La medida de calidad sería entonces mantener un lazo no tan estrecho entre unidades para evitar modificaciones entre unidades y así el costo sería menor.

2.2.2. Lenguaje de programación orientada a objetos (OOP)

- Este paradigma al igual que los lenguajes modernos del paradigma imperativo, utiliza abstracción de datos o clases en el diseño de sus programas; pero al ahondar más en las clases y aplicar jerarquías en ellas es entonces que se desliga la diferencia entre uno y otro paradigma. Un objeto tiene uno o más componentes que en este caso se determinan como variables y son equipadas como métodos que operan sobre él. Las variables componentes de los objetos son privados y se acceden a ellas por los métodos. El objeto representa la entidad del mundo real o del mundo virtual, que deben relacionarse entre sí para no cambiar independientemente y generar problemas de inconsistencia de datos.
- La unidad del OOP es la clase, que a su vez encierra una familia de objetos que tienen propiedades y métodos similares en su composición. Las clases pueden ser dependientes entre sí por inclusión o extensión y herencia entre clases, otro concepto en OOP, tiene lugar porque una clase (subclase) puede heredar métodos de otra (superclase), a menos que se especifique lo contrario. Ello hace que el programador tenga mayor productividad y se debe tener cuidado con las modificaciones que se realizan en una implementación de software, ya que un cambio puede afectar a muchas subclases y/o partes del programa. La programación OO contiene subclases, el lenguaje OOP puro contiene sólo clases, entonces los procedimiento individuales en un lenguaje imperativo se convierten en los métodos de una clase en un programa OO. Finalmente, en la programación OO también se incluye el concepto de polimorfismo que permite que un objeto de una subclase sea tratado como objeto de una superclase, se puede construir un objeto heterogéneo de diferentes clases, en este caso, se deben tener clases antecesoras en común.
- Los lenguajes OO predominaron en los 90´s porque los objetos permiten abstraer entidades de la realidad de una manera más natural; además, las clases y sus jerarquías, son unidades reutilizables, convenientes para programas largos.

2.2.3. Lenguaje de programación concurrente

La programación concurrente permite la ejecución de comandos para que se puede superponer por un intercalado arbitrario: multiprogramación o por una simultanea ejecución en muchos CPUs: multiproceso. La palabra concurrente tiene origen latín y significa “corriendo juntos” y su concepto esta relacionado a:

- La ejecución paralela de uno o más procesos, que opuestamente a la secuencia de comandos que ofrece la programación imperativa, se altera el orden y puede traer consecuencias de inconsistencia de data o falta de determinismo.
- La sincronización, permite que el programador pueda manejar el desorden de los procesos ante la estrecha conexión creada, pero aún se pueden presentar esporádicas fallas y los errores pueden ser irreproducibles.
- Acceso sincronizado para compartir datos mediante la exclusión mutua de inter procesos, que puede restaurar la semántica de las variables accedidas y actualizadas; y con la comunicación se completaría la interacción entre procesos ya sea en sistemas distribuidos o centralizados. Java usa hilos a nivel superclases para aplicar concurrencia y ADA 95 logra sincronización usando tipos protegidos con tipos tags[11].
- Transferencia sincronizada de data mediante la comunicación de interprocesos.

2.2.4. Lenguaje de programación funcionales

Indica que los lenguajes funcionales no usan comandos o procedimientos como los lenguajes imperativos y orientado a objetos, en lugar de eso se explota otros conceptos como uso de funciones. La unidad de un programa funcional son las funciones, una función que llama a otra, pudiendo ser la misma, en ese caso se está aplicando el concepto de recursividad.

Se usan funciones de orden superior, concepto que permite que las funciones sean parámetros de entradas y salida de otras funciones. El polimorfismo, es otro concepto del paradigma funcional que permite que los parámetros en las funciones sean de varios tipos en una sola definición.

Finalmente, emplea evaluación perezosa, que procesa las operaciones sólo en caso de ser necesario, si no son llamadas para realizar otros cálculos, no se llegan a procesar. Los programas en este paradigma comprenden muchas funciones y manejarlas puede ser algo complicado, pero en la práctica se pueden agrupar en paquetes. Además acota

[11] Tag o etiqueta (metadato), una palabra clave o término asociado con o asignado a una pieza de información.

que lenguajes como ML y Haskell agregan tipos abstractos como ventaja ante otros paradigmas, que computa nuevos valores a partir de antiguos, permitiendo expresividad en código.

2.2.5. Lenguaje de programación lógica

Aunque diferentes, los lenguajes de programación funcionales e imperativos tienen en común, la lectura de las entradas, la escritura de la salida y se puede ver como se mapea la salida en la implementación. En cambio los lenguajes de programación lógica manejan relaciones (dados los conjuntos S y T, se tiene un 'a' que pertenece a S y un 'u' que pertenece a T), se puede procesar muchas entradas para varias salidas, dado que la relación puede ser unaria, binaria, ternaria, etc. En estas relaciones deben cumplirse las aserciones de cada uno de los elementos de un conjunto, las colecciones de las cláusulas de Horn y el backtracking para garantizar fidedignos resultados. Las estructuras de la cláusula de Horn fuerzan a que el cálculo sea más simple y expresable en un lenguaje lógico, mientras que en los lenguajes imperativos y funcionales este mismo procedimiento se hace complejo y difícil. Además, los lenguajes lógicos como PROLOG poseen un mapeo potencialmente superior a los otros paradigmas porque puede no hacer diferencia entre las entradas y las salidas.

2.2.6. Lenguaje de programación con scripts.

Script es usado en muchas aplicaciones y controla muchos subsistemas que pueden estar escritos en muchos lenguajes pertenecientes a muchos paradigmas e incluso mismos scripts. Los scripts se caracterizan por su rápido desarrollo y evolución, porque es ligero el proceso de edición – ciclo de corrida del programa, cosa que no ocurre con los lenguajes imperativos y OO, que emplean tiempos de edición-compilación-enlazamiento-ciclo de corrida. En otros lenguajes scripts incluyendo Phyton, el código fuente es automáticamente compilado en código de máquina virtual, que luego es interpretado, compilado y enlazado, y esto no es visible al programador. Se sacrifica eficiencia y velocidad en la ejecución por los costos de interpretación y verificación de tipos. Es así que los scripts ofrecen un alto nivel de procesamiento de cadenas, las mismas que son ubicuas a nivel web, provienen de mails, consultas de bases de datos y resultados de los mismos, documentación XML, HTML; para analizar la estructura interna de las cadenas utilizan lo que llaman expresión regular (utilización de patrones y verificación de coincidencias). Muchos scripts proveen soporte de alto nivel para gráficos, y es que mantienen un vago emparejamiento entre el código y la GUI, es muy útil y

productivo ya que por ejemplo en TCL basta una línea de código para crear un botón, o para cambiar la apariencia de una etiqueta. Finalmente los lenguajes scripts enfrentan un reto con el sistema de tipos dinámicos, ya que si aplican un tipo polimorfismo paramétrico que permita reconocer los diferentes tipos que provienen de los formularios web, y otras fuentes de diferentes lenguajes de programación; corre el riesgo de no ser más de desarrollo rápido y evolutivo como indica el paradigma. Otra alternativa es declarar sin tipos las variables pero también hay otro riesgo por ese lado, ya que no permite legibilidad y promueve errores que muchas veces no son detectados e incluso pasan las pruebas de código sin problemas.

2.3. Según Robert W. Sebesta

En su libro “Concepts of Programming Languages”, indica que los lenguajes de programación son categorizados usualmente como imperativos, funcionales, lógicos y orientados a objetos. Sin embargo, no considera lenguajes que soportan programación orientada a objetos como parte de una categoría separada. Describe en su libro cómo los lenguajes más populares que soportan programación orientada a objetos surgen de los lenguajes de programación imperativos. Sin embargo, el paradigma de desarrollo de software orientado a objetos difiere grandemente del paradigma orientado a procedimientos utilizado usualmente por los lenguajes imperativos, las extensiones a un lenguaje imperativo requeridos para soportar programación orientada a objetos no son muy contundentes. Por ejemplo, las expresiones, las sentencias de asignación y de control en C y Java son muy parecidas. Por otro lado, los arreglos, subprogramas y semántica de Java son muy diferentes de las de C.

Sentencias similares pueden ser hechas para lenguajes funcionales que soporten programación orientada a objetos. Otro tipo de lenguaje son los lenguajes visuales que son una subcategoría de los lenguajes imperativos. El lenguaje visual más popular es Visual BASIC. NET (VB.NET) (Deitel et.al. 2002)

Estos lenguajes, o sus implementaciones incluyen capacidades de la generación “arrastrar y soltar” segmentos de código, llamados alguna vez lenguajes de cuarta generación, proveen una simple interfaz de usuario grafica a programas. Por ejemplo, en VB.NET el código para producir un formulario de control como el botón o el cuadro de texto, puede ser creado con una simple combinación de teclas. Estas capacidades están disponibles en lenguajes .NET.

Algunos autores refieren a los lenguajes scripts[12] como una categoría de lenguaje de programación separada. Sin embargo, los lenguajes de esta categoría están más unidos por su método de implementación, interpretación parcial o completa, que un diseño de lenguaje común. Los lenguajes que son típicamente llamados scripts son Perl, JavaScript y Ruby, son lenguajes imperativos en cierta manera.

Un lenguaje de programación lógico es un ejemplo de un lenguaje basado en reglas. En un lenguaje imperativo, un algoritmo es especificado en gran detalle, y la especificación del orden de ejecución de las instrucciones o sentencias deben ser incluidas. En un lenguaje basado en reglas sin embargo, las reglas no son especificadas siguiendo un orden, la implementación del sistema debe darse de manera que al ordenar las reglas se brinde el resultado esperado. Esta aproximación de desarrollo de software es completamente diferente a la programación imperativa, orientada a objetos y funcional.

En los últimos años, una nueva categoría de lenguajes han surgido, son los lenguajes híbridos markup. XHTML es un lenguaje markup comúnmente utilizado para la capa de información en documentos Web.

Otros lenguajes como JSTL – Java Server Pages Standard Tag Library y el XSTL – eXtensible Stylesheet Transformation Language no pueden ser comparados con ningún lenguaje de programación completo, porque según el autor, así como los RPG – Report Program Generator, son sólo utilizados para hacer reportes en negocios. Al igual que otras herramientas como el APT – Automatically Programmed Tools y los GPSS- General Purpose Simulation System, que son utilizados para programar en maquinarias y para simulación, respectivamente.

2.4. Según John C. Mitchell

En su libro “Conceptos de Lenguajes de Programación” incluye un estudio de programación multiparadigma, el cual puede soportar distintos paradigmas de programación con el objetivo de que un programador utilice el más conveniente a la hora de resolver un problema. Los paradigmas actuales que se basan en muchos de los paradigmas tradicionales, como son el paradigma imperativo, orientado a objetos, funcional y procedural.

[12] Los lenguajes scripts son lenguajes interpretados antes que compilados y son convertidos en permanentes archivos antes de ser ejecutados, y son utilizados mayormente para la construcción de sitios Web.

2.4.1. Paradigma orientado a eventos

La programación dirigida por eventos es un paradigma de programación donde la estructura y la ejecución de los programas están determinadas por los sucesos (eventos) que ocurran en el sistema o que ellos mismos se provoquen. Se caracteriza por el empleo de elementos visuales.

2.4.2. Paradigma orientado a servicios (POS)

En la POS los componentes se publican y usan los servicios [D5] de una manera par-a-par. Un cliente no se ata a un servidor particular, los proveedores de servicio son intercambiables. Utiliza una Arquitectura Orientada a Servicios (Service Oriented Architecture o SOA), que define la utilización de servicios para dar soporte a los requerimientos de software del usuario, provee un marco de trabajo para el desarrollo de software como de implantación, es un juego de servicios residentes en Internet o en una intranet, usando servicios web.

Cada compañía ha conceptualizado el modelo orientado a servicios acorde a sus propias iniciativas de tecnología: Microsoft .NET™, Hewlett Packard Cooltown™, Sun Microsystems Java™ / Jini™, y Openwings™.

2.4.3. Programación orientada a agentes

Aplicaciones informáticas con capacidad para decidir cómo deben actuar para alcanzar sus objetivos. Se define un objeto cuyos métodos representan sus habilidades, y las variables de instancia su estado mental. De esta forma, un objeto encapsulará en sus métodos capacidades de comportamiento de un agente y su conocimiento estará dado por el estado de sus variables de instancia. Los agentes operan sin la intervención directa de personas u otros, pueden interactuar con otros agentes, perciben el entorno o ambiente y responde rápidamente a cambios que ocurren en dicho entorno, no actúan simplemente en respuesta a su entorno, sino que son capaces de exhibir comportamiento dirigido hacia el objetivo, tomando la iniciativa, tienen la habilidad para trasladarse a través de una red o Internet, no comunicará información falsa a propósito, estará dispuesto a ayudar a otros agentes si esto no entra en conflicto con sus propios objetivos o metas, actúa de forma racional, intentando cumplir sus objetivos, si estos son viables, y no actuará de manera que impida que se cumplan. Metodologías orientadas a agentes: Aglets, JADE, MASE, Gaia, CoMoMAS, MAS-CommonKADS.”

Paradigmas a evaluar

Se aprecia a lo largo de la historia de los lenguajes de programación, y a través de los diferentes estudios acerca de la categorización de los lenguajes de programación que, éstos nacen, crecen, se conmutan y mueren. Además se puede identificar paradigmas base, como los casos de los paradigmas imperativos, orientados a objetos y funcionales. Sobre dichos paradigmas, se han ido diseñando nuevos paradigmas, y en otros casos se ha seguido un lineamiento híbrido, al tomar y reunir características de cada uno de ellos.

[Budd 89], PhD en Ciencias de la computación, recomienda que en una investigación acerca de los paradigmas de programación, se demuestre la complementariedad de los diferentes aspectos de los diferentes paradigmas, y no solamente enfocar los aspectos competitivos entre ellos.

Teniendo en cuenta lo expuesto, a continuación se expone a detalle los paradigmas: imperativo, orientado a objetos, funcional y multiparadigma.

a) Paradigma de programación imperativa

El paradigma de programación imperativo se adecua al punto de vista del computador. El énfasis se da en la ubicación de variables, sus valores y las operaciones legales que se pueden realizar en estas ubicaciones. Ubicaciones que pueden ser agrupadas y de ejecución estructurada. (Secuencias de operaciones para lograr un cálculo).

Según [Budd 89], este paradigma de programación exhibe las siguientes características:

- "Word-at-a-time" o "Palabra-en-un-tiempo" el procesamiento de cálculo consiste en muchos movimientos individuales y cálculos de elementos de datos pequeños.
- Programación de los side-effects o efectos secundarios porque cambia el "estado" de la máquina a través de la asignación.
- Orientado a lenguaje de máquina y lenguaje estructurado, con estructuras tanto de datos y de control que están muy cerca de la arquitectura de máquina subyacente.

[Petre 90] indica que la cercanía del lenguaje al modelo de la máquina es reflejada de manera práctica, en el buen control ofrecida por muchos lenguajes imperativos sobre la máquina en aspectos tales como la asignación de memoria y manejo de entrada/salida.

Sin embargo, la fortaleza de esta correspondencia significa incorporar restricciones basadas en hardware, y es ahí donde la legibilidad de las expresiones en el código es afectada, se deben definir límites y características que cambien en tiempo de ejecución.

Si bien es cierto, los lenguajes de programación imperativos como FORTRAN, ALGOL, COBOL, BASIC, B, C, Modula y Ada, son lenguajes de programación de nivel medio a alto, porque utilizan estructuras de control como bucles en su sintaxis, y mejoraron características como portabilidad y simplificaron el lenguaje de máquina como el lenguaje ensamblador[13]; aún el programador debe describir los procedimientos (how-to-do) para que el programa ejecute lo que debe hacer, haciendo uso de punteros e índices en arreglos. Estos conceptos a la fecha originan bugs o errores de programación.

Cuando se programa utilizando el paradigma imperativo, usualmente se definen las estructuras de datos que representarán las entidades relacionadas con la realidad en el programa y para ello, se escriben funciones. Si las representaciones son similares en tipo y en propiedad, la mayoría de las funciones serán aplicables de manera similar. Este es un escenario ideal, si es que se tiene un programa de tamaño razonable. Lo contrario ocurre cuando se cambia una propiedad, entonces es necesario cambiar todas las estructuras. Otra manera de programar es definir una estructura grande que contiene todo y se escriben funciones para ello. Esta manera es ineficiente, porque las funciones se llenan de estructuras selectivas; ello hace el programa muy difícil de depurar y mantener.

Un esquema para saber cómo se relaciona y actúan las unidades de un programa tradicional en un lenguaje imperativo se da en el libro de David Watt con el “Corrector ortográfico”. Se tienen procedimientos y variables globales, a las cuales se acceden mediante estos procedimientos. Cuando los procedimientos se llaman entre sí, se crea una conexión no tan estrecha y al modificar la implementación de un procedimiento es poco probable que afecte a otros procedimientos enlazados; sin embargo, cuando se accede a las variables globales se crea una conexión estrecha y cualquier modificación a la representación de una variable global puede forzar modificaciones de otros procedimientos, que tengan acceso a ellas.

La sensibilidad se da en la forma en que la variable global se inicializa, se inspecciona y se actualiza por otros procedimientos. Programas más grandes con arquitecturas similares son más difíciles y costosos de mantener, el programador puede no comprender el procedimiento individual, así como el rol de cada variable global; cualquier modificación

[13] Lenguaje ensamblador: lenguaje de bajo nivel para computadoras y otros dispositivos programables. Ver más: http://www.arl.wustl.edu/~lockwood/class/cs306/books/artofasm/toc.html

de procedimiento podría desencadenar una cascada de modificaciones a otros procedimientos. Por ello, la abstracción de datos se ha convertido en un concepto clave en los lenguajes y por ello, la programación orientada a objetos es más utilizada y segura.

b) Paradigma de programación orientada a objetos

Siguiendo el estudio de [Budd 89], la programación orientada a objetos es muy popular y ofrece distintiva aproximación a la realidad para resolver problemas. Los programas modelan aspectos del mundo real: objetos con propiedades. El paradigma de programación orientada a objetos exhibe las siguientes características:

- Interacción de agentes- los objetos se encapsulan y por ello se puede definir y leer sus estados y comportamiento. Los objetos interactúan vía paso de mensajes.
- Organización jerárquica- los objetos son clasificados jerárquicamente de acuerdo a las similitudes en sus comportamientos vía estructuras llamadas clases, y pueden ser sucesivamente redefinidos vía estructuras llamadas subclases, permitiendo a su vez herencia de objetos.
- Respuesta basada en atributos- los objetos interpretan mensajes basados en sus atributos: ambas propiedades particulares (instancia de variables) y propiedades generales (clase).

Ejemplos del paradigma orientado a objetos incluyen lenguajes Smalltalk y C++. C++ llegó a ser muy importante en el plan curricular, desplazando a Pascal como alternativa de lenguaje introductorio. [Budd 89]

C++ también llega a ser muy importante en la industria, y este entusiasmo hacia C++ por la industria hace que en cierto punto, se imparta la enseñanza de dicho lenguaje de manera imperativa en el plan curricular. ¿Pero es C++ la respuesta, es su paradigma el que debe ser impartido en las clases de pregrado? En la sección titulada "Living in a Multi-Paradigm Universe" del popular libro C++ Primer Lippman advierte:

La programación orientada a objetos está en avance evolutivo en el diseño y manejo de largos sistemas y la gestión de grandes sistemas de software. No se trata sin embargo, de deus exmachina[14] que viene a resolver las heridas del software de la industria. Buen código es difícil de producir bajo cualquier paradigma y, una vez en producción, errores de programación continúan siendo descubiertos. Esto es poco probable que cambie por algún tiempo todavía.

[14] Deus ex machina es un término latino se refiere a una solución externa o la fuerza como medio para resolver un complot.

El estudio de [Budd 89] concluye que es evidente que la industria del software puede experimentar cambios futuros con respecto a paradigmas, y debemos preparar a los estudiantes para tales cambios.

c) Paradigma de programación funcional

Mientras que la programación orientada a objetos direcciona el comportamiento de los objetos en el tiempo, la programación funcional se concentra en relaciones matemáticas sin tomar en cuenta el tiempo, para ello utiliza funciones, las cuales también pueden ser tomados como valores de entrada a otras funciones.

Según [Budd 89], presenta las siguientes características:

- Funciones"Primera clase"- funciones son asignables, transferibles y retornan valores.
- Composición funcional- un conjunto de funciones puede ser agrupados vía funciones de orden superior; es decir, una función puede tomar otras funciones como argumentos. Cuando una función se invoca a sí misma, hablamos de recursividad.
- Transparencia referencial- funciones cuyos valores pueden ser determinados solamente por valores de sus argumentos (funciones puras).

Ejemplo: Haskell y ML. [Pou 94], indica que C++ está aún lejos de ser desplazado por algún lenguaje de programación funcional; pero a su vez los programadores se han dado cuenta que los lenguajes de programación OO tampoco son verdaderamente perfectos, es así que propone un espacio compartido, una aproximación de híbrido. Idea también expuesta por [Mac 87] "es muy bueno que la programación orientada a funciones y la programación OO sean complementarias en lugar de competitivas".

[O'Sullivan 98] ratifica estas características y mencionas otras, tales como la evaluación perezosa, el polimorfismo paramétrico y la inferencia de tipos. La evaluación perezosa se refiere al método de evaluación (la forma en que se calculan las expresiones), se calcula una expresión (parcial), y solamente si es necesario y luego se calculan los demás valores para el resultado; esto permite tener listas infinitas. En lenguajes imperativos, que usan evaluación voraz, esto no es posible.
El polimorfismo paramétrico se refiere a un código que es escrito sin mención de ningún tipo específico y ser usado con nuevos tipos primitivos, sean estos cadenas de caracteres, de palabras o números enteros, flotantes, etcétera. La inferencia de tipos se refiere a que sin necesidad de declarar los tipos en las variables o en los resultados, se

pueden saber que el sistema de datos trabaja asignando el tipo de datos a cada valor o resultado en base a las operaciones realizadas.

CAPITULO III: EVALUACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Según se ha evaluado en los capítulos anteriores, los lenguajes más populares en el mercado de la industria del software, son los lenguajes de programación Java y C/C++; en el ámbito académico y en el científico, destaca el lenguaje de programación Haskell y GO es un nuevo lenguaje de programacion creado y patrociando por Google Inc. Estos lenguajes de programacion "representativos" pertenecen a los paradigmas: imperativo, orientado a objetos, funcional y multiparadigma, según el orden de mención. A continuación, se presenta a detalle el origen, propósito y evolución de cada uno de ellos:

3.1. El lenguaje de programación C

3.1.1. Origen, propósito y evolución

Es un lenguaje de paradigma imperativo, fue inventado por Dennis Ritchie a principios de los 70's en los laboratorios Bell, AT&T en una DEC PDP-11 que utilizaba el sistema operativo Unix. Como se puede apreciar de la figura 3.1, el lenguaje C tiene como antecesor a los lenguajes BCPL, que fue desarrollado por Martin Richards, que a su vez permitió el desarrollo del lenguaje B desarrollado por Ken Thompson, que permitió a su vez el desarrollo de C. En 1983, un comité estableció la creación del ANSI[15] para definir el lenguaje C, que finalmente adoptó en Diciembre 1989 el ISO[16], para finalmente lograr el ANSI/ISO, y es comúnmente llamada c89. Finalmente se desarrolla c99, que sigue conservando casi todas las características de c89, pero se mejoran dos áreas: la incorporación de varias librerías numéricas y el desarrollo de algunos usos especiales, pero innovadoras nuevas características, como arreglos de longitud variable y la restricción de puntero calificador. Estas innovaciones volvieron a colocar a C a la vanguardia del desarrollo informático de lenguajes de programación. [Schildt 00]

Este lenguaje de programación marcó un hito en los años 70 a nivel de lenguajes de programación entre ensamblador y otros lenguajes de alto nivel de la época como BASIC, FORTRAN y ALGOL, contiene la mayoría de las características que debería tener un

[15] American National Standards Institute
[16] International Standards Organization

lenguaje de programación y popularidad se extendió por muchas organizaciones debido a la creación del núcleo del sistema operativo UNIX, compiladores y variedad de funciones agregadas a sus librerías que siguen vigentes.

3.1.2. Características resaltantes de su diseño

Siguiendo el estudio de [Schildt 00] acerca del lenguaje de programación, se tiene:

- C es un lenguaje de nivel medio, porque combina elementos de lenguajes de nivel alto como la definición de tipos de datos y sus operaciones, conversión de tipos, con el control, eficiencia y flexibilidad que ofrece el lenguaje ensamblador.

  Como lenguaje de nivel medio, permite la manipulación de elementos básicos con los que funciona la computadora como bits, bytes y direcciones.

- C es portable, se adapta de un tipo de computadora/sistema operativo a otro diferente.

- C casi no especifica comprobación de errores en tiempo de ejecución; por ejemplo, no asegura los límites de los índices de una matriz o arreglo. Tampoco es fuertemente tipificado, no es estricto con la compatibilidad de un parámetro con un argumento. C permite que un argumento sea de cualquier tipo siempre y cuando pueda ser razonablemente convertido en el tipo de parámetro; más aún, C proporciona todas las conversiones automáticas para lograr esto.

- C89 cuenta con 32 palabras reservadas y c99 agregó sólo cinco más a ellas.

- C es un lenguaje estructurado, y no un lenguaje estructurado en bloques porque no permite la creación de funciones dentro de otras funciones. Una característica distintiva de un lenguaje estructurado es la compartimentación de código y datos. Esta es la capacidad de dividir en secciones y ocultar del resto del programa toda la información necesaria para realizar tareas específicas, a través de las subrutinas y el uso de variables locales (temporales) y evitar así, efectos secundarios que puedan afectar al resto del programa, ya que si se utilizan variables globales se corre el riesgo de tener errores de programación. Además, el uso de subrutinas hace que se pueda compartir fácilmente las funciones sin necesidad de saber cómo lo hace, sólo utilizarlas y llamarlas porque se sabe qué es lo que hacen. Otra manera de estructurar el código es a través de los bloques de código, la conexión de un grupo de sentencias tratados como una unidad segmentados con símbolos como las llaves

dentro de un if[17] por ejemplo. Ello permite al programador conceptualizar mejor la naturaleza de un algoritmo al ser programado.

- C es un lenguaje para programadores reconocido a nivel mundial, brinda al programador pocas restricciones, pocas quejas, bloques estructurados, funciones independientes y un compacto grupo de palabras reservadas. Mejoró el desorden y portabilidad existente de lenguaje ensamblador con respecto a estructuras, lectura y mantenimiento, y a su vez utilizó estructuras de lenguajes como Pascal y/o Modula-2.

- C es un lenguaje de propósito general, que comenzó a ser utilizado para programar editores, compiladores y enlazadores, por su eficiencia y portabilidad fue creciendo en popularidad y aún se mantiene en el mercado a pesar de la aparición de C++. Las razones de su existencia y sobrevivencia está en los sistemas embebidos los cuales no requieren de las bondades de un lenguaje orientado a objetos.

  Además existen miles de aplicaciones en el mercado desarrollados con C, los cuales requieren de un mantenimiento. La evolución de C con c99 indica además que la comunidad C sigue trabajando y está a la vanguardia de las necesidades de los programadores.

- C fue diseñado para ser compilado, aunque también se han escrito intérpretes de C y están disponibles en algunos ambientes (asistente de depuración o ambiente de prueba). Está diseñado para leer el programa entero y convertirlo en código objeto, también llamado código binario o código de máquina, el cual puede ser ejecutado directamente por la computadora. Una vez que el programa es compilado, una línea del código fuente no es tan significativa si se refiere al tiempo de ejecución del programa.

- En C, las mayúsculas y las minúsculas son diferentes en la definición de identificadores, y también se aplica a las palabras reservadas, por ejemplo ELSE es diferente de la palabra reservada “else”.

- Los programas en C consisten en una o más funciones, la regla indica que la función que debe presentarse como obligatoria es la función “main”, porque es la primera función que es llamada en el momento de ejecución.

- Las librerías que utiliza C permite que se pueda desarrollar programas que interactúen con dispositivos E/S[18], cálculos matemáticos de alto nivel, y manejo de cadenas de caracteres. La mayoría de compiladores incluyen estas librerías

[17] If es una palabra reservada del lenguaje C que se aplica para programar situaciones condicionales
[18] E/S son las iniciales que representan entrada y salida

estándares pero no siempre están todas las funciones, ejemplo: librerías graficas. Además, se incluye un "enlazador" que combina el código fuente con el código que se encuentra en la librería estándar cada vez que se llame a una.

- Las funciones en las librerías están en formato reubicables; es decir, las direcciones de memoria para las diferentes instrucciones de código máquina no han sido definidas por completo, sólo información sobre el desplazamiento se ha mantenido y sirve de guía para crear las direcciones de memorias actuales. Se puede utilizar el código prefabricado de las librerías y si se requiere de una función una y otra vez, se puede incluir en las librerías también.

- La compilación separada es permitida cuando el programa en C está escrito en más de dos archivos. Los archivos son compilados por separado y luego enlazados con las librerías correspondientes para formar el código objeto. Si un archivo es modificado, no es necesario recompilar todo el proyecto. Se ahorra mucho tiempo.

- Para crear una forma ejecutable en C, se requieren tres pasos: crear el programa, compilar el programa, enlazar el programa con la función que se requiere de una librería indicada. Algunos compiladores incluyen ambiente de programación con editor, y para versiones de compiladores independientes se separa el editor del ambiente de programación donde sólo se acepta entrada estándar de archivos de texto y no aceptará palabras procesadas en otros procesadores de texto porque contienen códigos de control y caracteres no imprimibles. Los detalles de la compilación en cada máquina puede ser leída en la respectiva documentación.

  Los programas de C crean y utilizan cuatro regiones lógicas de memoria que soportan la ejecución de un código. La primera región de memoria es la que soporta el código del ejecutable del programa. La siguiente región es la memoria donde se guardan las variables globales. Las otras dos regiones son la pila (stack) y el montículo (heap), la pila es usada mientras el programa se ejecuta y almacena la dirección que se retorna cuando se llama a una función, guarda variables locales y el estado actual del CPU. El heap es una región de memoria libre que el programa puede usar a través de las asignaciones de memoria dinámicas. En la figura 3.2 se muestra conceptualmente cómo el programa en C aparece en memoria.

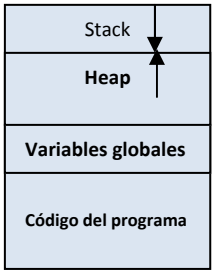


Fig. 3.1.

Mapa de la memoria conceptualizada de un programa en C

- Finalmente [Schildt 00] resalta C como subconjunto de la colección de C++, asimismo C++ es un superconjunto de C. Se puede compilar un programa de C con un compilador de C++, siempre teniendo en cuenta la extensión .c en lugar de .cpp.

Un estudio de la Universidad de California publicado el 30 de Mayo del 2011 denominado "Razones para usar C en la actualidad"[19], expone diez razones por las que se debe aprender y enseñar el lenguaje de programación C. Es un lenguaje fundamental para la tecnología moderna y ciencias de la computación, aún vigente y utilizado tanto en la academia como en la industria para sistemas embebidos y mecatrónica con interfaces de hardware. Una publicación en el Sitio Web de la empresa Microsoft, indica que C es un lenguaje modelo para los demás lenguajes de programación y ha permitido desarrollar la capacidad de los programadores.[20] Una demostración de ello, es el desarrollo de los juegos 2D y 3D, y entre las últimas mejoras del lenguaje por parte de Microsoft, se anunció el 15 de Junio del 2011, agregar capacidades de cómputo en paralelo masivo para el nuevo compilador de C++, tecnología C++AMP: Accerelated Massive Parallelism será construída sobre plataforma DirectX en Windows como parte de Visual Studio 2012.

A nivel código abierto al público en general, C y C++ son muy utilizados. Existe una Web que publica los proyectos ISO IEC[21] donde se puede visualizar cada una de las mejoras que se plantean para este lenguaje de programación, se puede tener acceso también a la documentación oficial, cada una de sus revisiones y versiones.[22]

3.2. El lenguaje de programación Java

3.2.1. Origen, propósito y evolución

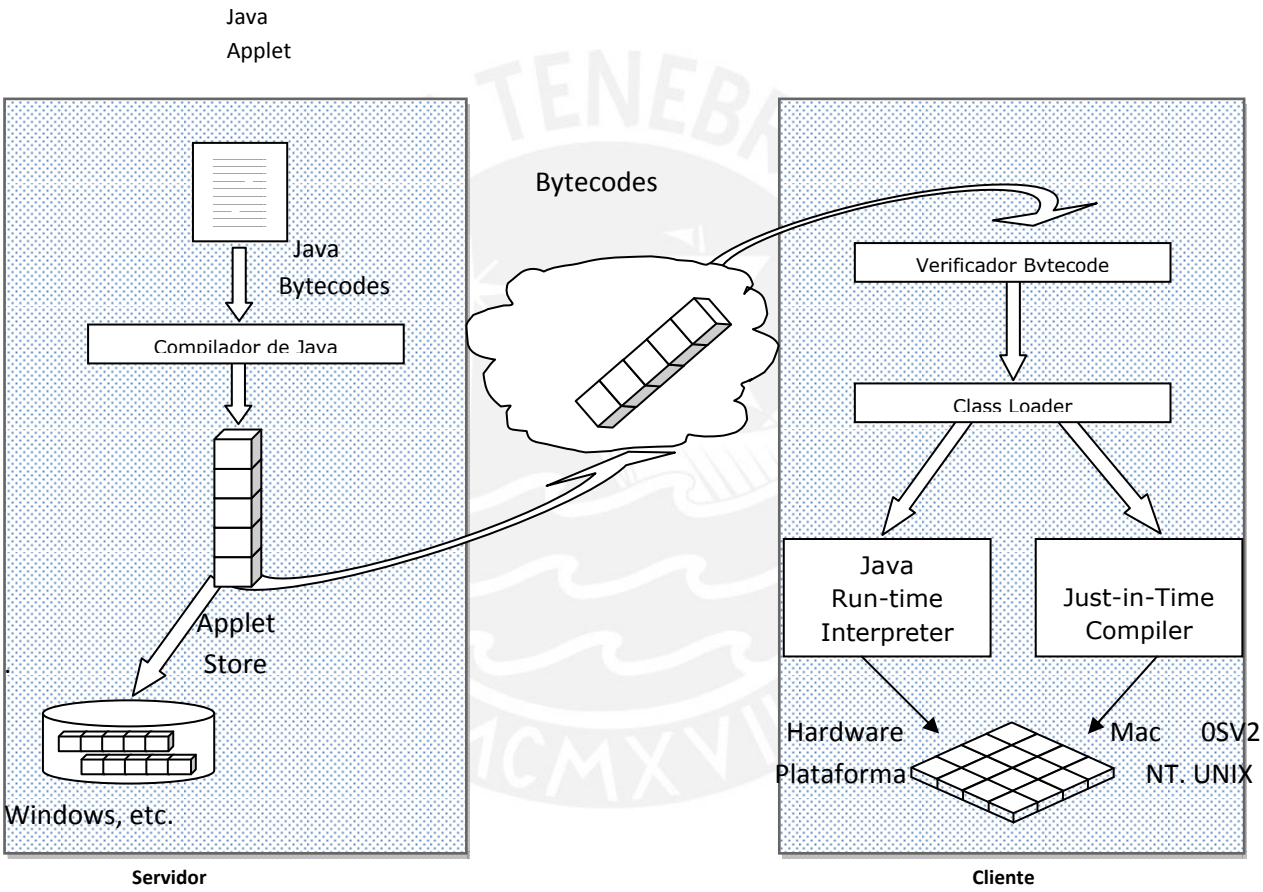
Es un lenguaje de paradigma orientado a objetos, fue inventado por James Gosling en 1993 en Sun Microsystems (ahora subsidiaria de la corporación Oracle), inspirado en la sintáxis de C++, incluye un sistema de gestión de memoria y un mecanismo de tratamiento de excepciones y concurrencia; y sobre todo se orienta hacia la programación de aplicaciones Web. La primera versión Java 1.0 en 1995 se basó en una máquina

[19] http://iel.ucdavis.edu/publication/WhyC.html
[20] http://www.zdnet.com/blog/microsoft/microsoft-outlines-plans-to-make-c-more-suited-for-massively-parallel-systems/9697?tag=mantle_skin;content

[21] http://www.open-std.org/JTC1/SC22/WG14/
[22] http://c-faq.com/

virtual estándar (Java Virtual Machine-JVM) que es incorporada en muchos servidores y clientes, sin importar la plataforma porque transforma el archivo del código fuente en "Bytecodes", el cual puede ser ejecutado en otro hardware con diferente plataforma (Ver figura 3.2). Java desarrolló actualizaciones como Java 1.1 que incluyó el desarrollo de múltiples librerías y reconfiguró el manejo de eventos que se había desarrollado con Java 1.0. Luego se continúan mejoras con versiones como J2SE Platform Standadrd Edition, J2SE 1.3, J2SE 1.4, J2SE5, Java SE6 y Java SE7[23].

Fig. 3.2.: Compilación de un programa en Java en una arquitectura Cliente Servidor



3.2.2. Características resaltantes de su diseño

Según [Martin 08], las características por las que Java se diferencia de los demás:

- Es un lenguaje totalmente orientado a objetos, todos los conceptos en los que se apoya esta técnica, encapsulación, herencia, polimorfismo, entre otros; están presentes en Java.

[23] Ver documentación completa de cada una de las implementaciones y versiones de Java en http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

- Los programas en Java están organizados en clases, en archivos de texto con archivo *.java.* Cada archivo de código fuente.java puede contener una o varias clases, aunque lo normal es que haya un archivo por clase.
- Disponibilidad de un amplio conjunto de librerías, Sun Microsystem pone a disposición del programador una serie de librerías con las cuales se puede realizar, prácticamente, todo tipo de aplicación, desde interfaces gráficas, gestión de red, multitarea, acceso de datos y un largo etcétera.
- Aplicaciones multiplataforma, una vez que se ha compilado el programa, éste puede ser ejecutado en diversos sistemas operativos sin necesidad de realizar cambios en el código fuente, sin que haya que volver a compilarse el programa. Esta independencia se logra con el concepto de máquina virtual.
- Ejecución segura de aplicaciones, la seguridad de los programas en Java se manifiesta porque por un lado el lenguaje carece de instrucciones que puedan provocar accesos descontrolados a la memoria, Java no muestra problemas de punteros como lo hace C/C++. Por otro lado, la máquina virtual, que es el entorno en el que ejecutan las aplicaciones Java, impone ciertas restricciones a las aplicaciones para garantizar una ejecución segura.
- Amplio soporte de fabricantes software, porque los programas Java no están vinculados a un determinado sistema operativo, tanto para el caso de entornos de desarrollo o los servidores de aplicaciones.
- Utiliza herencia que facilita reutilización de código y abstracción de datos.

Las aplicaciones de Java son diversas, entre juegos online y aplicaciones móviles, ocupa un gran porcentaje de aplicaciones en el mercado, casi un 100% de los reproductores Blu-ray corren en Java.[24] En proyectos de software libre y de Open Source se tiene una comunidad muy activa[25] y es el segundo lenguaje de programación más utilizado.

[24] Porcentaje de las aplicaciones de Java en diferentes proyectos: http://www.java.com/en/about

[25] http://java-source.net/

3.3. El lenguaje de programación Haskell

3.3.1. Origen, propósito y evolución

Es un lenguaje de paradigma funcional creado en un comité liderado por Simon Peyton Jones, investigador de Microsoft[26], y John Hughes en los años 80's. La mayoría de los desarrolladores y diseñadores de este lenguaje de programación son científicos con grado Phd de diferentes prestigiosas universidades de Norteamérica, quienes han tratado de evitar errores de diseño de otros lenguajes de programación.

Se crearon funciones predeterminadas de manera que el programador pueda utilizar menos código y por ello asignaron nombres a las funciones de acuerdo al rol que desempeña para que puedan ser empleadas de manera intuitiva.

Además diseñaron el sistema Glasgow Haskell Compiler (GHC) que consta de un compilador que genera código nativo rápido (ghc). Un intérprete interactivo y depurador (ghci) y un programa para correr programas codificados con Haskell como scripts sin necesidad de compilarlos (runghc). También se cuenta con un intérprete Hugs, el cual es utilizado mayormente con fines académicos. Básicamente Hugs 98 está basado en Haskell 98 y muestra en el prompt al comenzar la consola de Linux y/o Windows, el módulo estándar Prelude, así: “Prelude>”; el cual es un módulo que contiene los tipos, valores y funciones básicas que son comúnmente utilizados. [O'Sullivan 98]

Los recientes avances del cálculo tipo lambda, como la revisión de la sintáxis y semántica de los tipos de cálculo lambda con sumas y categorías cerradas bicatersianas (Bicartesian closed categories - BiCCCs) de las relaciones de Grothendieck[27], uso del algoritmo TDPE[28] son conceptos matemáticos que toma Haskell como base y adicionan un sistema fuertemente tipificado, que a diferencia del lenguaje funcional LISP, hace que Haskell sea mucho más seguro de utilizar. Además que cuenta con reglas de tipos que permiten que un programa de Haskell sea validado.

Debido a que Haskell evita los efectos secundarios (se obtiene siempre el mismo resultado de salida cuando se ingresa una y otra vez los mismos valores de entrada) ya que se basa en una fuerte teoría matemática que resuelve problemas con resultados exactos y aplicados a la programación, es que instituciones de renombre como la

[26] http://research.microsoft.com/en-us/um/people/simonpj/papers/haskell-tutorial/tasteofhaskell.pdf
[27] http://delivery.acm.org/10.1145/970000/964007/p64-balat.pdf?ip=200.62.245.186&acc=ACTIVE%20SERVICE&CFID=78147311&CFTOKEN=19221393&__acm__=1334874406_3dd92062852f1665cd02dce40b6671ec
[28] http://cs.au.dk/~danvy/NBE09/informal-proceedings/NBE09-Balat.pdf

NASA[29], han apoyado a programas de prácticas para estudiantes y trabajos[30], además de otras empresas que aplicaron Haskell a nivel industria[31].

Bajo licencia GPL o bajo licencia propietaria fueron creados ASIC y FPGA, diseños de chips de Bluespec Inc., Haskore, composición de música, compiladores y sus herramientas relacionadas (GHC), Darcs, controles de revisiones distribuidos y HappS de Galois Inc., Web middleware. Existen otras industrias que usaron Haskell, el ABN AMRO, banco internacional, que midió el riesgo de contraparte en las carteras de derivados financieros; Anygma, para herramientas de creación de contenido multimedia; Amgen, que creó modelos matemáticos y otras aplicaciones complejas; Eaton, para el diseño y verificación de los sistemas hidráulicos de vehículos híbridos, Software AG, una compañía importante de software alemán, el mercado de un sistema experto ejecutado en un mainframe IBM. [O'Sullivan 98]

3.3.2. Características resaltantes de su diseño

Las siguientes características están basados en las lecturas [O'Sullivan 98],[Lipovača 11]:

- Haskell emplea características de un lenguaje funcional como la evaluación perezosa, la cual realiza operaciones cuando se le indique, no antes, ni después.
- Haskell emplea un sistema de tipo de datos estático, que hace que los errores sean detectados en tiempo de compilación, si evalúa la operación entre un valor de tipo numérico con uno booleano (True && 1), nos arrojará un mensaje de error.
  Además cuenta con un sistema de inferencia de datos, el cual permite no declarar los tipos, es así que si colocamos a= 3+4, Haskell resuelve que "a" es de tipo numérico.
  Utiliza tipo amplia gama de tipos integrados (desde caracteres hasta números grandes), listas de comprensión, definición de nuevos tipos de datos y sinónimos de tipos. En un terminal Haskell, el comando ':t' indica el tipo de dato ya definido. Haskell define a 'Nothing' y 'Just' como valores del tipo de dato 'Maybe'. 'Just' es apropiado en lugar de Nothing (un análogo de 'NULL'). El sistema de tipos de Haskell define typeclasses, que son conjuntos de tipos de datos. Se tiene así que el typeclass 'Num' está conformado por otros dos sub-typeclasses: 'Integral' y 'Floating'. El typeclass 'Integral', incluye a los tipos 'Int' y 'Integer'; y el typeclass 'Floating', incluye los tipos 'Float' y 'Double'.

[29] http://www.nasa.gov/
[30] http://www.haskell.org/pipermail/haskell-cafe/2009-December/070558.html
[31] http://www.haskell.org/haskellwiki/Haskell_in_industry

- Haskell utiliza funciones de orden superior, lo que significa que la salida de una función puede ser entrada o parámetro que puede ser utilizado por otra función. Haskell también permite composición de funciones.
- Haskell hace uso del patrón sintáctico que incluye asignación de expresiones, condicionales 'if-else', 'case', uso de 'guards', notación 'do' y patrones 'as'. Un ejemplo aplicable con connotación matemática/física es:

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)
```

Un ejemplo de uso de la palabra reservada 'let' es:

```
ghci> [let square x = x * x in (square 5, square 3, square 2)]      - - square: funcion que eleva al cuadrado
[(25,9,4)]
```

Un ejemplo de uso de la palabra reservada ' if - else'

```
ghci> 4 * (if 10 > 5 then 10 else 0) + 2
42
```

Un ejemplo de uso de la palabra reservada 'case'

```
describeLista :: [a] -> String                    - - se define los tipos de entrada y salida en la función
describeLista xs = "Esta lista esta " ++ case xs of [ ] -> "vacia."
                                                    [x] -> "conformada por un elemento."
                                                    xs -> "conformada por muchos elementos."
```

Ejemplos de uso de los 'guards' son:

```
contador :: Int -> String
contador n
      | n == 0 = "no hay elementos"
      | n == 1 = "1 elemento"
      | n >  1 = show n ++ " elementos"
```

```
myCompare :: (Ord a) => a -> a -> Ordering
a `myCompare` b
   | a > b     = GT
   | a == b    = EQ
   | otherwise = LT
```

Un ejemplo de uso de la notación 'do' es:

```
foo :: Maybe String
foo = do
   x <- Just 3
   y <- Just "!"
   Just (show x ++ y)
```

Un ejemplo de uso de la notación 'as' es:

```
capital :: String -> String
capital "" = "cadena vacia"
capital all@(x:xs) = "The first letter of " ++ all ++ " is " ++ [x]
```

* 'all' es el *alias* para (x:xs), es así que si corremos el programa se tiene:

```
ghci> capital "Dracula"
"The first letter of Dracula is D"
```

- Haskell hace uso de la recursión para tratar datos que se trataban con estructuras "loops o bucles" en los lenguajes tradicionales como C.
  Ejemplo:

```
replicar :: (Num i, Ord i) => i -> a -> [a]
replicar n x
  | n <= 0   = [ ]
  | otherwise = x : replicar (n-1)  x
```

- En Haskell se utilizan 'Modules', esto es un análogo de las librerías que se utilizan en los lenguajes imperativos.

3.4. El lenguaje de programación GO

Google anunció un nuevo lenguaje de programación el 10 de noviembre de 2009 durante una charla técnica de Go a cargo de Rob Pike, quien demostró una compilación limpia de la librería estándar con duración de unos pocos segundos.

Go es un lenguaje multiparadigma, proviene de C y también toma algunas consideraciones del paradigma orientado a objetos, es así que al igual que Objective-C, Go tiene un tiempo de ejecución que permite el despacho dinámico. Mark Summerfield declara en su libro recientemente publicado, "Go está más cerca en espíritu a C que a cualquier otro lenguaje de programación".

La versión Go 1 define un lenguaje y un conjunto de librerías del kernel para proporcionar una base estable para la creación de productos fiables, proyectos y publicaciones. Está disponible en las distribuciones binarias compatibles. Están disponibles para Linux, FreeBSD, Mac OS X, y Windows. La motivación que impulsa a un Go es la estabilidad para sus usuarios. Los programadores pueden estar seguros de que esos programas van a continuar para compilar y ejecutar sin cambios, en muchos ambientes, en una escala de tiempo de años. Del mismo modo, los autores que escriben libros sobre Go 1 puede estar seguro de que sus ejemplos y explicaciones será útil para los lectores de hoy y en el futuro. El código que se compila en un Go, con pocas excepciones, se siguen para compilar y ejecutar durante la vida útil de esa versión, así como nosotros publican actualizaciones y correcciones de errores, tales como la versión Go 1.1, 1.2, y así sucesivamente. La última versión de GO “1.7.1” fue lanzada el 22 de Agosto.

Durante el evento llamado “Google I / O”, el lenguaje de programación GO fue presentado en cuatro sesiones. Rob Pike enfatizó que la concurrencia es la clave para diseñar servicios de alto rendimiento en la red. Concurrencias primitivas de GO (goroutines y canales) ofrecen un medio sencillo y eficaz para expresar la ejecución concurrente. Gustavo Niemeyer (Canonical), Keith Rarick (Heroku), Evan Shaw (Iron.io), y Patrick Crosby (StatHat) comparten su experiencia de primera mano con Go en entornos de producción. Se utilizó la API de Google Maps and Go en App Engine para construir una aplicación para crear conjuntos personalizados de azulejos para Google Maps. La aplicación muestra el uso de idoneidad para el cálculo de Go en la nube y de las características clave de App Engine escalabilidad, como las colas de tareas y backends.[32]

[32] http://blog.golang.org/

CAPITULO IV:

CRITERIOS DE EVALUACIÓN DE UN LENGUAJE DE PROGRAMACIÓN

4.1. Según David A. Watt

Para seleccionar el lenguaje adecuado en determinado proyecto de desarrollo de software, se recomienda evaluar criterios de relevancia técnica y económica como:

4.1.1. Escalabilidad

El lenguaje debe permitir la construcción de programas desde unidades de compilación que han sido codificados y probados separadamente, quizás por diferentes programadores. La compilación separada es una necesidad práctica, dado que inconsistencias de tipos son menos comunes al interior de una unidad de compilación (escrito por un único programador) que entre las unidades de compilación (quizás escrita por diferentes programadores) y lo último no es detectado por compilación independiente.

4.1.2. Modularidad

El lenguaje debe soportar la descomposición de los programas en unidades de programa de tal manera que podemos distinguir claramente entre lo que hace cada una unidad de programa (vista de programador) y cómo será codificada (vista de implementador); esta separación es una herramienta de interés intelectual esencial para la gestión del desarrollo de programas de gran envergadura. Conceptos de este contexto son los procedimientos, paquetes, tipos abstractos, y clases.

4.1.3. Reusabilidad

El lenguaje debe soportar la reutilización de unidades de programa para acelerar la reutilización en las pruebas que se hace a dichas unidades de programa, e incluso éstas pueden tomarse para desarrollar nuevas unidades de programa. Los conceptos relevantes asociados a este criterio son los paquetes, tipos abstractos, clases y particularmente unidades genéricas.

4.1.4. Portabilidad

El lenguaje debe ayudar a la escritura de código portable; es decir, el código puede ser movido de una plataforma a otra, sin sufrir grandes cambios.

4.1.5. Nivel

El lenguaje debe motivar a los programadores a pensar en términos de abstracciones de alto nivel orientado a la aplicación y no forzarlos a pensar todo el tiempo en cuanto a detalles de bajo nivel, tales como bits y punteros. Ello puede propiciar errores, especialmente si se utilizan punteros (aunque a veces es necesario su uso).

4.1.6. Confiabilidad

El lenguaje debe ser diseñado de tal manera que pueda detectar y eliminar los errores de programación lo más rápidamente posible. Lo ideal es que los errores deben ser detectados en tiempo de compilación para garantizar la ausencia de errores en el programa en tiempo de ejecución. Los errores detectados en tiempo de ejecución garantizan que no causan algún daño, que no sea una excepción, o en el peor de los casos dar por concluido el programa. Los errores no detectados en absoluto pueden causar daño sin límites (datos corruptos) antes de que el programa se bloquee, congele o cuelgue. La confiabilidad es siempre importante y esencial en sistemas críticos.

4.1.7. Eficiencia

El lenguaje debe ser capaz de ser aplicado de manera eficiente. Algunos aspectos de la programación OO implican sobrecargas en tiempo de ejecución, tales como tags de clases, dispatch dinámicos, garbage collection que hace más lento el programa en momentos impredecibles y los controles en tiempo de ejecución (aunque algunos compiladores están dispuestos a suprimirlos, con riesgo por cuenta del programador).

El código interpretado es unas diez veces más lento que el código máquina nativo, aunque las partes críticas del programa debe ser altamente eficiente y el lenguaje debe permitir ajustes de codificación de bajo nivel, o hacer llamadas a procedimientos escritos en un lenguaje de bajo nivel.

4.1.8. Legibilidad

El lenguaje debe motivar buenas prácticas de programación, no debe imponer sintaxis críptica, identificadores muy cortos, declaraciones por defecto o ausencia de información de tipo; ya que eso hace que sea difícil escribir código legible. El punto es que el código es leído (por su autor y otros programadores) con más frecuencia de lo que es escrito.

4.1.9. Modelamiento de datos

El lenguaje debe ofrecer tipos, y operaciones asociadas que sean adecuadas para la representación de las entidades en las pertinentes áreas de aplicación. Ejemplos de ello serían los registros y archivos de procesamiento de datos, números reales y las matrices de cálculo científico, cadenas en el procesamiento de texto, listas, árboles, mapping de los traductores, las colas en los sistemas operativos y simuladores. Si el lenguaje en sí mismo carece de los tipos, es necesario que permita a los programadores definir nuevos tipos y operaciones que las entidades con precisión el modelo en el área de aplicación. Conceptos de este contexto son los tipos abstractos y las clases.

4.1.10. Modelamiento de procesos

El lenguaje debe proporcionar estructuras de control adecuadas para modelar el comportamiento de las entidades en las pertinentes áreas de aplicación. En particular, se necesita concurrencia para modelar simultáneos procesos en simuladores, sistemas operativos y controladores de procesos en tiempo real.

4.1.11. Disponibilidad de los compiladores y herramientas

El lenguaje debe tener disponible compiladores de buena calidad que haga cumplir la sintaxis del lenguaje y las reglas tipo, que genere código objeto correcto y eficiente, que genere controles en tiempo de ejecución (por lo menos como opción) para atrapar cualquier error que no se pueden detectar en tiempo de compilación, y los informes de todos los errores de forma clara y precisa. Además se debe disponer de un entorno integrado de desarrollo de buena calidad (IDE) porque aumenta la productividad mediante la combinación de un editor de programas, compilador, enlazador, depurador y herramientas relacionadas a un único sistema integrado.

4.1.12. Familiaridad

El lenguaje debe ser conocido por los programadores disponibles en el mercado. En caso no sea así, se debe analizar si existe disponibilidad de formación de alta calidad y si la inversión que se aplicará en tiempo y dinero justifica la formación para futuros proyectos.

4.2. Según Robert W. Sebesta

[Sebesta 10] examina cuidadosamente los conceptos subyacentes de las diversas construcciones y capacidades de los lenguajes de programación y se evalúa sus características con el impacto del proceso de desarrollo de software y mantenimiento.

4.2.1. Legibilidad

Refiere a la manera que un programa pueda ser leído y comprendido. Algunos factores que afectan legibilidad son el costo de mantenimiento de software, el contexto del dominio del problema, simplicidad en general, características múltiples, sobrecarga de operadores y ortogonalidad. Si el programa es largo, horas de lectura de y comprensión del programa son demandadas por parte de un programador (el mismo que pudo haber escrito el código o uno diferente). Asimismo, si el lenguaje no fue diseñado para tal uso y la escritura fue forzada para que funcione, entonces la lectura se hace dificultosa.

Si un lenguaje tiene muchas construcciones básicas, se hace más difícil de comprender, y muchas veces por aprender todas ellas, el programador ignora otras características del lenguaje. Los problemas ocurren si se tiene más de una manera para cumplir la misma operación, si un operador tiene más de un significado o función. Pequeñas construcciones primitivas pueden ser combinadas para construir otras estructuras y sentencias de control. Otro problema se da cuando el programador ha aprendido un diferente subconjunto de instrucciones dentro de las instrucciones de un lenguaje de programación con el que está familiarizado y lo aplica en otro lenguaje de programación.

Por lo tanto el lenguaje debe tener una adecuada definición de los tipos de datos y un establecimiento de formas de identificadores y connotación de variables. Tener en cuenta las palabras especiales: formas y apariencias de las palabras reservadas, formación de sentencias compuestas y grupos de sentencias de control; y la forma y significado: semántica o significado se desprende de la sintaxis y consiste en que su apariencia logre indicar el propósito de la sentencia.

4.2.2. Fácil escritura

Refiere a la forma que el lenguaje puede ser usado para crear programas en determinados dominios y que sea comprensible para la relectura. Se basa en la simplicidad y ortogonalidad, soporte para abstracciones y expresividad.
Simplicidad y ortogonalidad: Si el lenguaje tiene muchas construcciones, el programador puede que no se familiarice con todas, y ello origina el desuso de otras características que permita un código más elegante y/o eficiente. Por otro lado, si se tiene un número pequeño de construcciones y un consistente conjunto de reglas para combinarlas (ortogonalidad) es mucho más simple.
Soporte para abstracciones, refiere a la definición y uso de estructuras complicadas y operaciones sin especificar muchos detalles. Se soportan dos categorías de abstracción en un lenguaje de programación, a nivel de procesos y a nivel de datos; y son muy útiles en el caso que se replique en todos lados del programa cada vez que se necesiten.
Expresividad, que refiere a que los operadores permitan muchos cálculos abreviados obteniendo así cortos programas, siendo oportunos y evaluando conveniencia de su uso.

4.2.3. Confiabilidad

Refiere a desarrollar las especificaciones bajo todas las condiciones.
Verificación de tipos: Si el programa tiene errores de tipo de dato es preferible identificarlos en tiempos de compilación o ejecución. Mientras más temprano se detectan hay menos costo de reparación.

Manejo de excepciones: Si el programa tiene la habilidad de interceptar errores en tiempo de ejecución para tomar medidas correctivas y continuar.

Uso de Alias: Si se tienen dos o más nombres distintos para acceder a la misma celda de memoria, el programador debe recordar los cambios efectuados y ello hace que no sea tan confiable el programa. Algunas veces es usado para sobrellevar deficiencias de abstracción de datos pero no es confiable.

Legibilidad y escritura: si un programa escrito en un lenguaje que no soporta formas naturales para expresar lo que requiere el algoritmo se usarán aproximaciones no naturales y son menos probables de ser corregidos o modificadas, afectando costos de mantenimiento.

4.2.4. Costo

De entrenamiento del programador: en base a la simplicidad y ortogonalidad del lenguaje, además de la experiencia del programador.

De escritura: tiene que ver con la cercanía en el propósito de la aplicación en particular, se debe evaluar ambiente del lenguaje de programación.

De compilación: optimización es el nombre dado a la colección de técnicas de compilación que se pueden usar para decrecer el tamaño o incrementar la velocidad de un código producido.

De implementación: disponibilidad del compilador o del intérprete del lenguaje de programación.

De baja confiabilidad: si el programa es aplicado en industrias que requieren cálculos exactos e infalibles, libre de riegos y futuras pérdidas.

De mantenimiento: si se requiere agregar o modificar funcionalidades de un programa.
El costo de mantenimiento es de 2 a 4 veces mayor que el costo de desarrollo.
También refiere el autor a un costo de portabilidad, la manera que un programa puede ser movido de una implementación a otra. Esto es influenciado por el grado de estándares del lenguaje, y muchas veces conseguir la estandarización de un lenguaje de programación demanda muchos años de investigación.
Finalmente, generalidad, que refiere a la aplicabilidad a un gran rango de aplicaciones, con una completa definición y documentación oficial del lenguaje de programación.

4.3. Según Terrence W. Pratt

[Pratt 98] nombra los siguientes atributos como atributos de un buen lenguaje. Además señala, que a pesar de la gran importancia de algunas de estas influencias externas, es el programador en último término, aunque a veces de manera indirecta, quienes determinan qué lenguajes viven o mueren. Se podrán sugerir múltiples razones para explicar por qué los programadores prefieren un lenguaje respecto a otro.

4.3.1. Claridad, sencillez y unidad

Un lenguaje de programación proporciona a la vez un marco conceptual para pensar acerca de los algoritmos y un medio de expresar esos algoritmos. El lenguaje debe constituir una ayuda para el programador mucho antes de la etapa misma de codificación.

Debe proveer un conjunto claro, sencillo y unificado de conceptos que se puedan usar como primitivas en el desarrollo de algoritmos. Para ello es deseable contar con un número mínimo de conceptos distintos cuyas reglas de combinación sean tan sencillas y regulares como sea posible. Llamamos a este atributo integridad conceptual.

La sintaxis de un lenguaje afecta la facilidad con la que un programa se puede escribir, poner en prueba, y más tarde entender y modificar. La legibilidad de los programas es importante. Una sintaxis que es particularmente tersa o hermética suele facilitar la escritura de un programa (para el programador con experiencia) pero dificulta su lectura cuando es necesario modificarlo más tarde. Muchos lenguajes contienen construcciones sintácticas que favorecen una lectura errónea al hacer que dos enunciados casi idénticos signifiquen en efecto cosas radicalmente distintas. Un lenguaje debe tener la propiedad de que las construcciones que signifiquen cosas distintas se vean diferentes; es decir, las diferencias semánticas deberán reflejarse en la sintaxis del lenguaje.

4.3.2. Ortogonalidad

Se refiere al atributo de ser capaz de combinar varias características de un lenguaje en todas las combinaciones posibles, de manera que todas ellas tengan significado. Por ejemplo, supóngase que un lenguaje dispone de una expresión que puede producir un valor, y también dispone de un enunciado condicional que evalúa una expresión para obtener un valor de verdadero o falso. Estas dos características del lenguaje, la expresión y el enunciado condicional, son ortogonales si se puede usar (y evaluar) cualquier expresión dentro del enunciado condicional.

Cuando las características de un lenguaje son ortogonales, entonces es más fácil aprender el lenguaje y escribir los programas porque hay menos excepciones y casos especiales que recordar. El aspecto negativo de la orotogonalidad es que un programa suele compilar sin errores a pesar de contener una combinación de características que son lógicamente incoherentes o cuya ejecución es en extremo ineficiente. Por ello, la ortogonalidad es aún una característica controversial.

4.3.3. Naturalidad para la aplicación

Un lenguaje necesita de una sintaxis que al usarse correctamente permita que la escritura del programa refleje la estructura lógica subyacente del algoritmo. Idealmente, deberá ser posible traducir directamente a un diseño de programa de ese tipo a enunciados de programa adecuados que reflejen la estructura del algoritmo. Los

algoritmos secuenciales, algoritmos concurrentes, algoritmos lógicos, etc. todos ellos tienen estructuras naturales diferentes que están representadas por programas en esos lenguajes. El lenguaje deberá suministrar estructuras de datos, operaciones, estructuras de control y una sintaxis natural apropiada para el problema que se va a resolver. Una de las razones principales de la proliferación de lenguajes es precisamente esta necesidad de naturalidad, un lenguaje particularmente adecuado para una cierta clase de aplicaciones puede simplificar grandemente la creación de programas individuales en esa área.

4.3.4. Apoyo para la abstracción

Incluso con el lenguaje de programación más natural para una aplicación, siempre queda una brecha considerable entre las estructuras de datos y las operaciones abstractas que caracterizan la solución de un problema y las estructuras de datos primitivos y operaciones particulares integradas en un lenguaje. Una parte considerable de la tarea del programador es proyectar las abstracciones adecuadas para la solución del problema y luego implementar esas abstracciones empleando las capacidades más primitivas que provee el lenguaje de programación mismo. Idealmente, el lenguaje debe permitir la definición y el mantenimiento de las estructuras de datos, de los tipos de datos y de las operaciones como abstracciones autosuficientes. El programador puede emplear todo esto en otras partes del programa conociendo sólo sus propiedades abstractas, sin preocuparse por los detalles de su implementación.

4.3.5. Facilidad para verificar programas.

La confiabilidad de los programas escritos en un lenguaje es siempre una preocupación medular. Existen muchas técnicas para verificar que un programa ejecuta correctamente la función„ requerida. Se puede probar que un programa es correcto a través de un método formal de verificación, se puede probar informalmente que es correcto por verificación de escritorio (leer y verificar visualmente el texto del programa), se puede poner a prueba ejecutándolo con los datos de entrada de prueba y comparando los resultados de salida con las especificaciones, etc.

Para programas grandes se suelen emplear alguna combinación de todos éstos métodos. El uso de un programa que dificulta la verificación de programas puede ser mucho más problemático que el de uno que apoya y simplifica la verificación, no obstante que el primero puede proporcionar muchas capacidades que superficialmente parecen hacer más fácil la programación. La sencillez de la estructura semántica y sintáctica es un aspecto primordial que tiende a simplificar la verificación de programas.

4.3.6. Entornos de programación

La estructura técnica de un lenguaje de programación es solo uno de los aspectos que afectan su utilidad. La presencia de un entorno de programación adecuado puede facilitar el trabajo con un lenguaje técnicamente débil en comparación con un lenguaje más fuerte con poco apoyo externo. Se podría incluir una larga lista de factores como parte del entorno de programación. La disponibilidad de una implementación confiable, eficiente y bien documentada del lenguaje debe encabezar la lista. Los editores especiales y paquetes de prueba hechos a la medida para el lenguaje pueden acelerar mucho la creación y puesta a prueba de programas.

Los recursos para el mantenimiento y modificación de múltiples versiones de un programa pueden simplificar mucho el trabajo con programas grandes. De los lenguajes que se estudian en este libro, sólo Smalltalk fue proyectado específicamente alrededor de un entorno de programación compuesto de ventanas, menús, uso del ratón y un conjunto de herramientas para operar sobre programas escritos Smalltalk.

4.3.7. Portabilidad de programas

Un criterio importante para muchos proyectos de programación es el de la portabilidad de los programas resultantes de la computadora en la cual se desarrollaron hacia otros sistemas de computadoras. Un lenguaje que está ampliamente disponible y cuya definición es independiente de las características de una máquina particular constituye una base útil para la producción de programas transportables. Ada, FORTRAN, C y Pascal tienen todos ellos definiciones estandarizadas que permiten la implementación de aplicaciones transportables. Otros, como ML, provienen de una implementación de fuente única que permite al diseñador de lenguajes cierto control sobre las características transportables del lenguaje.

4.3.8. Costo de uso

Costo de ejecución del programa: Antes, las cuestiones de costos de referían casi exclusivamente a la ejecución de los programas. Era importante el diseño de compiladores que optiman, la asignación eficiente de registros y el diseño de mecanismos eficientes de apoyo al tiempo de ejecución. El costo de ejecución de programa, es de importancia primordial para grandes programas de producción que se van a ejecutar con frecuencia. En la actualidad, sin embargo, para muchas aplicaciones la velocidad de ejecución no es la preocupación mayor. Con máquinas de escritorio que

trabajan a varios millones de instrucciones por segundo y que están ociosas gran parte del tiempo, se puede tolerar un incremento de 10% o 20% del tiempo de ejecución si ello significa un mejor diagnóstico o un control más fácil por parte del usuario sobre el desarrollo y mantenimiento del programa.

Costo de traducción del programa: Cuando un lenguaje como FORTRAN o C se utiliza en la enseñanza, la cuestión de una traducción (compilación) eficiente, más que la ejecución eficiente, puede ser de importancia capital. Típicamente, los programas estudiantiles se compilan muchas veces cuando están depurando pero se ejecutan sólo unas pocas veces. En casos así, es importante contar con un compilador rápido y eficiente en vez de un compilador que produzca un código ejecutable optimizado.

Costo de creación, prueba y uso de programas: Para una cierta clase de problemas se puede diseñar, codificar, probar, modificar y usar una solución con una inversión mínima en tiempo y energía del programador. Smalltalk es económico en cuanto a que se minimizan el tiempo y esfuerzo totales que se invierten en la solución de un problema en la computadora. La preocupación por esta clase de costo de conjunto en el uso de un lenguaje se ha vuelto tan importante en muchos casos como la inquietud por la ejecución y compilación eficiente de los programas.

Costo de mantenimiento de programas: Muchos estudios han demostrado que el costo más grande que interviene en cualquier programa que se utiliza a lo largo de un periodo de años no es el costo del diseño, codificación y prueba inicial del programa, son los costos totales del ciclo vital, que incluyen los costos de desarrollo y el costo de mantenimiento del programa en tanto continúa en uso productivo. El mantenimiento incluye la reparación de errores, los que se descubren después de que se comienza a usar el programa, cambios que requiere el programa cuando se actualiza el hardware subyacente o el sistema operativo, y extensiones y mejoras al programa que se requieren para satisfacer nuevas necesidades.

4.4. Según investigaciones de otros autores

Según la investigación de [Garcia 07], muchos lenguajes de programación modernos soportan programación de genéricos en nivel básico, suficientes para implementar contenedores polimórficos con seguridad de tipos. Algunos lenguajes han ido más allá de este soporte básico, y al hacerlo, han permitido una forma más amplia y poderosa de programación genérica. En su estudio identificó ocho propiedades que soportan programación genérica, ellos son: soporte para conceptos multi-tipo, restricciones de

tipos asociados, conveniente acceso de tipos asociados, restricciones de tipos de asociados, modelado retroactivo, los alias de tipo, compilación independiente de algoritmos estructuras de datos, deducción implícita tipo de argumento para los algoritmos genéricos. El estudio fue aplicado a los lenguajes de programación Java, Haskell, C#, Schema, C++, ML, Eiffel, Cecil y OCalm. El estudio demostró que estas características son necesarias para evitar diseños complicados, mantenimientos deficientes y código demasiado extenso.

Por ello, es importante que los diseñadores de lenguajes de programación tengan en cuenta estas características necesarias para permitir que los lenguajes soporten mejor y de manera más eficiente los genéricos, debido a que el uso efectivo de los mismos y su ausencia puede causar dificultades para los programadores. Un ejemplo de dificultad, se puede dar en la dificultad en lectura de código, existe un tiempo invertido que se da en el tiempo de mantenimiento de muchos proyectos de software.

[Garcia 07] realiza un estudio comparativo de los lenguajes de programación: Estándar ML, Objective Caml, C++, Haskell, Eiffel, Java, C# y Cecil, debido a que éstos soportan programación genérica. La comparación se basa en cómo la programación genérica puede ser expresada en cada uno de los lenguajes de programación en mención; ya que se requieren muchos aspectos para programar diferentes tipos de datos y funciones. Ello involucra parámetros de tipos, conceptos de multi-tipos, restricciones múltiples, accesos de tipos asociados, alias de tipos, modelo retroactivo, entre otros.

Google, la empresa con mejor reputación en el mundo 2010, que según el FRC (Foro de Reputación Corporativa), logró 78,62 puntos de 100, sobre Sony (78,47) y The Walt Disney Company (77,97); ha venido desarrollando un lenguaje de programación denominado GO. Este trabajo liderado por Russ Cox, Robert Griese, Rob Pike, Ian Taylor, Ken Thompson junto con la participación de David Symonds, Nigel Tao entre otros muchos contribuidores de la comunidad Open Source ha sido presentado por primera vez en la Universidad de Stanford el 28 de Abril del 2010 [Pike 10] . En él, se puede apreciar que fueron tomados algunos aspectos de los lenguajes de programación Java y C entre otros para mejorar debilidades y enfocarse en las aspectos demandantes en los software en la actualidad. Indica que hay un nicho por llenar, un lenguaje de programación debe tener buenas cosas y evitar las malas, que se ajuste a infraestructuras modernas en cómputo, que tengan legibilidad, tipos estáticos, que sean luz en las páginas, que sean rápidas en el trabajo, que escalen bien, que no requieran

herramientas, pero que las soporten bien, que sean buenas en la red y en multiprocesos. Plantea que el lenguaje debe cumplir lo siguiente: rápida compilación, expresividad a través del sistema de tipos, concurrencia, recolector de basura intrínseco, capacidades de cubrir todo sistema operativo, claridad y ortogonalidad. GO también considera reflexión, escalable y eficiente.

En el presente estudio, se ha considerado la publicación de la sección "Programming Languages" del "Computer Science" de "Cornell University Library" denominado "Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, PHP & Scheme - a Team 11 COMP6411-S10 Term Report. [Venkatreddy 10]. Esta publicación está basada en un investigación que compara lenguajes de programación [Prashant 08] y una investigación que cuestiona la popularidad del lenguaje de programación Java [Živanović 10].

En estos estudios se presentan básicamente como criterios de evaluación de los lenguajes de programación:

- Buenas prácticas de seguridad de programación
- Aplicaciones Web, diseño y composición de Servicios Web.
- Reflexión, abstracción basada en orientada a objetos.
- Programación funcional y programación declarativa
- Batch scripting y diseño de prototipos UI.

CAPITULO V: ESTUDIO COMPARATIVO HASKELL FRENTE A C, C++, JAVA Y GO

A continuación se presentan programas escritos en el lenguaje Haskell, los cuales serán comparados con programas escritos en los lenguajes de programación elegidos a lo largo del presente trabajo: C, C++, Java y GO. Los programas en los tres lenguajes de programación resolverán el mismo problema planteados frente a los criterios de evaluación también elegidos y detallados en el capítulo anterior: lectura de código, escritura del código, confiabilidad, reflexión y soporte para programación genérica.

Estos códigos han sido probados en una laptop Toshiba Portege R835-P50X[33], arquitectura de 64 bits en un sistema operativo Linux, utilizando la distribución Fedora16 y Ubuntu 11.10.

En la primera parte del presente capítulo se evalúa si un programa es fácil de leer. Se desarrolla un algoritmo que realiza un cálculo y se expone la resolución en líneas de códigos en los tres lenguajes de programación: C, Java y Haskell. Los programas son presentados a los alumnos de pregrado PUCP, quienes tienen previos conocimientos de lenguajes pertenecientes tanto al paradigma imperativo, orientado a objetos y funcional.

En la segunda parte se evalúa si un programa es fácil de escribir. Se muestran tres estructuras prácticas y básicas para probar código, el primer ejemplo se da con el uso de funciones, el segundo ejemplo refiere a la conexión de base de datos y el tercero, a la lectura de un archivo de diferente extensión.

En la tercera parte se evalúa la confiabilidad de los programas, se prueba si el lenguaje de programación brinda todas las facilidades para que el código detecte errores de programación a tiempo (en tiempo de compilación preferiblemente o en tiempo de corrida como segunda mejor alternativa).

En la cuarta parte se presentan programas que cumplen la definición de reflexión, se hará uso de estructuras de datos adaptables a cada lenguaje de programación que soporte esta característica. Este concepto relacionado directamente a la programación dinámica es prácticamente aplicado en herramientas que brindan servicios Web.

Finalmente, en la quinta parte se presentan programas que brindan soporte a la programación genérica. En este caso se ha adaptado el algoritmo denominado

[33] Intel® Core™ i3-2310M processor de 64 – bit, y 4GB DDR3 1333MHz de memoria.

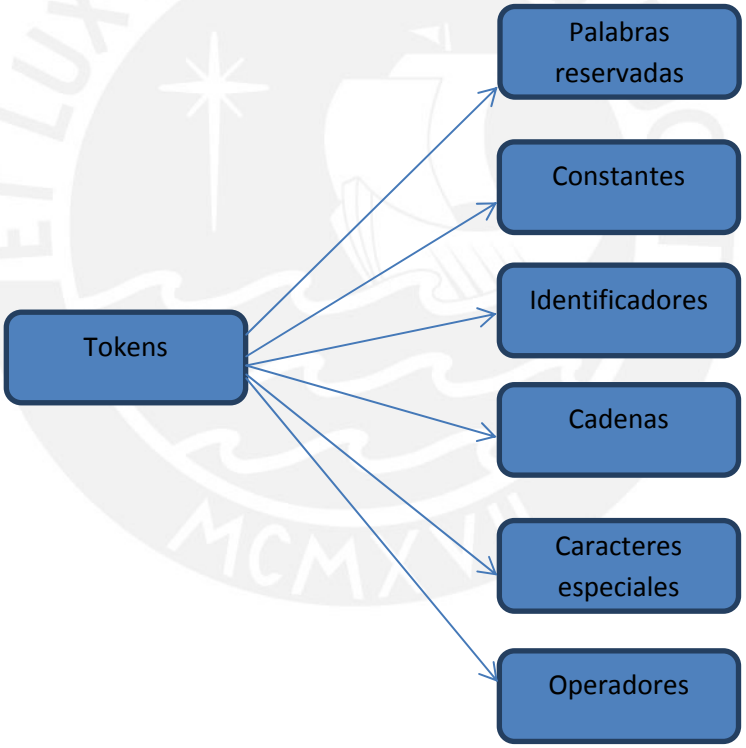
"Quicksort", para que pueda ordenar una lista de elementos y sin importar el tipo de dato de la lista. Se han empleado todas las estructuras y sintaxis posibles para poder lograrlo.

5.1 Fácil de Leer

Los lenguajes de programación C, C++ Java, Haskell y GO han sido diseñados utilizando el concepto de ciencias de la computación: BNF (Backus Normal o Forma de Backus-Naur Form)34.

BNF es una de las técnicas principales para la notación de las gramáticas libres de contexto, a menudo usados para describir la sintaxis de los lenguajes de programación. [35] Estas notaciones gramaticales están compuestas por caracteres, que a su vez se agrupan en tokens (unidad mínima de expresión del lenguaje).

Figura 5.1.: Tokens, estructura léxica de un lenguaje de programación



Fuente: http://www.c4learn.com/wp-content/uploads/2010/07/Tokens-in-C-Programming-Basic-Building-Blocks.jpg

Los tokens en C[36], los tokens en Java[37] y los tokens en Haskell[38] repercuten en la lectura de un código mientras un programador pretende aprender y/o familiarizarse con un

[34] http://bnfc.digitalgrammars.com/

[35] http://www.sc.ehu.es/jiwnagom/FLP/FLP-archivos/SintaxisBNF.pdf

[36] http://www.iitg.ernet.in/physics/fac/charu/courses/ph508/lecture2.ppt

lenguaje por primera vez, o cuando tiene que hacer mantenimiento de un proyecto de software.
La mayoría de sentencias conformadas por los tokens de un lenguaje de programación se leen en el idioma inglés, y muchas veces la funcionalidad del código se deduce por las variables con nombres alusivos al contexto definidas en determinada función.

Encuestas

Para probar la comprensión de los tokens en los lenguajes de programación C, Java, Haskell y GO, se presenta un mismo programa codificado en los cuatro lenguajes de programación y a través de una encuesta a los alumnos del primer ciclo de la PUCP, se les pide indicar la respuesta o salida del programa. (Ver anexo 04)

En esta experiencia no se utilizan estructuras de datos complejas, como punteros en el caso de C, Frameworks en caso de Java, monadas en el caso de Haskell, o goroutines en el caso de GO; debido a que cuando se enseña a programar a un alumno, se incluyen ejemplos de programas que tienen una entrada de datos, con calculo y salida de datos.

Las encuestas fueron realizadas de manera presencial a los alumnos del horario 01 y 02, que acaban de culminar el semestre 2012 – I. Los alumnos encuestados han tenido experiencias con el lenguaje de programación Visual Basic for Applications en el curso Introducción a la Computación.

Se obtuvieron 62 encuestas válidas, 20 encuestas del horario 01 y 42 encuestas del horario 02. La edad promedio de los encuestados fue 18.15.
Población masculina de un 80.5% y de población femenina de 19.5%.
La encuesta presenta el programa que calcula el BMI (Body Mass Index o índice de masa corporal de una persona). A continuación se presenta un pseudocódigo para el cálculo.

Inicio

Leer Peso, Leer Altura

bmi = $\left(\frac{peso}{altura}\right)^2$

Si bmi <= bajo_peso entonces
    Imprimir "Tiene un peso debajo del promedio"

[37] http://www.cs.cmu.edu/~pattis/15-1XX/15-200/lectures/tokens/lecture.html

[38] http://www.ualberta.ca/~jhoover/325-Notes/section/HaskellParse.htm

Sino

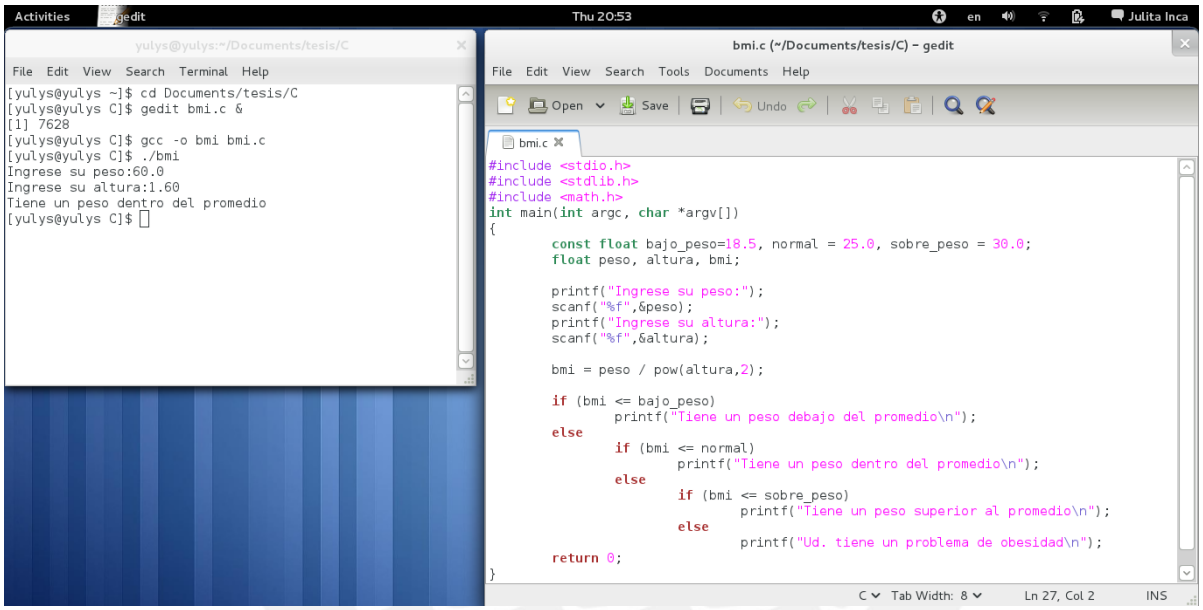
Si bmi <= normal entonces
    Imprimir "Tienes un peso promedio"

Sino
    Imprimir "Ud. está obeso"

Fin.

a) Lenguaje de Programación C

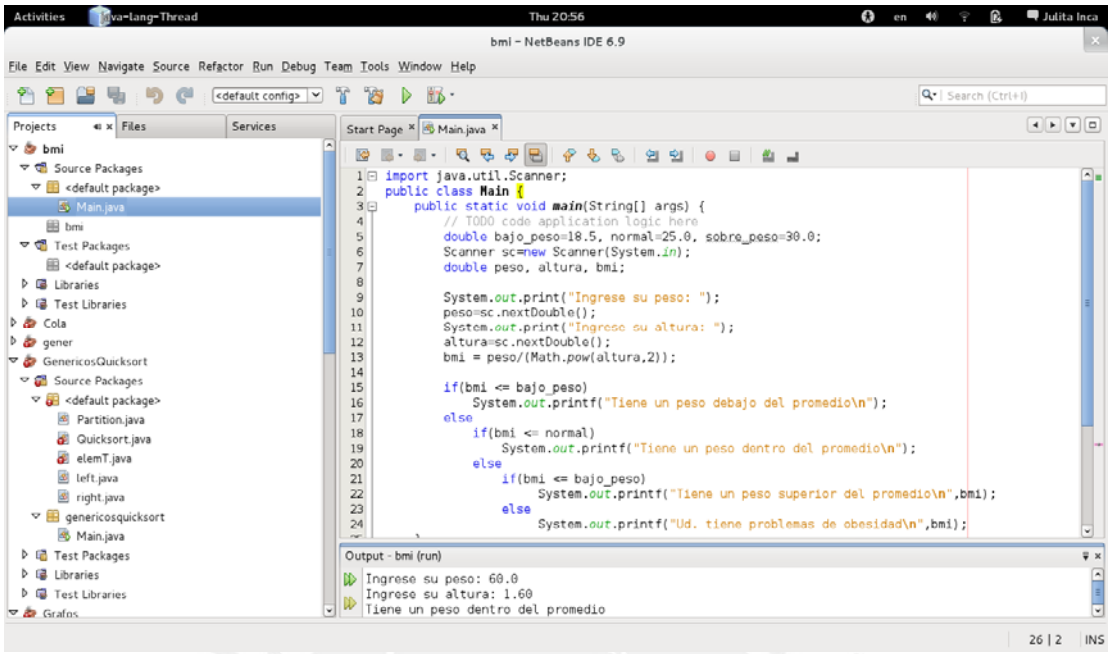
Figura 5.2.: Escritura de programa, compilación y ejecución del programa bmi.c.



Tal como se aprecia en la parte derecha de la Figura 5.2, se escribe el programa con la aplicacion Gedit. En la parte izquierda se aprecia en la cuarta línea como se emplea el comando gcc para poder compilar el programa y finalmente se corre el programa en la quinta fila. Al correr el programa pide datos de entrada como el peso y la altura, para finalmente mostrar el mensaje “Tiene un peso dentro del promedio”, según lo calculado.

b) Lenguaje de Programación JAVA

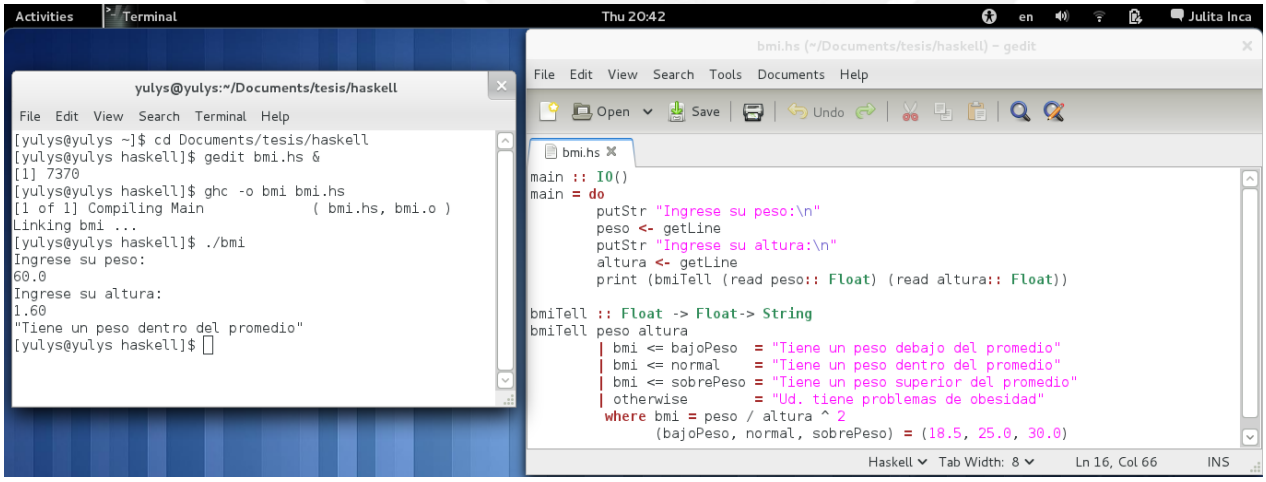
Figura 5.3.: Escritura de programa, compilación y ejecución del programa bmi en Java



Tal como se aprecia en la Figura 5.3, se crea el proyecto bmi utilizando la herramienta NetBeans IDE 6.9 y, corriendo la clase Main.java se obtiene el Output – bmi (run) del programa. Al correr el programa pide datos de entrada como el peso y la altura, para finalmente mostrar el mensaje “Tiene un peso dentro del promedio”, según lo calculado.

c) Lenguaje de Programación HASKELL

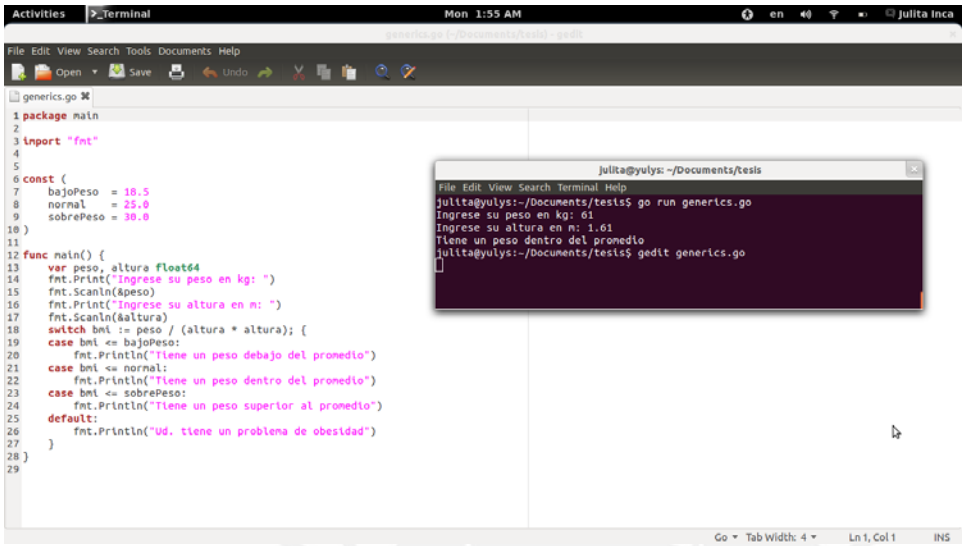
Figura 5.4.: Escritura de programa, compilación y ejecución del programa bmi en Haskell



Tal como se aprecia en la parte derecha de la Figura 5.4, se escribe el programa con la aplicacion Gedit. En la parte izquierda se aprecia en la cuarta línea como se emplea el comando ghc para poder compilar el programa y finalmente se corre el programa en la quinta fila. Al correr el programa pide datos de entrada como el peso y la altura, para finalmente mostrar el mensaje “Tiene un peso dentro del promedio”, según lo calculado.

d) Lenguaje de Programación GO

Figura 5.5.: Escritura de programa, compilación y ejecución del programa bmi en GO



Tal como se aprecia en la Figura 5.5, se escribe el programa bmi en GO utilizando la herramienta Gedit. Al correr el programa con el comado 'go run', pide datos de entrada como el peso y la altura, para finalmente mostrar el mensaje "Tiene un peso dentro del promedio", según lo calculado.

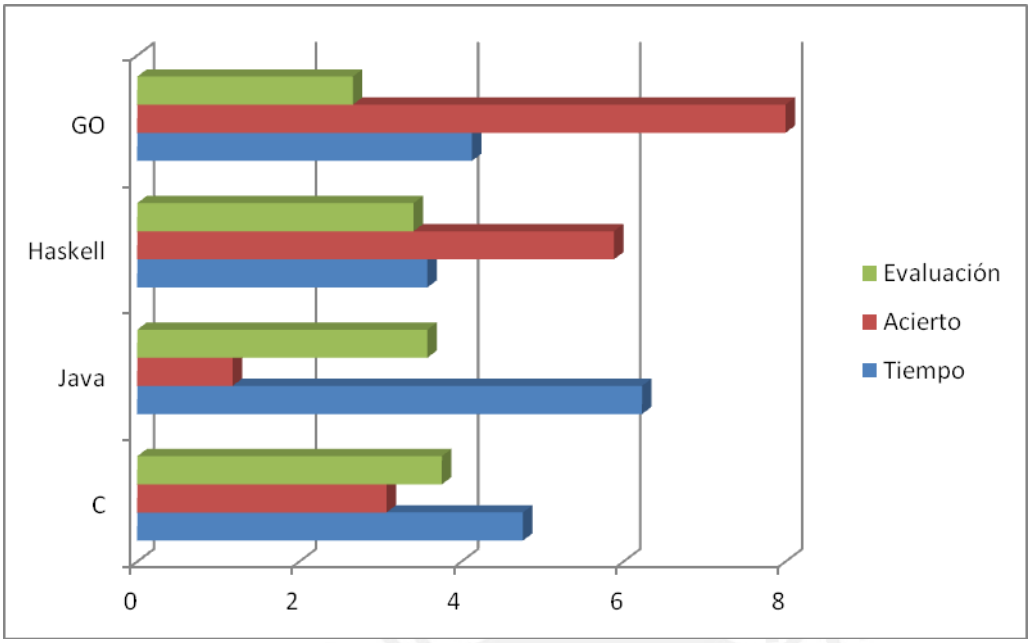
Los cuatro programas mostrados en las figuras 5.2, 5.3, 5.4 y 5.5, fueron presentados a los alumnos a manera de encuestas; las cuales han sido repartidas de manera aleatoria, de tal forma que a un alumno que le toca resolver la encuesta con un lenguaje no evalúa los otros dos.

Se plantearon tres factores que afectan la comprensión de lectura de un lenguaje de programación.

- ✓ Evaluación de comprensión del código
- ✓ Tiempo de respuesta de resolución del programa
- ✓ Acierto de la respuesta del programa

Los resultados generales de la encuesta se detallan en el Anexo 04; a continuación, los gráficos que demuestran estadísticas de acierto, tiempo de respuesta y evaluación:

Figura 5.6.: Gráfica de los resultados generales de las encuestas de comprensión de lectura de código en los lenguajes de programación C, Java, Haskell y GO.



Evaluación promedio del código

- 13 encuestas que contenían el programa desarrollado con el lenguaje de programación C evaluaron el código con un promedio de 3.76 minutos.
- 17 encuestas que contenían el programa desarrollado con el lenguaje de programación Java evaluaron el código con un promedio de 3.58 minutos.
- 17 encuestas que contenían el programa desarrollado con el lenguaje de programación Haskell evaluaron el código con un promedio de 3.41 minutos.
- 15 encuestas que contenían el programa desarrollado con el lenguaje de programación GO evaluaron el código con un promedio de 2.66 minutos.

GO fue el lenguaje calificado como menos dificultoso en la comprensión de lectura de código, seguido por Haskell, C y finalmente Java, que fue el más difícil de comprender.

Tiempo promedio de resolución

Haskell tuvo un tiempo de respuesta menor con respecto a los demás lenguajes de programación, con un tiempo promedio de 3.58 minutos; pero este resultado debe estar relacionado al porcentaje de acierto de la respuesta brindada por los alumnos. Si el alumno respondio en menor tiempo y bien, entonces si se obtuvo una buena comprensión de lectura de código.

El tiempo promedio de resolución utilizado en el lenguaje GO fue de 4.13 minutos, seguido por C con 4.76 minutos; y finalmente, Java obtuvo un tiempo de 6.23 minutos.

Porcentaje de acierto de la respuesta

El lenguaje de programación GO obtuvo un mayor porcentaje de acierto en la respuesta con un 80 % y un tiempo promedio de resolución de 4.13 minutos.

Haskell obtuvo un 58.82 % con un tiempo de 3.58 (diferencial delta de menos de un minuto), C obtiene el tercer lugar con un 30.76% y un tiempo promedio de resolución de 4.76 minutos, y finalmente Java obtiene el último lugar en esta prueba con 11.76 % de acierto con un tiempo de 6.23 minutos.

Evaluación de escritura de código

[Premchand 12] realiza un estudio de la evaluacion de la legibilidad y fácil lectura de código a 120 estudiantes de ciencias de la computación con 2.2 millones de líneas de código, enfatiza que la fácil lectura de código va directamente relacionada a un buen mantenimiento y calidad de software (ocupa 70% de la parte del ciclo de vida de un software). Indica que para que un código sea fácil de leer, debe tener documentación, buen uso de identificadores y variables, ser portable, reusable y estar bien indentado.
Evoca a Dijkstra que señala que la lectura es fácil si se tienen estructuras simples como secuencias de control y un diseño del sistema de lectura 'top-down'.

Asimismo, desarrolladores con muchos años de experiencia avalan que para que un código sea más fácil de leer se debe ser bien indentado[39], y debe utilizar palabras reservadas de pequeña longitud en lugar de utilizar símbolos "crípticos".[40]

[Dehnadi 06] realiza un estudio empírico donde personas sin previo conocimiento de lenguaje de programación, resuelven códigos de *asignación* (es decir asignar un valor a una variable) en Java en universidades de Canadá y Australia. Este estudio sugiere que las personas traen diferentes patrones de conocimientos o modelos mentales cuando entran en un nuevo proceso de aprendizaje. Este estudio sostiene que el lenguaje de programación a enseñarse en cursos introductorios se puede predecir. De acuerdo a las

[39] http://es.wikipedia.org/wiki/Indentaci%C3%B3n
[40] http://williamedwardscoder.tumblr.com/post/18319031919/programming-language-readability

respuestas consistentes capturadas en las hojas marcadas propuestas, se puede diferir los modelos mentales que tienen los principiantes, y esto permite crear nuevas herramientas que puedan medir la habilidad para programar y el paradigma que pueden aprender los estudiantes.

Siguiendo estas premisas, se ha evaluado el lenguaje de programación Haskell con respecto a los lenguajes de programación C, Java y GO:

a) Con respecto a la indentación, el único lenguaje que arrojó un mensaje de error en el momento de compilación, por no indentar código, fue el lenguaje de programación Haskell. Haskell y su compilador tienen una restricción para indentar que obliga al programador a hacerlo. Si no lo hace, el compilador no puede interpretar cual es el inicio y cual el fin de las sentencias.

   Tanto C y Java, tienen un botón para el formato indentación en los IDEs; sin embargo, sin indentación, los programas escritos en C y Java corrieron satisfactoriamente. GO a nivel editor de archivo, también corrió sin indentación.

b) Con respecto a las estructuras de código, en el programa desarrollado con Haskell, se tiene la palabra reservada 'where' que se coloca en una posición del código donde se puede deducir que, se trata de un preámbulo para establecer valores. Los alumnos comprendieron eso sin especificaciones previas. Asimismo se tiene estructuras condicionales en los lenguajes C, C++, Java y GO que permitieron a los alumnos determinar el valor de la salida del programa.

c) Con respecto a los tokens, el código de Haskell contiene símbolos casi crípticos como '::', '=>', '<=', y '|'. El programador que comienza a aprender el lenguaje de programación Haskell, debe conceptualizar el significado de cada uno de estos símbolos, y eso toma muchas horas de práctica.
   - '|' representa el concepto "guards". Los "guards" son una forma de comprobar si alguna propiedad de un valor (o varios de ellos) son verdaderas o falsas.
   - El símbolo '::' indica que la variable, identificador o token declarado "es de tipo".
   - '=>' separa dos partes, la parte que esta antes del código refiere a la 'restricción de clase'. Se puede leer de la siguiente manera: la función 'bmi' tiene dos valores cualesquiera de entrada de tipo genérico 'a' y devuelve un resultado tipo 'String'.
   - El símbolo '<=' permite asignación luego del resultado del cálculo de 'bmi'.

d) Con respecto a la asignación de valores, los cuatro lenguajes establecen claramente constantes y variables para realizar el cálculo del BMI. Haskell por un lado pretende tener una sintaxis matemática debido a sus bases 'lambda'[41], aunque los símbolos que utiliza hacen menos comprensible el código cuando se trata de la lectura del mismo. Ello se comprueba en el resultado de la encuesta realizada.

[41] http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.ndjfl/1093883253

5.2 Fácil de Escribir

Según [Sebesta 10] la simplicidad y ortogonalidad, el soporte para abstracciones y la expresividad son conceptos que propician una fácil escritura de código. La simplicidad y ortogonalidad refiere a formar expresiones y sentencias consistentes a partir de pocas estructuras de datos y construcciones (pequeño número de tokens primitivos y un pequeño conjunto de reglas combinables). El soporte para abstracciones refiere a la capacidad que tiene un lenguaje de programación para definir y usar estructuras complejas u operaciones que permita ignorar detalles. Y la expresividad refiere a un conjunto conveniente de formas de definir operaciones.

Para probar facilidad de escritura de código, se presentan programas en los lenguajes de programación C, C++, Java, Haskell y GO. En la primera parte se utilizan funciones básicas, una función matemática y otra función que permite el manejo de cadenas.

En la segunda parte se presentan programas que permiten la conexión a una base de datos MySQL, que realiza una consulta de los tipos de datos de la fila de una tabla. Y en la tercera parte se presentan programas que permiten la lectura y escritura de un archivo con extensión TXT.

Mediante el uso de las funciones, tanto matemática como el manejo de cadenas, se pretenderá diferenciar cuan simple es el código en determinado lenguaje; además, la ortogonalidad se dara con la combinación de los operadores y tipos primitivos. La abstracción y expresividad serán medidas en los ejemplos de la conexión a la base de datos y el manejo de archivos TXT, y como se conceptualizara en código dichos objetos.

5.2.1. Escritura de funciones

Las funciones son parte de la vida diaria de un programador, con ellas se definen librerías, se realizan cálculos, se automatiza tareas y, además se relacionan directamente al concepto de modularidad, que ayuda a los programadores a tratar con la complejidad de los sistemas de software.[42]

5.2.1.1. Función matemática

A continuación se presenta el código de la función matemática que calcula la suma de cubos de los números del 1 al 10. Tanto para los lenguajes de programación C y Java,

[42] www.inteco.es/file/N85W1ZWFHifRgUc_oY8_Xg

se define la función suma de cubos de la misma manera, la variación en líneas y caracteres empleados, se da con el desarrollo en Haskell.

a) Lenguaje de Programación C y Java

El código que calcula la suma de cubos del 1 a 'n', en los lenguajes de programación C y Java, se define de la siguiente manera:

```
int sum_cubes (int n)
{
      int sum = 0;
      for (index = 1; index < = n; index ++)
            sum += index * index * index;
      return sum;
}
```

1. Se comienza escribiendo el tipo de dato del resultado de la función 'int'.
2. Se define un nombre alusivo al programa 'sum_cubes' y se comprende que se trata de una suma de cubos.
3. Se escribe el parámetro de la función 'n' y su tipo de dato 'int'.
4. Se indica comienzo de bloque que corresponde a la función con el símbolo '{'.
5. Se inicializa la variable sum con '0' y se indica su tipo de dato primitivo 'int'.
6. Se utiliza una estructura 'for', el cual utiliza una sintaxis que utiliza una variable auxiliar 'index', la misma que tiene un tipo de dato y debe ser declarado.
7. Se realiza el cálculo de la suma de cubos, para ello se multiplica el parámetro de entrada tres veces, y se asigna el resultado a la variable sum.
8. Se devuelve el resultado que se guardó en la variable sum, utilizando 'return'.

b) Lenguaje de Programación Haskell

El código que calcula la suma de cubos del 1 al 10, en Haskell es:

```
sum_cubes n = sum (map (^3) [1 .. n] )
```

1. Se comienza escribiendo el nombre alusivo al programa 'sum_cubes' y se comprende que se trata de una suma de cubos.
2. Se escribe el parámetro de la función 'n'.
3. Se realiza el cálculo de la suma con la función predeterminada 'sum' que suma lo que tiene dentro del paréntesis.
4. Se utiliza 'map' y es aplicado a los parámetros que le siguen. 'map' es una función predeterminada que toma una lista y aplica la operación del primer

parámetro '(^3)' a cada elemento de la lista '[1 . . n]' (lista del 1 al n, en este caso n vale 10, definido como input por el usuario), y devuelve una nueva lista.

c) Lenguaje de Programación GO

El código que calcula la suma de cubos del 1 al 10, en GO es:

```
package main

import "fmt"

func main() {
    var sum float32
    for _, x := range []float32{1,2,.10} {
        sum += x*x*x
    }
    fmt.Println(sum)
}
```

1. Se comienza escribiendo la función con el nombre alusivo al programa 'sum_cubes' y se comprende que se trata de una suma de cubos.
2. Se escribe el parámetro de la función 'n', y se tiene que definir el tipo 'int64', asicomo el tipo del resultado de la función 'int64'.
3. Se realiza el cálculo, se guarda el resultado en 't' asignándose con ':='. Finalmente se muestra la salida con la multiplicación de 't' a través de 'return'.

5.2.1.2. Función que maneja cadena de caracteres

Las funciones que tratan cadenas de caracteres mayormente tienen presencia en proyectos de software de seguridad de datos o en proyectos que incluyen módulos de seguridad de datos.

Es tan importante tener presencia en la Web como protegerse en dicho medio y empresas transnacionales como IBM ha publicado Guía de Seguridad IBMi V6.1[43]. Más adelante se presentarán tecnologías que aseguran la integridad de datos en cada uno de los lenguajes de programación.

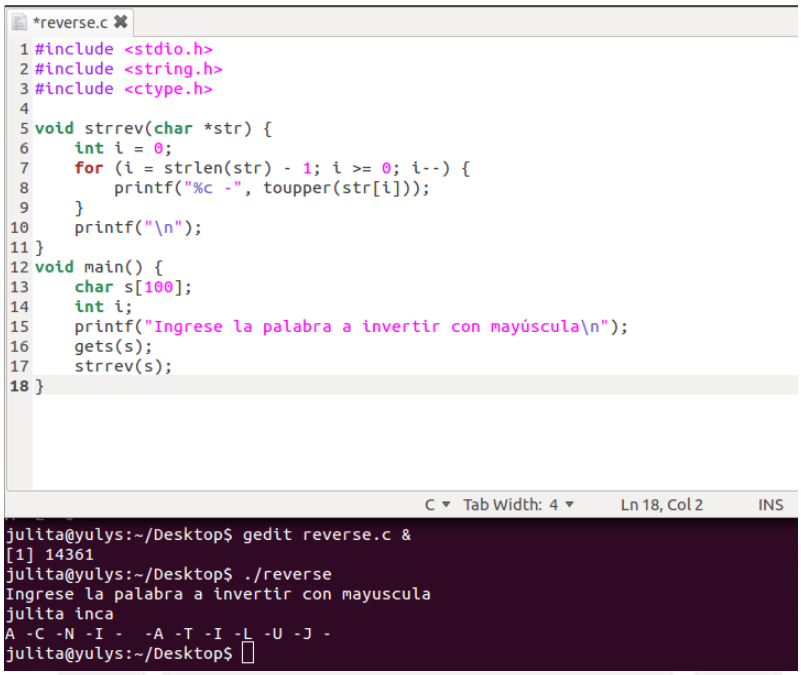
A continuación se probará la escritura de código en los cuatro lenguajes de programación: C, Java, Haskell y GO, con una función que trata las cadenas de

[43] http://www.redbooks.ibm.com/redbooks/pdfs/sg247680.pdf

caracteres. Se invertirá la cadena de caracteres ingresada, la misma que se ingresará en minúsculas y se obtendrá la cadena invertida en mayúsculas.

a) Lenguaje de Programación C

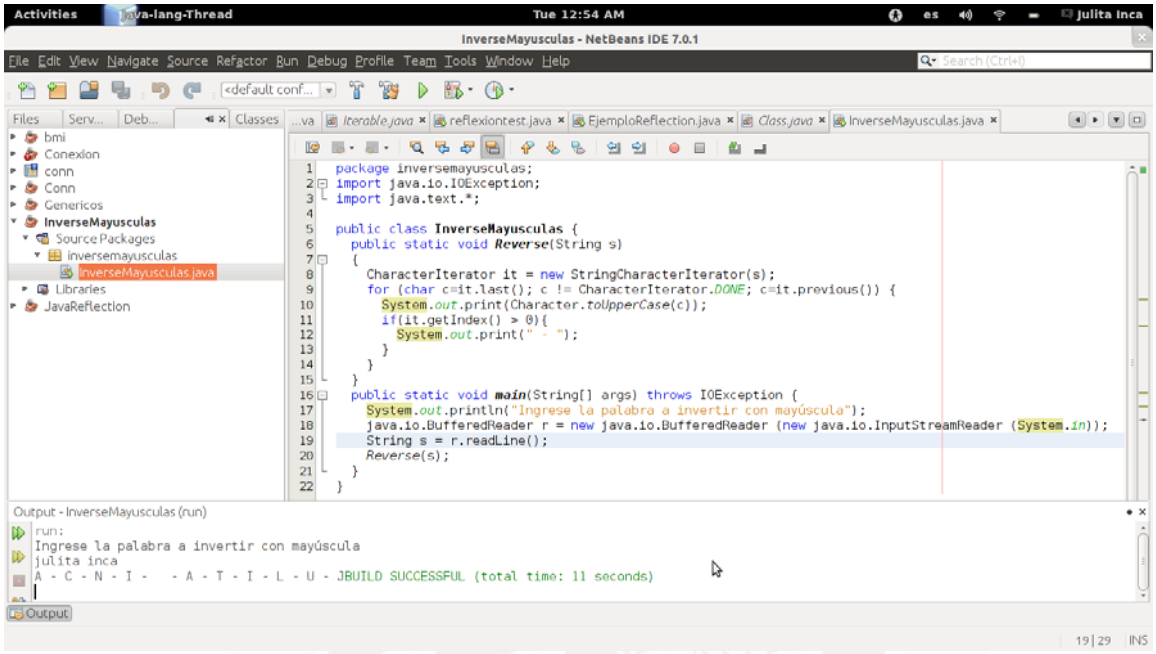
Figura 5.7.: Escritura de código, compilación y ejecución del programa reverse en C.



El código mostrado en la figura e importan la librería stdio.h para utilizar operaciones estándar de entrada y salida como 'printf' y 'gets'; la función string.h para manejar cadena de caracteres: 'strrev'; y ctype.h para operar con caracteres, se utiliza la macro 'toupper'. Declara y desarrolla la función 'strrev' que se encargará de invertir la cadena de caracteres. La función tiene el parámetro 'str', el cual utiliza puntero y para poder manejar cada carácter que conforma una cadena de caracteres. Se utiliza una estructura 'for' para recorrer desde la última posición de la palabra hasta llegar la posición cero e imprime cada carácter en mayúscula. En la función principal se imprime el mensaje para el usuario con 'printf' donde se le indica que 'Ingrese la palabra a invertir en mayúsculas', con 'gets' se captura la entrada almacenada en la variable 's' y se le aplica la función definida 'strrev'.

b) **Lenguaje de Programación Java**

Figura 5.8.: Escritura de código, compilación y ejecución del programa reverse en C.

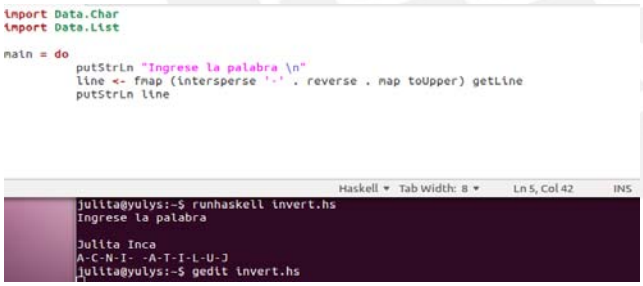


Con Java, se importa la librería java.io.IOException para hacer uso de las excepciones relacionadas a la lectura de datos para la lectura de datos se hace desde consola. La librería java.text.* para hacer uso de las funciones relacionadas a manipulación de textos, se hace uso de la interfaz “CharacterIterator” que nos permite recorrer de manera bidireccional los caracteres de un texto, de la clase “StringCharacterIterator” que implementa la interfaz mencionada para recorrer específicamente cadenas del tipo “String”. Se crea la clase principal del programa, “InverseMayusculas” con 2 funciones: “Reverse” que implementa la funcionalidad de imprimir una cadena de caracteres empezando desde el último carácter hasta el primero, y la función “main” que corresponde a la punto de partida de ejecución. Dentro de la función “Reverse”, se declara una variable “it” del tipo “CharacterIterator”, como este tipo de dato corresponde a una interfaz, se debe instanciar con una clase que implemente dicha interfaz, por lo que se utiliza la clase “StringCharacterIterator” que permite el recorrido de una cadena de caracteres. Mediante una estructura “for” se recorre la cadena de caracteres, utilizando la variable interna “c” para almacenar el caracter que se lee en cada iteración. Dentro de la estructura “for” identificamos 3 elementos: Punto de partida: “it.last()”, el cual posiciona al

iterador en la última posición de la cadena para empezar el recorrido. Se presenta una condición para seguir iterando: “CharacterIterator.DONE”, el cual es un atributo estático de la interfaz “CharacterIterator” que permite representar al iterador cuando llega al final de su recorrido. Otro incremento de la variable de iteración se da con “it.previous()”, función que decrementa al índice interno del iterador y devuelve el caracter en dicha posición. Mediante la función “print” se imprime en consola el caracter leído por el iterador. La función “Character.toUpperCase()” permite convertir un caracter ha mayúsculas. Se añade una condición para imprimir entre los caracteres leídos un guión (“-“) solo cuando el índice interno del iterador sea 0, es decir cuando no hayamos llegado al primer caracter. Dentro de la función “main”, se imprime mediante consola un enunciado pidiendo al usuario ingresar una palabra en mayúsculas para posteriormente invertirla. Luego se declara una variable “r” del tipo “BufferedReader” el cual es una clase que permite leer caracteres de un flujo de datos (consola, un archivo de texto plano, etc.). Al instanciar esta variable observamos que en su constructor enviamos como parámetro “System.in” que corresponde al flujo de datos de entrada estándar, que corresponde a la lectura de datos por teclado. Dentro de la variable “s” del tipo “String” almacenamos toda la línea que ingresó el usuario obtenida mediante la función “r.readLine()”.
Finalmente, invocamos a la función “Reverse” para imprimir la cadena invertida.

c) Lenguaje de Programación HASKELL

Figura 5.9.: Escritura de código, compilación y ejecución del programa reverse en Haskell.



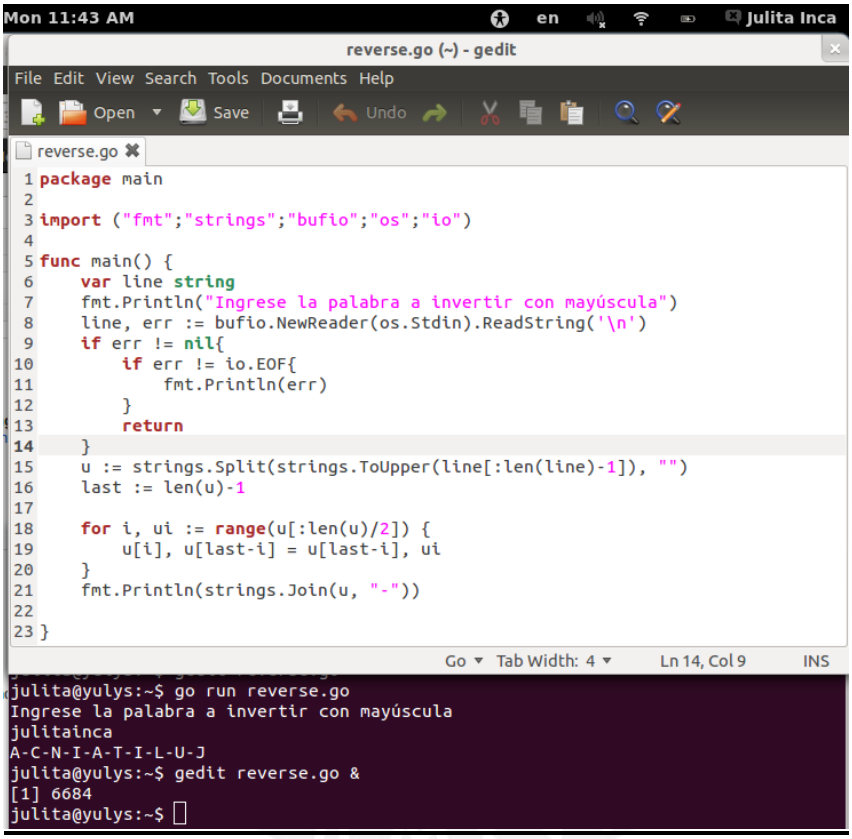
Con Haskell se importan la librería Data.Char para utilizar la función ‘toUpper’ que convierte un caracter a mayúsculas) y Data.List para manejar la lista con ‘reverse’. Por defecto Haskell incluye en sus programas a la librería “Prelude”[44], la misma que ha permitido en este código el uso de las funciones ‘’putStrLn‘, getLine’ y ‘fmap’. Con la palabra reservada ‘do’ se escribe el bloque principal ‘main’, en el cual se define un mensaje para que el usuario ingrese la oración a invertir en mayúsculas con ‘putStrLn’. Se captura la cadena ingresada por el usuario en la variable ‘line’ y se opera con ‘getLine’ y se aplica las operaciones de convertir a mayúsculas ‘toupper’, invertir caracteres

[44] http://www.haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html

'reverse' y finalmente separar caracteres con el carácter '-'. 'map' hace que se aplique a cada uno de los caracteres y la función 'fmap' generaliza a la función 'map' usada previamente. Se muestra impresa la salida en consola el resultado de line con 'putStrLn'.

d) Lenguaje de Programación GO

Figura 5.10.: Escritura de código, compilación y ejecución del programa reverse en GO.



Con GO se importan cinco librerías, se emplean funciones de tratamiento de cadena 'len' y 'toUpper' para convertir a mayúsculas la cadena ingresada. Se asigna con el símbolo ':=', se emplea estructura 'for', asegura que se ingrese alguna cadena valida utilizando 'nil' y 'fmt.Println(err)' para imprimir el error. Para recibir por consola las entrada de datos, line, err := bufio.NewReader(os.Stdin).ReadString y ('\n') para el salto de línea.

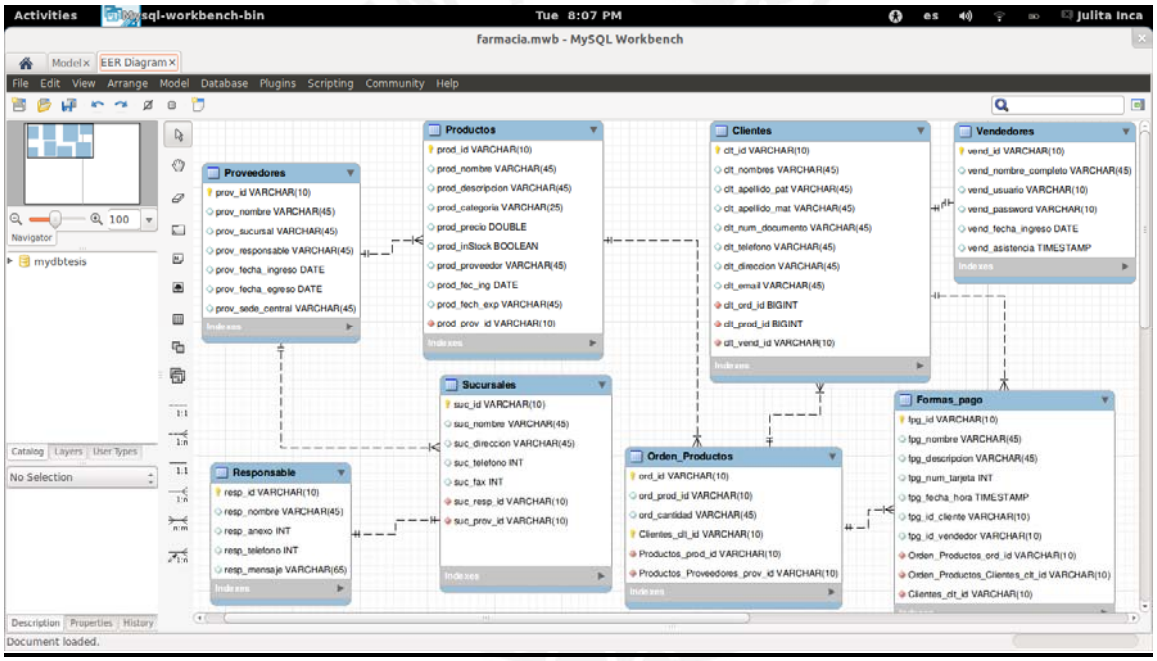
Se puede observar que el código de Haskell es más compacto que el código presentado en los programas en C, Java y GO. Haskell omite símbolos como los paréntesis y puntos y comas al fin de cada sentencia; a cambio de ello, adiciona indentación en su estructura de código. Ademas emplea simplicidad en las librerías y el uso de las funciones que generalmente están inlcuidas en la librería que viene por defecto 'Prelude'. En este caso

se demuestra que Haskell hace más simple la aplicación de funciones a una lista de caracteres.

3.2.2. Escritura de conexión base de datos

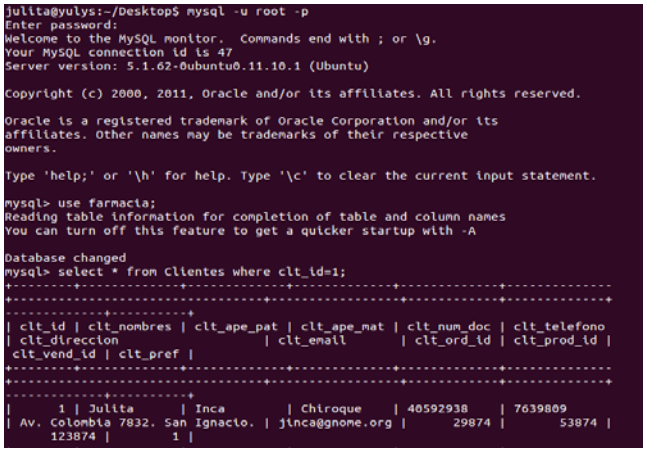
Un programa necesario, de real aplicación, es el que desarrolla una conexión a la base de datos. En este caso, para probar la conexión desde C, Java y Haskell, se ha diseñado tablas de una base de datos farmacia, utilizando MySQL Workbench. La base de datos 'farmacia' contiene las tablas 'Clientes', 'Proveedores', 'Productos', 'Vendedores', etc.

Figura 5.11.: Diagrama de entidad –relación de una farmacia en MySQL Workbench



Las conexiones que se realizarán desde C, Java, Haskell y GO, permitirá visualizar cada uno de los datos de la primera fila de la tabla 'Clientes'; y a su vez mostrará, el tipo de dato de cada una de ellos. Se ha trabajado con un motor de base de datos MySQL.

Figura 5.13.: Consulta de la data en Mysql, el resultado muestra información del cliente



A continuación se presentan los programas de la conexión a la base de datos farmacia en los lenguajes de programación C, Java y Haskell:

Lenguaje de Programación C

```
#include <mysql.h>
#include <stdio.h>
#include <stdlib.h>
char * field_type_name(int);
int main() {
    MYSQL *conn;
    MYSQL_RES *res;
    MYSQL_ROW row;
    MYSQL_FIELD * field;
    int it;

       char *server = "localhost";
       char *user = "root";
       char *password = "root-pass";
       char *database = "farmacia";
       conn = mysql_init(NULL);

       /* abriendo conexión a la base de datos */
    if (!mysql_real_connect(conn, server, user, password, database, 0, NULL, 0)) {
        fprintf(stderr, "%s\n", mysql_error(conn));
        exit(1);    }
    if (mysql_query(conn, "SELECT * FROM client")) {
        fprintf(stderr, "%s\n", mysql_error(conn));
        exit(1);
    }
    res = mysql_use_result(conn);
    printf("\n");
    field = mysql_fetch_field(res);
    while ((row = mysql_fetch_row(res)) != NULL) {
        for (it = 0; it < 3; ++it) {
          printf("%s\t%s\t%s\n",field[it].name,field_type_name(field[it].type),row[it]);
```

```
            //printf("%s\t%s\n", row[0], row[1], row[2]);
        }
    }
    /* cerrando conexión */
    mysql_free_result(res);
    mysql_close(conn);
    exit(0);    }

char * field_type_name(int key) {
    switch (key) {
    case MYSQL_TYPE_TINY:
        return "TINYINT";
    case MYSQL_TYPE_SHORT:
        return "SMALLINT";
    case MYSQL_TYPE_LONG:
        return "INT";
    case MYSQL_TYPE_INT24:
        return "MEDIUMINT";
    case MYSQL_TYPE_LONGLONG:
        return "BIGINT";
    case MYSQL_TYPE_DECIMAL:
        return "DECIMAL";
    case MYSQL_TYPE_NEWDECIMAL:
        return "DECIMAL";
    case MYSQL_TYPE_FLOAT:
        return "FLOAT";
    case MYSQL_TYPE_DOUBLE:
        return "DOUBLE";
    case MYSQL_TYPE_BIT:
        return "BIT";
    case MYSQL_TYPE_TIMESTAMP:
        return "TIMESTAMP";
    case MYSQL_TYPE_DATE:
        return "DATE";
    case MYSQL_TYPE_TIME:
        return "TIME";
    case MYSQL_TYPE_DATETIME:
        return "DATETIME";
    case MYSQL_TYPE_YEAR:
        return "YEAR";
    case MYSQL_TYPE_STRING:
        return "CHAR";
    case MYSQL_TYPE_VAR_STRING:
```

```
        return "VARCHAR";
    case MYSQL_TYPE_BLOB:
        return "BLOB";
    case MYSQL_TYPE_SET:
        return "SET";
    case MYSQL_TYPE_ENUM:
        return "ENUM";
    case MYSQL_TYPE_NULL:
        return "NULL";
    default:
        return "OTHER";    }
}
```

El código presentado es una adaptación al código que accede a una base de datos en general[45]. Se ha trabajado con la librería 'mysql.h'[46] y se han empleado las funciones correspondientes para establecer la conexión 'mysql_real_connect'.

Corrida del programa.-

Figura 5.14.: Resultado de la corrida del programa de conexión a base de datos en C

```
julita@yulys:~$ time ./connOK

clt_id  BIGINT  1
clt_nombres     VARCHAR Julita
clt_ape_pat     VARCHAR Inca
clt_ape_mat     VARCHAR Chiroque
clt_num_doc     VARCHAR 40592938
clt_telefono    VARCHAR 7639809
clt_direccion   VARCHAR Av. Colombia 7832. San Ignacio.
clt_email       VARCHAR jinca@gnome.org
clt_ord_id      BIGINT  29874
clt_prod_id     BIGINT  53874
clt_vend_id     BIGINT  123874
clt_pref        TINYINT 1
CPU usage: User = 0.008000,System = 0.000000

real    0m0.011s
user    0m0.008s
sys     0m0.000s
```

Lenguaje de Programación Java

La página oficial de Java presenta las tecnologías de bases de datos de Java SE.[47] JavaDB, Java Data Objects (JDO) y The Java Database Connectivity (JDBC):

- Java DB es una distribución que soporta bases de datos Apache Derby.

[45] http://www.pmoghadam.com/homepage/HTML/c-mysql.html
[46] http://dev.mysql.com/doc/refman/5.0/en/c.html

[47] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html

- JDO, desarrollado por Apache JDO, proporciona varias opciones para la persistencia. Ejemplo: de RDBMS, a OODB, a los archivos.
- El API JDBC proporciona una API de nivel-llamadas para el acceso a la base de datos basada en SQL. Para Java SE 7, se tiene la versión 4.1 de JDBC.[48]

En este caso se hará uso de las librerías de Java java.sql.Connection, java.sql.DriverManager, java.sql.ResultSet, java.sql.ResultSetMetaData y java.sql.Statement, que facilitarán la escritura del código en el lenguaje de programación Java para establecer una conexión a la base de datos. Se ejecuta una sentencia SQL contra la tabla 'Clientes' de la base de datos y el resultado de la consulta será almacenada en un conjunto de datos (ResultSet). En la estructura de la lista genérica, el programa recorre la lista elemento por elemento e imprime el tipo de dato de la columna correspondiente al dato obtenido y el valor del dato. Por ejemplo: BIGINT – 123.

Se emplea Connection en una excepción que contiene funciones básicas de conexión y comunicación con una base de datos, mediante una cadena de conexión en donde se incluye el nombre del servidor, la base de datos, el usuario y la contraseña.

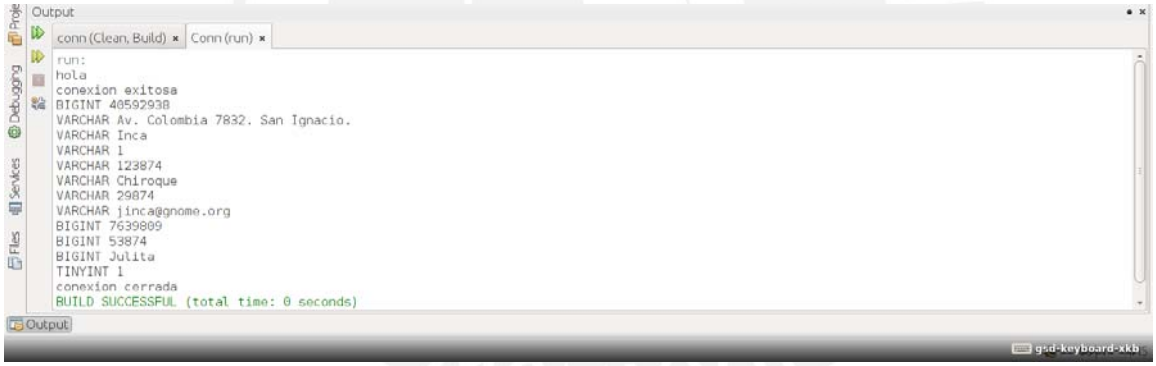
```
package edu.tesis.julita;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class Main {
    public static void main(String args[]) {

        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost/farmacia", "root", "root-
pass");
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery("SELECT * FROM client");
            ResultSetMetaData rsmd = rs.getMetaData();
            int numCol = rsmd.getColumnCount();
            while (rs.next())
            {
                for (int i = 1; i <= numCol; i++) {
                    System.out.println(
```

[48] http://download.java.net/jdk8/docs/technotes/guides/jdbc/jdbc_41.html

```
                                rsmd.getColumnTypeName(i)

                                + " " + rs.getString(i));  }
                }
                rs.close();
                st.close();
                conn.close();

            } catch (Exception ex) {
                System.out.println(ex.getMessage());

              }

        }
```

Corrida del programa.-

Figura 5.15.: Resultado de la corrida del programa de conexión a base de datos en Java



Lenguaje de Programación HASKELL

Para poder realizar la prueba se han realizado los siguientes pasos:

1. Se deben instalar algunos paquetes previamente:

- Instalar cabal-install
- Instalar libghc-haskelldb-hdbc-dev (headers del haskell y hdbc)
- Instalar libmysqlclient-dev (headers del mysqlclient)

2. Descargar el hdbc-mysql
   git clone git://github.com/bos/hdbc-mysql.git

3. Compilar

cd hdbc-mysql

runhaskell Setup configure

runhaskell Setup build

runhaskell Setup install

4. Probar ejemplo

Compilar el archivo con make:

ghc -idist/build -lmysqlclient --make conn

Opcionalmente, se puede correr con runhaskell

runhaskell conn.hs

A continuación el programa conn.hs en Haskell:

Figura 5.16.: Desarrollo del programa de conexión a base de datos en Haskell

```
module Main where
import Control.Monad
import Database.HDBC
import Database.HDBC.MySQL

connectDatabase :: IO Connection
connectDatabase = connectMySQL defaultMySQLConnectInfo
                  {
                                  mysqlHost     = "127.0.0.1",
                                  mysqlUser     = "root",
                                  mysqlPassword = "root-pass",
                                  mysqlPort = 3306,
                                  mysqlDatabase = "farmacia"
                }

go :: IO ()
go = do conn <- connectDatabase
*
        putStrLn $ "driver " ++ (show $ hdbcDriverName conn)
        putStrLn $ "server version " ++ (show $ dbServerVer conn)

        rows <- quickQuery' conn "SELECT * from Clientes" []
        forM_ rows $ \row -> do
                      putStrLn $ show row

main :: IO ()
main = handleSqlError (replicateM_ 1 go)
```

Corrida del programa.-

Figura 5.17.: Corrida en consola del programa de conexión a base de datos en Haskell

```
julita@yulys:~$ time runhaskell conn.hs
driver "mysql"
server version "5.1.62-0ubuntu0.11.10.1"
[SqlInt64 1,SqlByteString "Julita",SqlByteString "Inca",SqlByteString "Chiroque",SqlByteString "40592938",SqlByteString "7639809",SqlByteString "Av. Co
lombia 7832. San Ignacio.",SqlByteString "jinca@gnome.org",SqlInt64 29874,SqlInt64 53874,SqlInt64 123874,SqlInt32 1]

real    0m1.411s
user    0m0.432s
sys     0m0.064s
julita@yulys:~$
```

Lenguaje de Programación GO

```
package main

import (
        _ "code.google.com/p/go-mysql-driver/mysql"
        "database/sql"
        "fmt"
        "os"
)

type Column struct {
        Name, DataType string
}

func ClienteColumns(db *sql.DB) ([]Column, error) {
        const sqlColumns = `
                SELECT column_name, data_type
                FROM information_schema.columns
                WHERE table_name = 'Clientes'
                AND table_schema = 'farmacia'
                ORDER BY ordinal_position
        `
        var cols []Column
        rows, err := db.Query(sqlColumns)
        if err != nil {
                return nil, err
        }
        for rows.Next() {
                var col Column
                err = rows.Scan(&col.Name, &col.DataType)
                if err != nil {
                        return nil, err
                }
                cols = append(cols, col)
        }
        err = rows.Err()
        if err != nil {
                return nil, err
        }
        return cols, nil
}

func ClienteRows(db *sql.DB) error {
        cols, err := ClienteColumns(db)
        if err != nil {
                return err
        }
```

```
        var sqlSelectExpr string
        sep := ""
        for _, col := range cols {
                sqlSelectExpr += sep + " " + col.Name
                sep = ","
        }
        sqlClientes := `SELECT ` + sqlSelectExpr + ` FROM Clientes ORDER BY
clt_id`
        rows, err := db.Query(sqlClientes)
        if err != nil {
                return err
        }

        var data = make([]string, len(cols))
        var i int
        for i = 0; rows.Next(); i++ {
                err = rows.Scan(
                  &data[0], &data[1], &data[2], &data[3], &data[4], &data[5],
                   &data[6], &data[7], &data[8], &data[9], &data[10],
                   &data[11],
                )
                if err != nil {
                        return err
                }
                fmt.Println("Row:", i+1)
                for i, datum := range data {
                        fmt.Println(cols[i].Name, cols[i].DataType, ":", datum)
                }
        }
        err = rows.Err()
        if err != nil {
                return err
        }
        fmt.Println("Total Rows: ", i)
        return nil
}

func main() {
        db, err := sql.Open("mysql", "root-pass@tcp(127.0.0.1:3306)/farmacia")
        if err != nil {
                fmt.Println(err)
                os.Exit(1)
        }
        defer db.Close()

        err = ClienteRows(db)
        if err != nil {
                fmt.Println(err)
                os.Exit(1)     }
}
```

Corrida del programa.-

Figura 5.18.: Corrida del programa de conexión a base de datos en GO

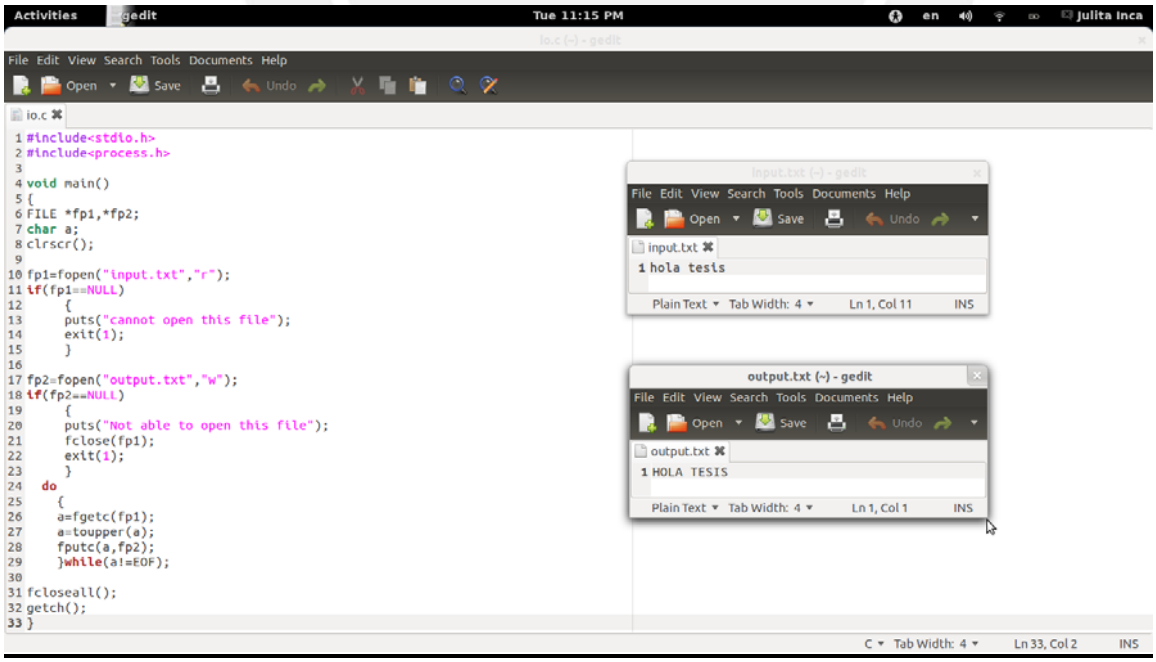
```
julita@yulys:~$ time go run conn.go
Row: 1
clt_id bigint : 1
clt_nombres varchar : Julita
clt_ape_pat varchar : Inca
clt_ape_mat varchar : Chiroque
clt_num_doc varchar : 40592938
clt_telefono varchar : 7639809
clt_direccion varchar : Av. Colombia 7832. San Ignacio.
clt_email varchar : jinca@gnome.org
clt_ord_id bigint : 29874
clt_prod_id bigint : 53874
clt_vend_id bigint : 123874
clt_pref tinyint : 1
Total Rows:  1

real    0m0.435s
user    0m0.352s
sys     0m0.064s
```

4.2.3. Escritura de manejo de archivos

Otra aplicación práctica y real es la escritura de un programa que lee archivos del exterior, y puede modificar data en él. Para poder hacer la prueba, se leerá un archivo con extensión TXT.

Lenguaje de Programación C

Figura 5.19: Programa C, abre el contenido de un archivo y devuelve su contiene en mayúsculas

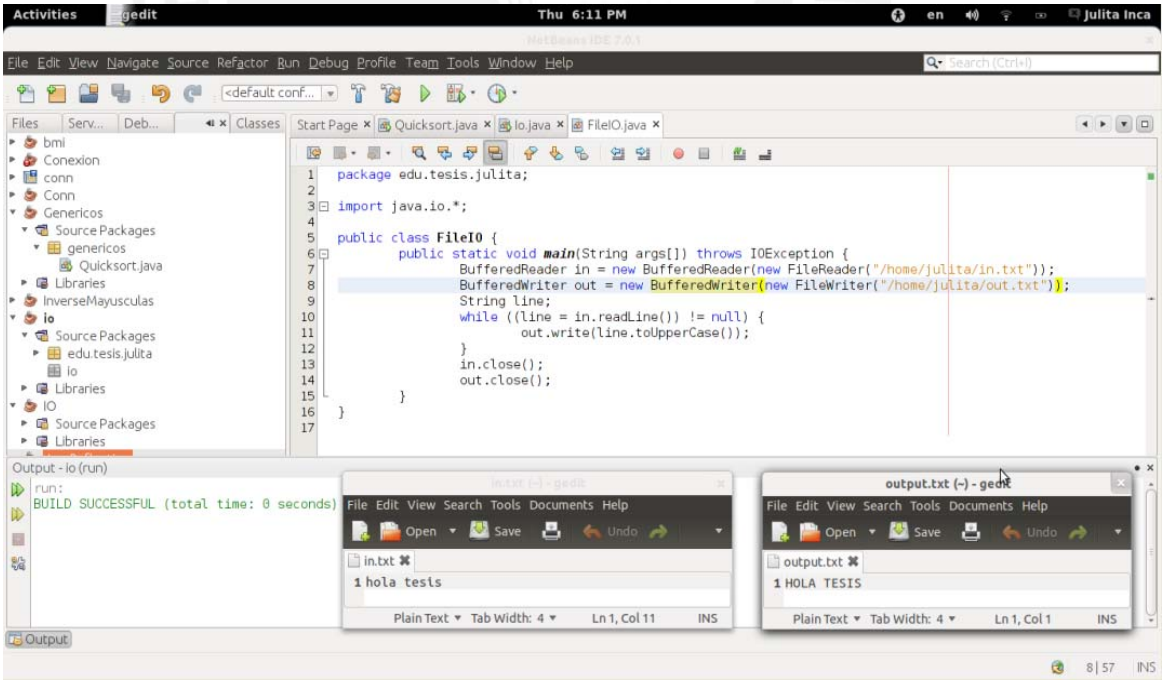


Adaptación de código de la siguiente fuente:
http://www.c4learn.com/c-program-to-convert-the-file-contents-in-upper-case-write-contents-in-a-output-file.html

1. Se declaran las librerías ‘stdio.h’ donde se han definido funciones como ‘fopen’, ‘fcloseall’[49], y la librería ‘process.h’ que controla sistema y en este caso ‘exit(1)’[50].
2. Se declara dos variables tipo archivo ‘fp1’ y ‘fp2’, el primero se abre tipo lectura ‘r’ y el segundo tipo escritura ‘w’. Se declaran con estructura puntero para poder leer caracter por caracter.
3. Se lee caracter por caracter el primer archivo y se almacena cada uno de estos caracteres. Cada uno de ellos es convertido a mayúscula para el segundo archivo, utilizando ‘fgetc’ y ‘fputc’, hasta que se llega al final del primer archivo ‘EOF’.
4. En el caso de no poderse abrir alguno de los dos archivos se terminará el programa con el mensaje que no se pudo abrir el archivo correspondiente.

Lenguaje de Programación Java

Figura 5.20.: Programa Java, abre contenido de un archivo y devuelve su contiene en mayúsculas



[49] http://man.he.net/man3/fcloseall

[50] http://www.cs.cf.ac.uk/Dave/C/node22.html

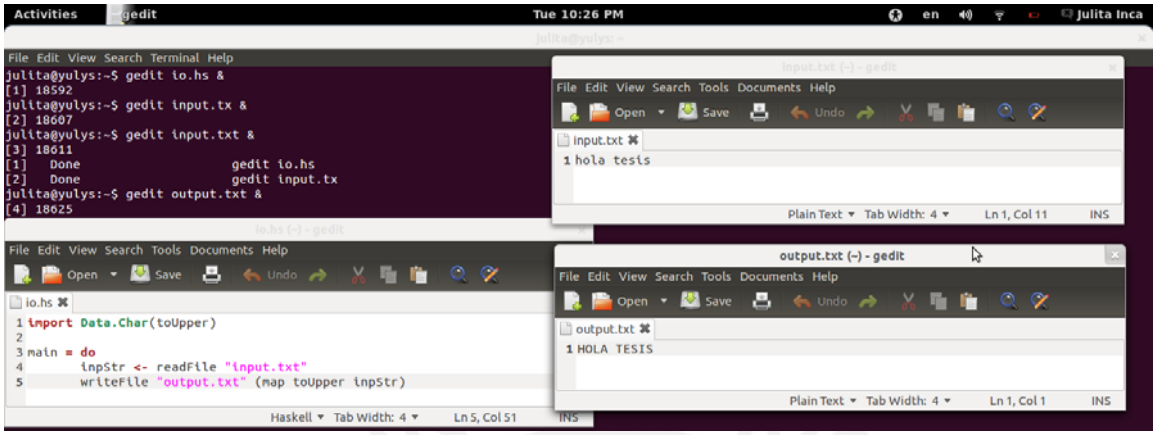
En este código presentado en el lenguaje de programación Java, se declara la librería que nos ayudará con el manejo de archivos: java.io.*.

En este caso se abre el primer archivo, siguiendo la ruta brindada: '/home/julita/in.txt'. El contenido que se obtiene luego de la ejecución de la sentencia es guardado en la varaiable 'in', definido como BufferedReader, y el resultado final se almacenará en la variable 'out', definida como BufferedWriter, dada la ruta de almacenamiento de la salida '/home/julita/out'.

En el código se emplea la estructura 'while' para leer línea por línea el primer archivo y luego se toma cada línea y se pasa por el método 'toUpperCase', convirtiendo caracter por caracter de esa cadena (la línea anteriormente obtenida) a mayúscula.

Lenguaje de Programación HASKELL

Figura 5.21.: Programa Haskell, abre contenido de un archivo y devuelve contiene en mayúsculas



Fuente del código probado: http://book.realworldhaskell.org/read/io.html

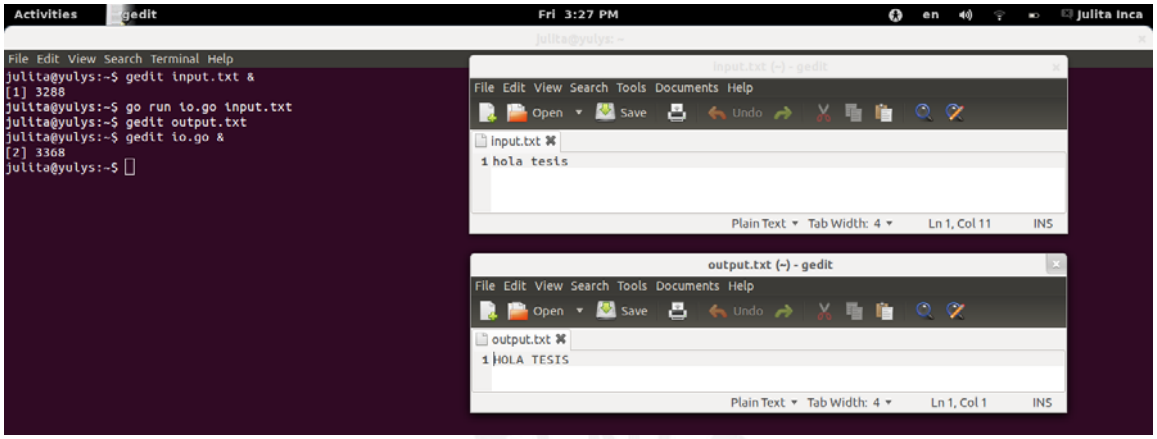
En este código, presentado en el lenguaje Haskell, se importa la librería 'Data.Char' para poder emplear la función 'toUpper', la misma que convertirá el contenido de las palabras en mayúsculas. Se lee el archivo 'input.txt' con la función 'readFile', función definida en la librería que viene por defecto en Haskell 'Prelude', y se almacena en 'inpStr'. Finalmente, se imprime el contenido convertido en mayúscula en el archivo 'output.txt' con 'writeFile', también definido en la librería 'Prelude'. Se aplicó la función 'map' a 'toUpper' para poder aplicar a cada uno de los caracteres almacenados en 'inpStr'.

Lenguaje de Programación GO

```
package main

import (
    "fmt"
    "io/ioutil"
    "strings")

func main() {

  b, err := ioutil.ReadFile("input.txt")
    if err != nil {
        fmt.Println(err)
        return
    }
    r := []byte(strings.ToUpper(string(b)))

    if err = ioutil.WriteFile("output.txt", r, 0666); err != nil {

        fmt.Println(err)     }
```

}

Figura 5.22.: Corrida en GO, abre contenido de un archivo y devuelve contiene en mayúsculas



Evaluación de escritura de código

Siguiendo las pautas de evaluación de escritura de código según el autor [Sebesta 10], se tiene lo siguiente:

1. La expresividad se da en los cuatro lenguajes de programacion. La simplicidad es más notoria en el caso de Haskell que en los casos de C, Java y GO. Esto ocurre porque Haskell ha sido diseñado en base a teorías matemáticas y la expresión se asemeja a una expresión matemática que define los rangos y dominios de una función.

   Si los programadores son o han sido estudiantes de ciencias de la computación o carrera a fin, cuentan con buenos cimientos matemáticos, y es muy probable que se puedan familiarizar con la sintaxis de Haskell. Se ha debido tener un previo conocimiento de las palabras reservadas y funciones predeterminadas como 'map' para lograr obtener un código conciso.

   En los caso de C y Java toma un tiempo mayor para la escritura de cada uno de los símbolos del código. La orotogonalidad es expresada en los cuatro lenguajes de programación. En C, Java y GO se declaran de cada uno de los tipos de las variables que se utilizan en la función; mientras que en Haskell se omiten declaraciones de tipos. En C y Java se tiene la estructura de control 'for', en la cual los parámetros están entre paréntesis separados por punto y comas, mientras que en Haskell tan sólo se escribe el nombre de la función y el parámetro a utilizar, en este caso 'n'. No son menos líneas de código pero si se

dan menos caracteres en el programa. Haskell, se presenta una sintaxis más simple que da una apariencia de código compacto y ordenado. GO presenta el cálculo directo y usa return como C.

Por otro lado, la simplicidad y compactación del código en Haskell puede resultar también costosa al momento de hacer mantenenimieno del software. Si otros programadores leen el código, y no se tiene declaraciones explicitas de los tipos de las variables y expresiones en el programa, tendrán que inferirlas tal como lo hace el compilador de Haskell y eso toma tiempo durante la revisión del programa.

2. El soporte para las abstracciones, se da en los cuatro lenguajes de programación. En los cuatro casos se escribe y se comprende por el nombre de la función, que se trata de una suma de cubos. Pero al tratar de comprender cada uno de los símbolos que se utilizan en el programa, tanto en el caso de C, Java, Haskell y GO, se trata de dar un significado a cada una de las formas planteadas, y esto implica un tiempo y costos de aprendizaje y mantenimiento.

Los cuatro lenguajes de programación permitieron expresar la reversión de una cadena de caracteres, la conexión a base de datos y el manejo de archivos de entrada y salida. C es flexible y permite expresar casi cualquier código de bajo nivel. Al ser un lenguaje extremadamente permisible, es muy fácil incurrir en errores de sintaxis. Java y Haskell tienen ventaja sobre ello porque son de mayor nivel y manejan algoritmos eficientes para expresar abstracciones de tipos.

C es un lenguaje relativamente de bajo nivel, caracterizado por las operaciones de manipulación de bits y por el uso universal de los punteros para conseguir el efecto de los parámetros de referencias y para definir los tipos recursivos. Java es de alto nivel, evitando toda manipulación de puntero explícito. (Todos los objetos son implícitamente acceder a través de punteros, pero es sólo evidente en la semántica de referencia del lenguaje).

En el caso de Haskell, permite fácil escritura pero al comienzo es difícil de dominar, requiere mucha experiencia el uso correcto a nivel profesional y competitivo. Aunque Haskell presenta simplicidad en la escritura con el uso de símbolos, el costo de ello como contraparte se da la comprensión de lectura de los mismos. Muchas veces los tokens en Haskell no son muy claros para un programador y da la apariencia de leer un código casi críptico.

Ejemplo:

```
levenshtein::String->String->Int
levenshtein s t =
    d!!(length s)!!(length t)
    where d = [[distance m n|n<-[0..length t]]|m<-[0..length s]]
          distance i 0 = i
          distance 0 j = j
          distance i j = minimum [d!!(i-1)!!j+1, d!!i!!(j-1)+1,
d!!(i-1)!!(j-1) + (if s!!(i-1)==t!!(j-1) then 0 else 1)]
```

Este código refiere a la Distancia de Levenshtein, distancia de edición, o distancia entre palabras, que devuelve el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra.

La distancia de Levenshtein entre "casa" y "calle" es de 3 porque se necesitan al menos tres ediciones elementales para cambiar uno en el otro.

5.3 Confiabilidad

Según [Sebesta 10], C es muy poco fiable porque establece numerosas trampas para los incautos programadores. El sistema de tipo de C es débil porque permite sin problemas conversiones de tipo y de puntero aritmético, y no hace cumplir los controles de tipo entre las unidades de compilación, exponiendo así a los programas a una variedad de errores de tipo en tiempo de ejecución.

Por otro lado, soporte de re-uso, no ambigüedad y manejo de errores son otros factores relacionados con la confiabilidad de un lenguaje de programación. [51]

A continuación se prueba la confiabilidad de los sistemas de tipos de los tres lenguajes de programación C, Java y Haskell. La primera evaluación se da con una operación de división entre dos números ‘enteros’, donde se sabe que el resultado debe arrojar un número decimal ‘float’. La segunda prueba consiste en verificar los errores esperados que afectaría la integridad de datos en el programa Corrector Ortográfico.

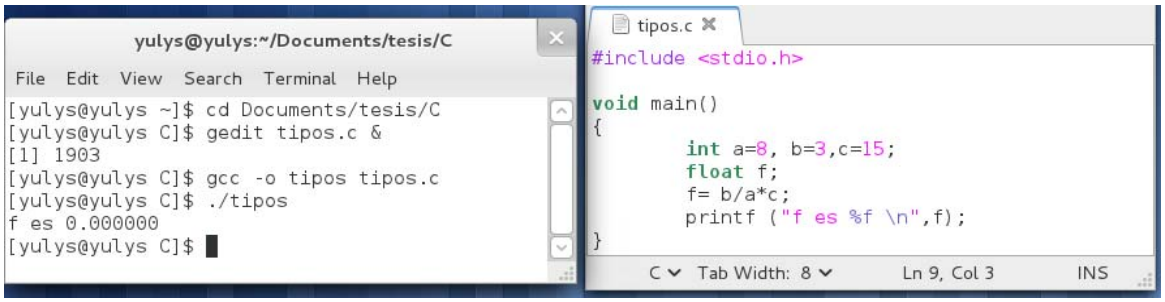
4.3.1. Tipos de Datos

Se sabe que la división de dos números enteros devuelve como resultado un número decimal. En este caso se define una variable f, veamos el resultado obtenido.

[51] www.cs.pdx.edu/~harry/musings/RelLang.pdf

Lenguaje de Programación C

Figura 5.23.: Desarrollo y corrida de un programa que divide enteros y devuelve un número real



1. Se ha definido las variables a, b y c; cada una de estas variables tiene un valor predefinido, en el caso de a, vale 8; b, vale 3; c, vale 15; definidos con tipo 'int'.

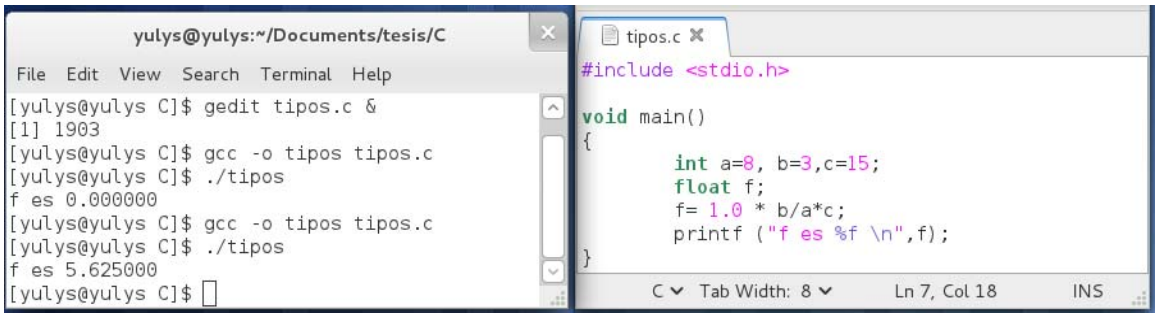
2. Se define f como 'float'.

3. Se imprime 'f', luego de haber realizado el cálculo correspondiente 'b/a*c'.

4. Luego de compilar y correr el programa, el resultado de 'f es 0.000000'. El resultado para la maquina es correcto, lo incorrecto es el uso de los tipos de datos y el cast.

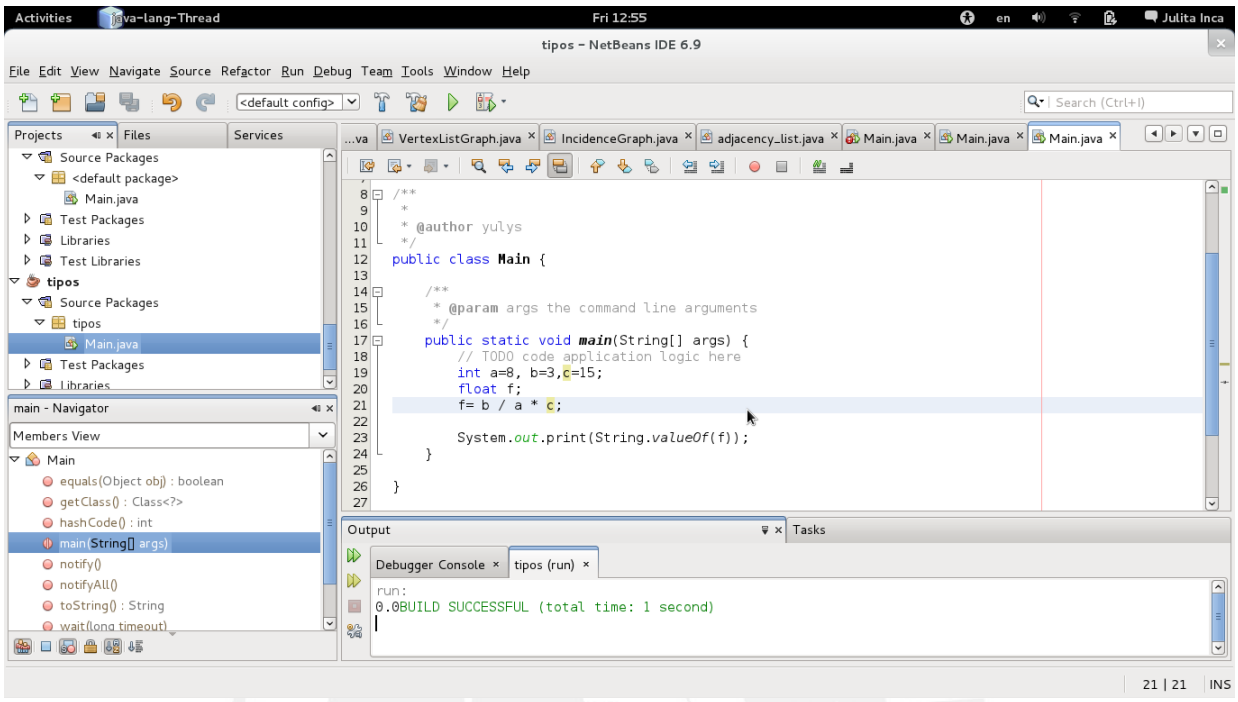
* Si se modifica el programa 'tipos.c' en el cálculo de 'f' y multiplicamos la expresión matemática por un número flotante '1.0', se obtiene una respuesta correcta.

Figura 5.23.: Desarrollo y corrida de un programa que divide enteros, aplicando casteo



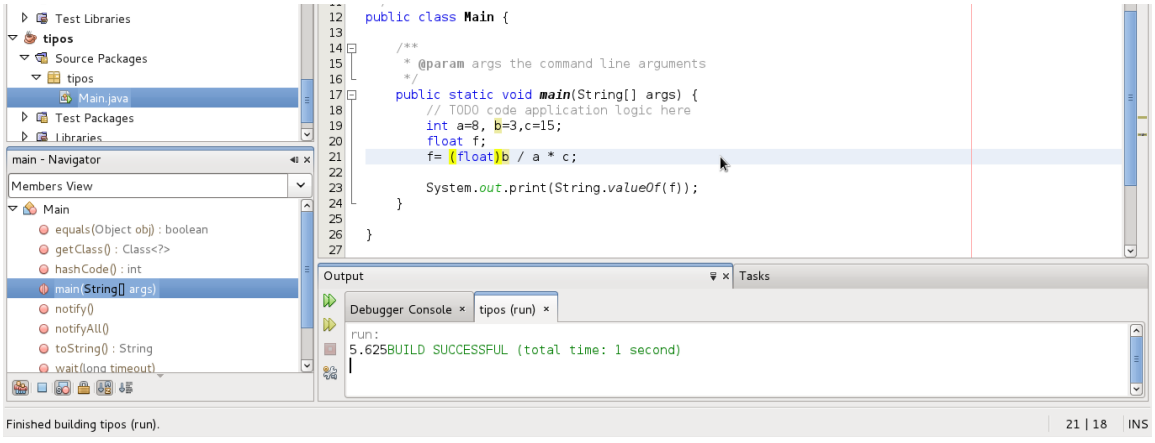
Lenguaje de Programación JAVA

Figura 5.24.: Desarrollo y corrida de un programa que divide enteros



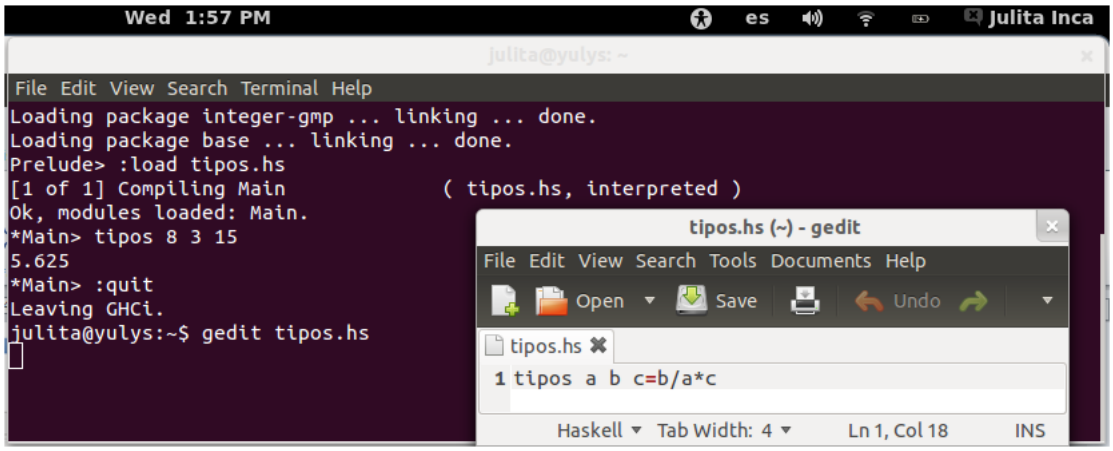
1. Se ha definido las variables a, b y c; cada una de estas variables tiene un valor predefinido, en el caso de a, vale 8; b, vale 3; c, vale 15; definidos con tipo 'int'.
2. Se define f como 'float' y se imprime 'f', luego de haber realizado el cálculo correspondiente 'b/a*c'.
3. Luego de correr el programa, se muestra el resultado de '0.0'. Al igual que el caso anterior, el programador debe tener solido conocimiento del sistema de tipos porque tanto C como Java, respetan la tipificación de los datos.

Figura 5.25.: Desarrollo y corrida de un programa que divide enteros, aplicando casteo



Lenguaje de Programación HASKELL

Figura 5.26.: Desarrollo y corrida de un programa que divide enteros



En este caso no es necesario definir los tipos de las variables ‘a’, ‘b’ y ‘c’; sin embargo, se colocan como parámetros de ‘tipos’ al cual se le asigna el resultado del calculo de ‘b/a*c’ y devuelve un resultado correcto.

Lenguaje de Programación GO

Figura 5.27.: Desarrollo y corrida de un programa que divide enteros en GO, se visualiza error:

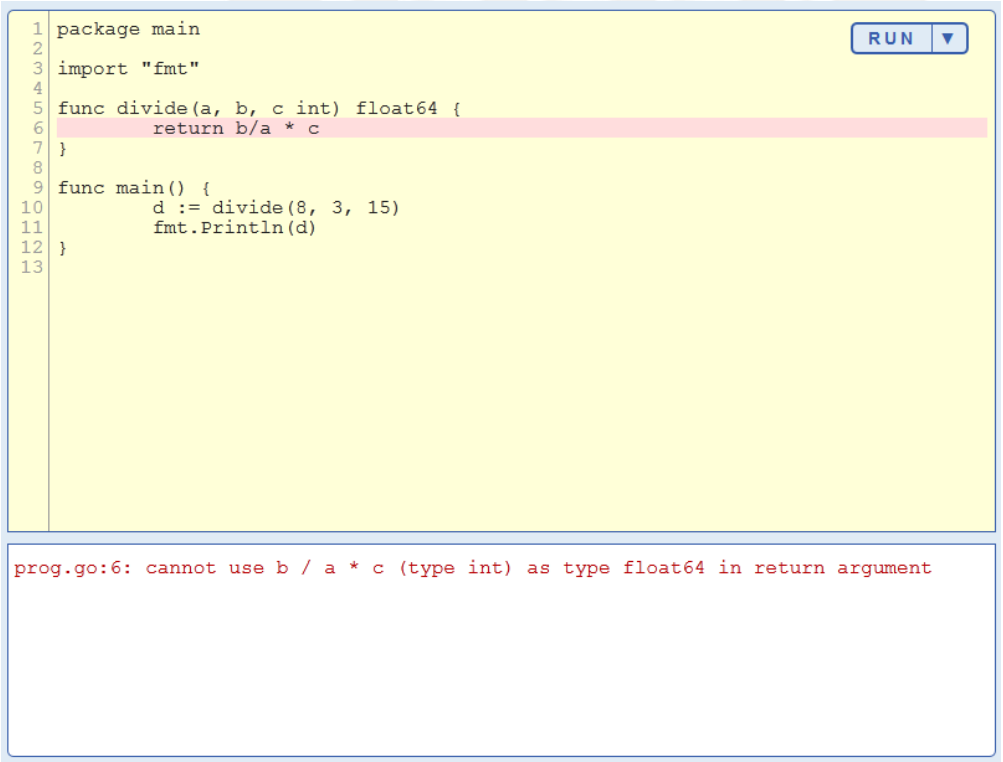


Figura 5.28.: Desarrollo y corrida de un programa que divide enteros en GO, casteando:

```
package main

import "fmt"

func divide(a, b, c int) float64 {
        return float64(b) / float64(a) * float64(c)
}

func main() {
        d := divide(8, 3, 15)
        fmt.Println(d)
}
```

RUN

```
5.625
```

Evaluación de confiabilidad

Se ha verificado que el sistema de tipos de Haskell es más sólido y más confiable que en los casos de Java y C. El sistema de tipos de Haskell determina e infiere el tipo de cada una de las variables y expresiones en el código. Tanto en C, como en Java, a medida que el programa se extiende es muy difícil de corregir los errores.Una variable "x" se puede inicializar con cierto valor y luego al ser procesado, puede cambiar de valor (luego se verá que puede ser esto una desventaja).

5.4 Reflexión

El concepto de reflexión aparece en los años 80, un estudio de ello aplicado al lenguaje de programación LISP es presentado en una tesis doctoral [Cantwell 82].

Pasan los años, pero el término aún no es muy bien comprendido, es así que un estudio de [Maes 87] hace hincapié en la definición y conceptos relacionados al tema. Define reflexión computacional como el comportamiento exhibido por un sistema reflexivo, donde

un sistema computacional que se refleja a sí mismo en una conexión causal. Un sistema computacional es un sistema basado en una computadora cuyo propósito es responder preguntas y/o soportar acciones en algún dominio específico, entonces el sistema está relacionado a dicho dominio. Ello implica incorporar estructuras internas que representen ese dominio. Estas estructuras incluyen datos que representen entidades y relaciones en el dominio y un programa escrito que indique cómo estos datos deben ser manipulados. La computación resulta de la ejecución del programa por parte del procesador. Cualquier programa corriendo es un ejemplo de un sistema computacional. Un sistema es conectado causalmente a su dominio si la estructura interna y el dominio que representa son vinculados de tal manera que si uno de ellos cambia, conlleva a un correspondiente efecto en el otro. Ejemplo, un sistema que controla el brazo de un robot incorpora estructuras que representan la posición del brazo. Estas estructuras podrían estar causalmente conectadas a la posición del brazo del robot tal que: (i) si el brazo del robot es movido por una fuerza externa, las estructuras también cambian (ii) si algunas de las estructuras son cambiadas (por cómputo), el brazo del robot se mueve hacia la posición correspondiente. Por eso, un sistema conectado causalmente siempre tiene una representación precisa de su dominio y podría causar cambios en el dominio como un efecto exclusivo de su cómputo.

La reflexión permite mantener rendimiento de estadísticas, propósitos de depuración (agregar extra código y luego borrarlo), ver paso a paso (seguimiento), modificación (sistemas de aprendizaje), optimización y activación (con los daemons[52]).

Computación reflexiva no contribuye directamente a resolver problemas en el dominio externo del sistema. En realidad contribuye a la organización interna del sistema o a su interface con el mundo exterior. Su propósito es garantizar la eficiencia y fluido funcionamiento del objeto de cómputo.

Este concepto de reflexión trae consigo una nueva forma de abstraer el sistema computacional. Se incorpora una parte objeto computacional y una parte reflexiva. La tarea del objeto computacional es resolver problemas y retornar información acerca del dominio externo, mientras que el nivel reflexivo, resuelve problemas y retorna información acerca de un objeto computacional. Para ello, el intérprete del lenguaje tiene que dar a un sistema que está corriendo el acceso a la representación de los datos (aspectos) del mismo sistema. Es así que el sistema tiene la posibilidad de realizar reflexión

52 Un programa o proceso que se queda suspendido en el sistema hasta que sea invocado para realizar su tarea.

computacional incluyendo código que perciba como estos datos deben ser manipulados. Además el intérprete debe garantizar la conexión causal entre estos datos y estos aspectos del sistema que representan, es total. En consecuencia, las modificaciones del sistema son reflejadas en sus estados y cómputos.

Según [Wu 98], este criterio nos sirve de mucho para el caso de ejecución de archivos maliciosos y la detección de los mismos mientras se ejecuta un sistema computacional.

Esta característica es tomada en cuenta al momento de diseñar y mejorar características de un lenguaje de programación.
El diseñador del lenguaje de programación Scala y de los Genéricos en Java para el compilador 2007: Martin Ordesky[53], en el evento Lang.Next, el 02 de Abril del 2012[54], presenta esta característica aplicada al lenguaje de programación Scala y comienza con las semejanzas y diferencias entre un Compilador y Reflexión. Indica que ambos tratan con tipos, símbolos, nombres, árboles, anotaciones, entre otros; indican cuáles son los miembros de una clase de un tipo, los tipos de los miembros de un tipo de base, si dos tipos son compatibles, o si un método es aplicable a algunos argumentos.

Además, Ordesky señala que la reflexión se basa en la información subyacente de una máquina virtual, invoca código pre generado, permite lanzar excepciones, necesita ser segura para los hilos: 'thread-safe', y maneja los tipos como constantes.

Lenguaje de Programación C

No se ha encontrado bibliografía al respecto.

Lenguaje de Programación JAVA

En la página oficial del proyecto Java se puede encontrar toda la documentación acerca de esta propiedad.[55] En la misma página se pueden hallar ejemplos para poder acceder a los campos y métodos de una clase; asimismo como el acceso a los campos y métodos de un determinado objeto. Se puede acceder al contenido de los campos públicos; y aunque es recomendable que los campos de un objeto (de su clase) sean privados, se

[53] http://en.wikipedia.org/wiki/Martin_Odersky
[54] http://channel9.msdn.com/Events/Lang-NEXT/Lang-NEXT-2012/Reflection-and-Compilers

[55] http://docs.oracle.com/javase/tutorial/reflect/

tiene disponibilidad a los métodos accesores (getters) y modificadores (setters) que operan sobre esos campos.

[González 04] publica un ejemplo de un proyecto real, para facilitar los logs de objetos cuyo método toString no daban demasiada información. El autor recomienda el uso de la clase BeanUtils de Apache Commons[56], y presenta la reflexión como introspección. Indica un ejemplo como el "debugueo", un programa que nos ayude a "debuggear" otros programas, describiendo el contenido de cualquier objeto que queramos. Indica que se puede crear una interfaz que debe cumplir los objetos describibles, pero qué ocurre cuando esos objetos no han sido creados por nosotros. A continuación probaremos el código presentado para acceder a los campos y métodos utilizando el API Reflection.

```
package javaapplication;

public class EjemploBean {
  public String nombre = "Keko";
  private String email = "keko@miservidor.es";
  public String direccion = "ubigeo";
  private void setNombre(String s) {
    nombre = s;       }
  protected String getNombre() {
    return nombre;    }
  public void setEmail(String s) {
    email = s;        }
  public String getEmail() {
    return email;     }
  public void setDireccion(String s){
     direccion = s;   }
  public String getDireccion() {
    return direccion;}
}
```

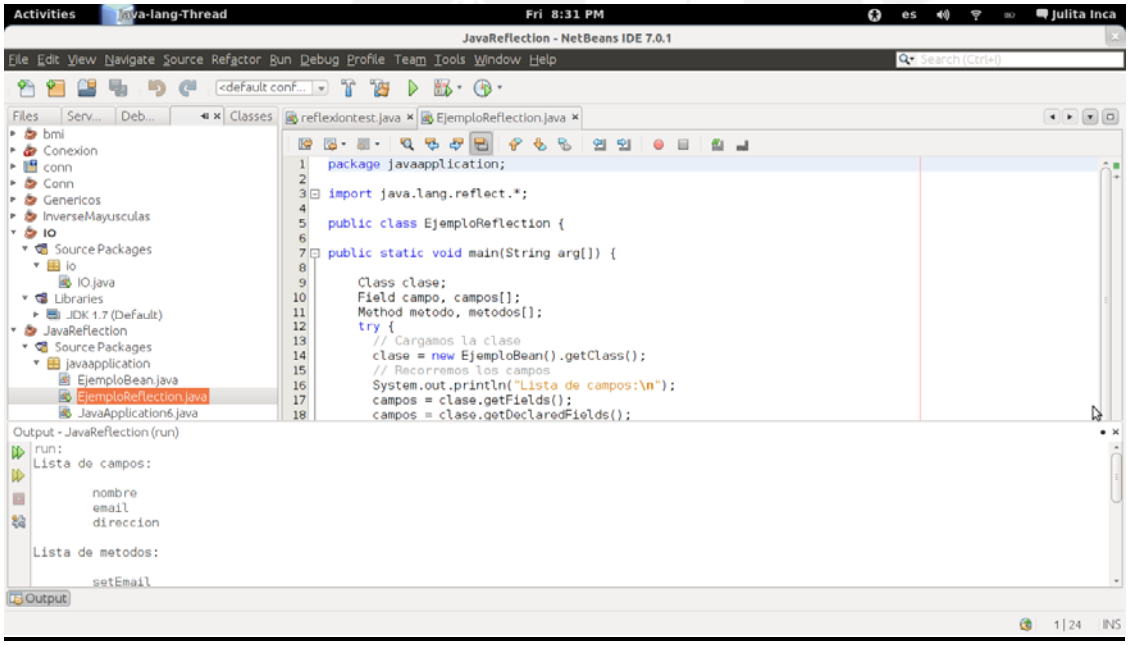
Seguidamente se presenta la clase 'EjemploReflection', la cual realizará el trabajo:

```
package javaapplication;

import java.lang.reflect.*;

public class EjemploReflection {
public static void main(String arg[]) {

    Class clase;

    Field campo, campos[];
    Method metodo, metodos[];
```

[56] http://jakarta.apache.org/commons/beanutils/

```
    try {
      clase = new EjemploBean().getClass();
      System.out.println("Lista de campos:\n");
      campos = clase.getFields();
      campos = clase.getDeclaredFields();
      for (int i=0; i < campos.length; i++) {
        campo = campos[i];
        System.out.println("\t" + campo.getName());
      }
      System.out.println("\nLista de metodos:\n");
      metodos = clase.getMethods();
      for (int i=0; i < metodos.length; i++) {
        metodo = metodos[i];
        System.out.println("\t" + metodo.getName());
      }
    } catch (Exception e) {
    System.out.println("No se ha encontrado la clase. " + e);    }
  }
}
```

Figura 5.29.: Corrida de un programa que aplica Reflexión



El ejemplo presentado puede hacerse sin utilizar reflexión, si y solo si se conoce el nombre de la clase y las propiedades; pero qué sucede si el nombre de la clase es una variable, es entonces que la reflexión juega un rol importante para obtener dicha información.

Otra funcionalidad de la reflexión en Java se puede dar también cuando se pretende instanciar una clase de manera dinámica y ejecutar un método dinámicamente. Un ejemplo se da si se tiene que ejecutar un método, éste debe ser de una clase; y tanto la

clase como el método son 'Strings', se dan como 'Strings', son variables que hay que ejecutar y la reflexión nos ayuda para dicho propósito.
A continuación se presenta un programa con tres clases, la primera es la clase reflexión:

Lenguaje de Programación HASKELL

Según [Venkatreddy 10], Haskell tiene librerías para tipos dinámicos pero aún no soportan reflexión completa.

Según una entrevista online con tres miembros de la comunidad Haskell: Edward Kmett y el colaborador de Google, Mark Lentczner; Haskell es un lenguaje que no tiene la intención de mejorar su capacidad de reflexión porque por cada construcción o avance en ese ámbito, afecta directamente la parametricidad. La parametricidad es una propiedad que tiene mayor valor en Haskell, y es que existe una intrínseca relación inversa entre la poderosa reflexión y otras razonables poderosas características. Edward

Kmett indica "No es que no podemos hacerlo, y es que permitir reflexión arbitraria nos cuesta la capacidad de hacer una amplia gama de las optimizaciones del compilador que no están abiertos a otros lenguajes de programación y nos cuesta la capacidad de razonar sobre el comportamiento del código".

Agrega que "Simon Peyton ha trabajado en 'scrap your boilerplate'[57], que es lo que cuenta como una librería 'genérico transversal'. Ha trabajado en el nuevo API de GHC.Generics, librería de recorrido genérico, siendo ésta toda la reflexión que Haskell puede ofrecer".

[Washburn 07] realiza una investigación acerca de cómo se puede obtener reflexión en Haskell sin arriesgar otras características que ofrece el lenguaje. En este "trade-offs" presenta un poderoso razonamiento e intento de construir un sistema de tipos que permita reflexión y razonamiento paramétrico.

En ejemplo de Reflexión en el lenguaje de programación Java, se demostró que se pueden obtener los campos y métodos de las instancias del programa.
Haskell no cuenta con ningún método, ni clases tradicionales; la única estructura que está allí para reflexionar es el tipo del tipo de datos, el nombre de sus campos y la capacidad de realizar recorridos genéricos. Entonces en el entorno limitado que ofrece, la única

[57] http://research.microsoft.com/en-us/um/people/simonpj/papers/hmap/gmap3.pdf

capacidad de reflexión que tiene sentido son los de recorrido genérico, pero el resto de las capacidades de reflexión de Java estilo simplemente no tienen análogo.

En Haskell estos campos son visibles a través del mecanismo 'Programación genérica'. La libreria de Haskell ‘GHC.Generics‘ brinda acceso para registrar los campos. Esta es la aproximación de comparación análoga que se puede encontrar con la reflexión de Java.

Lenguaje de Programación GO

1. La reflexión va de valor de la interfaz al objeto de reflexión.

En el nivel básico, la reflexión es simplemente un mecanismo para examinar el tipo y el par de valores almacenados dentro de una variable de interfaz. Para empezar, hay dos tipos que necesita saber acerca de reflejar en el paquete: Tipo y Valor. Estos dos tipos de dar acceso a los contenidos de una variable de interfaz, y dos funciones simples, llamados reflect.TypeOf y reflect.ValueOf, recuperar piezas reflect.Type y reflect.Value fuera de un valor de interfaz. (Además, desde el reflect.Value es fácil llegar a la reflect.Type, pero vamos a mantener el valor y los conceptos de tipo separada por ahora.)

```
package main

import (
    "fmt"
    "reflect"
)

func main() {
    var x float64 = 3.4
    fmt.Println("type:", reflect.TypeOf(x))

}
```

La salida del programa es:

```
type: float64
```

Evaluación de reflexión

Cada lenguaje de programación toma la reflexión de diferente manera. Java lo soporta como introspección de objetos para poder obtener propiedades y métodos. Ello peude ayudar a re-construir los programas en el caso que se requiera aplicar de manera inversa el diseño para migrar a otro lenguaje de programación o para poder obtener las entidades erlacion para poder trabajar con bases de datos. GO también soporta reflexión para poder descubrir los tipos con Typeof(). En el caso de C++, cumple esta propiedad pero

con librería externas, no propias del lenguaje de programación de manera oficial. En Haskell no se toma en cuenta esta propiedad debido a que no se tienen estructuras de clases para inspeccionar, como es el caso de Java.

5.5 Soporte para Programación genérica

Según la definición traducida de lo que es Programación Genérica de [Jazayeri 98], se trata de una sub disciplina de las ciencias de la computación que trata de encontrar la representación abstracta de algoritmos eficientes, estructuras de datos, y otros conceptos de software y su organización sistemática. El objetivo de la programación genérica es expresar algoritmos y estructuras de datos bastante aceptables, formas interoperables que permitan su uso directo en construcciones de software. Ideas principales incluyen:

- Expresar algoritmos de con mínimas suposiciones de abstracciones de datos y viceversa, además hacerlos interoperables como sea posible.

- Llevar un algoritmo concreto a un nivel lo general como sea posible sin perder eficiencia, es decir, la forma más abstracta de modo que cuando esté especializada a un caso concreto, el resultado es igual de eficiente que el algoritmo original.

- Cuando el resultado de llevar un algoritmo a código no es lo suficientemente general como para cubrir todos los usos de un algoritmo, adicionalmente de proveer una forma más general, pero garantiza que la forma especializada más eficiente es automáticamente seleccionada cuando sea aplicable.

- Proporcionar más de un algoritmo genérico para el mismo propósito y en el mismo nivel de abstracción, cuando ninguno domina a las demás en la eficiencia para todas las entradas de datos. Esto introduce el término necesidad de proporcionar suficientemente precisas caracterizaciones de dominio para el que cada algoritmo es el más eficiente.

Lenguaje de Programación C

El lenguaje de programación C no tiene soporte de programación genérica, lo que a continuación se presenta es un "hardcoding" de lo que es una programación genérica soportada por los lenguajes de programación Java y Haskell.

Para que C sea genérica no debe ser dependiente del tipo de dato, incluso debería poder soportar structs. En este caso se ha utilizado una estructura 'case' para poder parametrizar el tipo de dato.

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
void **leer (int, char);
void imprimir (void **, int, char);
int intComp (const void *v1, const void* v2);
int floatComp (const void *v1, const void* v2);
int strComp (const void *v1, const void* v2);
void **leer (int num, char tipo) {
int* pInt;
float* pFloat;
char* pCad;
char buffer[500];
void **v;
int i;
    v = malloc(num * sizeof (void*));
    for (i = 0; i < num; i++) {
        printf("Ingrese dato %d: ", i + 1);
        switch (tipo) {
            case 1:
                pInt = (int*) malloc(sizeof (int));
                scanf("%d", pInt);
                v[i] = pInt;
                break;
            case 2:
                scanf("%s", buffer);
                pCad = (char*) malloc(strlen(buffer) + 1);
                strcpy(pCad, buffer);
                v[i] = pCad;
                break;
            case 3:
                pFloat = (float*) malloc(sizeof (float));
                scanf("%f", pFloat);
                v[i] = pFloat;
                break;
        }
    }
    return v;
}
void imprimir(void **datos, int num, char tipo) {
    int i;
    for (i = 0; i < num; i++) {
        switch (tipo) {
            case 1:
                printf("%d\n", *((int*) datos[i]));
                break;
```

```
            case 2:
                printf("%s\n", (char*) datos[i]);
                break;
            case 3:
                printf("%f\n", ((float*) datos[i])[0]);
                break;
        }
    }
}
int intComp(const void *v1, const void* v2) {
    return ((int**) v1)[0][0] - ((int**) v2)[0][0];}
int floatComp(const void *v1, const void* v2) {
    float aux1 = ((float**) v1)[0][0], aux2 = ((float**) v2)[0][0];
    float r = aux1 - aux2;
    return r > 0 ? 1 : (r < 0 ? -1 : 0);
}
int strComp(const void *v1, const void* v2) {
    return strcmp(((char**) v1)[0], ((char**) v2)[0]);
}
int main(int numArg, char **arg) {
    int num;
    int tipo;
    void ** datos;
    printf("Ingrese el número de valores: ");
    scanf("%d", &num);
    printf("Ingrese el Tipo de dato a ordenar\n");
    printf("(1)Entero, (2)Texto y (3)Decimal:\n");
    scanf("%d", &tipo);
    datos = leer(num, tipo);
    switch (tipo) {
        case 1:
            qsort(datos, num, sizeof (void *), intComp);
            imprimir(datos, num, tipo);
            break;
        case 2:
            qsort(datos, num, sizeof (void *), strComp);
            imprimir(datos, num, tipo);
            break;
        case 3:
            qsort(datos, num, sizeof (void *), floatComp);
            imprimir(datos, num, tipo);
            break;
    }
    return 0;
}
```

1. En el módulo principal se ingresan las variables ‘num’ para indicar el número de valores en el arreglo y ‘tipo’ para indicar el tipo de dichos valores, los cuales son parámetros de la función ‘leer’. La función ‘leer’ ha sido definida utilizando el concepto de punteros a funciones.

Un puntero a función es puntero, con una forma de declaración especial, que puede recibir la dirección de inicio del código de una función y ejecutarla a través de él.[58]

2. Se define una estructura 'case' para poder indicar la resolución de acuerdo al tipo de valor de las variables. En este caso se han definido las operaciones para los valores '1' para realizar operaciones con valores de tipo 'entero'; '2' para realizar operaciones con valores de tipo 'string' o cadenas de caracteres; y '3' para realizar operaciones con valores de tipo 'float' o decimales.

3. En cada uno de los casos, se llama a la función predefinida 'qsort'. Esta función ha sido definida en la librería 'stdlib.h' y tiene como parámetros a 'datos' el cual ha sido definido con doble puntero porque puede ser puntero a carácter 'char', y/o puntero a 'int' o dato tipo 'float'; y de acuerdo sea el caso, se destina un espacio de memoria con 'sizeof' definido en la librería 'malloc.h' para cada tipo de dato. A cada variable puntero de tipo como 'pInt' se asigna una dirección valida de manera dinámica con su puntero y casteo correspondiente, en este caso es '(int*)'.

4. Se emplean cada una de las funciones predefinidas para que se puedan comparar los datos según su tipo. 'intComp' que comprara enteros; 'floatComp' para comparar datos decimales, y 'strComp' para comparar cadena de caracteres.

5. Luego se imprime de acuerdo al tipo de datos, para ello se ha utilizado el concepto de puntero genérico para poder tratar de ajustar el concepto de programación genérica escribiendo código en C.
   En este caso para poder ordenar los datos en un arreglo se ha casteado para poder ocupar el lugar en memoria y hacer la ordenación como corresponde para cada tipo, asi como para la lectura y escritura de los elementos ordenados para cada tipo.

C++ soporta muy bien programación para genéricos a través de su sistema de plantillas.[59]

[58] http://agora.pucp.edu.pe/inf2170681/

[59] http://www.generic-programming.org/languages/cpp/techniques.php

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

const int Entero = 1;
const int Cadena = 2;
const int Decimal = 3;

template<typename T>
void demoSort(int n) {
    T item;
    int i;
    vector<T> items;
    for (i = 0; i < n; i++) {
        cin >> item;
        items.push_back(item);
    }
    sort(items.begin(), items.begin() + n);
    cout << "Después de ordenar\n ";
    for (typename vector<T>::const_iterator it = items.begin();
            it != items.end(); ++it) {
        cout << " " << *it <<"\n";
    }
}

int main() {
    int op, n;

    cout << "Ingrese el número de valores: ";
    cin >> n;
    cout << "(1)Entero, (2)Cadena y (3)Decimal: ";
    cin >> op;
    switch (op) {
    case Entero:
        demoSort<int>(n);
        break;
    case Cadena:
        demoSort<std::string>(n);
        break;
    case Decimal:
        demoSort<double>(n);
        break;
    default:
        break;
    }
    return 0;
}
```

Figura 5.30.: Desarrollo, corrida y salida del programa que ordena elementos en C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

const int Entero = 1;
const int Cadena = 2;
const int Decimal = 3;

template<typename T>
void demoSort(int n) {
    T iten;
    int i;
    vector<T> items;
    for (i = 0; i < n; i++) {
        cin >> iten;
        items.push_back(item);
    }
    sort(items.begin(), items.begin() + n);
    cout << "Después de ordenar\n";
    for (typename vector<T>::const_iterator it = items.begin();
         it != items.end(); ++it) {
        cout << " " << *it <<"\n";
    }
}

int main() {
    int op, n;

    cout << "Ingrese el número de valores: ";
    cin >> n;
    cout << "(1)Entero, (2)Cadena y (3)Decimal: ";
```

```
julita@yulys: ~/Documents/tesis
File Edit View Search Terminal Help
julita@yulys:~/Documents/tesis$ gedit genericos.cpp &
[2] 15465
[1]   Done                    gedit genericos.hs
julita@yulys:~/Documents/tesis$ g++ -o genericos genericos.cpp
julita@yulys:~/Documents/tesis$ ./genericos
Ingrese el número de valores: 5
(1)Entero, (2)Cadena y (3)Decimal: 2
t e s i s
Después de ordenar
 e
 i
 s
 s
 t
julita@yulys:~/Documents/tesis$
```

En este caso, se está presentando los códigos tanto en C y en C++, y se puede apreciar que el código es más compacto en C++. Se ha evolucionado de manera que el programador pueda hacer uso de librerías predefinidas reutilizando código.

Lenguaje de Programación JAVA

```java
package genericos;
import java.util.List;
import java.util.ArrayList;
import java.util.Scanner;

public class Quicksort<T extends Comparable<T>> {
    public void sort(List<T> lista){
        sort(lista, 0, lista.size() - 1);
    }

    public void sort(List<T> lista, int start, int end){
        int i = start, k = end ;
         if ((end - start) >= 1){
            T pivot = lista.get(start);
            while (i < k){
                while (lista.get(i).compareTo(pivot) <= 0 && i < end && k > i){
                    i++;
                }
                while (lista.get(k).compareTo(pivot) > 0 && k >= start && k >= i){
                    k--;
                }
                if (i < k){
                    intercambiar(lista, i, k);
                }
            }
            intercambiar(lista, start, k);
            sort(lista, start, k - 1);
            sort(lista, k + 1, end);
        }
        else{
            return;
        }
```

```
}
private void intercambiar(List<T> lista, int i, int j){
    T temp = lista.get(i);
    lista.set(i, lista.get(j));
    lista.set(j, temp);
}
public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    int numVals;
    int tipoVals;

    System.out.println ("Ingrese el número de valores: ");
    numVals = sc.nextInt();
    System.out.println ("Ingrese el Tipo de dato a ordenar: ");
    System.out.println ("(1)Entero, (2)Cadena y (3)Decimal: ");
    tipoVals = sc.nextInt();

    List<Integer> listaEnteros = new ArrayList <Integer>();
    List<String> listaCadenas = new ArrayList <String>();
    List<Float> listaFloat = new ArrayList <Float>();

    if (numVals > 0){

        for (int i=1; i <= numVals; i++){
            switch (tipoVals) {
                case 1:  listaEnteros.add(new Integer(sc.nextInt()));
                         break;
                case 2:  listaCadenas.add(new String(sc.next()));
                         break;
                case 3:  listaFloat.add(new Float(sc.next()));
                         break;
                default:
                         break;
            }
        }
   }

    switch (tipoVals) {

         case 1: Quicksort<Integer> quicksortEntero = new Quicksort<Integer>();
                 quicksortEntero.sort(listaEnteros);
                 System.out.println("Después de ordenar");
                 for (Integer i : listaEnteros){
                    System.out.println(i.intValue());
                 }
                 break;
         case 2: Quicksort<String> quicksortCadena = new Quicksort<String>();
                 quicksortCadena.sort(listaCadenas);
                 System.out.println("Después de ordenar");
                 for (String i : listaCadenas){
                    System.out.println(i);
                 }
                break;
         case 3: Quicksort<Float> quicksortFloat= new Quicksort<Float>();
                 quicksortFloat.sort(listaFloat);
                 System.out.println("Después de ordenar");
                 for (Float i : listaFloat){
                    System.out.println(i);
```

```
                    }
                    break;

            default:

                    break;  }
    }

}
```

Figura 5.31.: Corrida y salida del programa que ordena elementos en Java

1. Se agrega la sentencia de importación de la librería ***java.util.List***, la cual nos permitirá usar exclusivamente a la clase list la cual provee funcionalidades para colecciones de datos. Se utiliza List en lugar de la librería Vector porque no es muy óptimo en el uso de memoria cuando se duplica si se llega al límite; la librería ***java.util.ArrayList***, la cual nos permitirá usar exclusivamente la clase vector, la cual provee funcionalidades para colecciones de datos; la librería ***java.util.Scanner***, la cual nos permitirá hacer uso de la clase scanner, la cual provee funcionalidades para el ingreso de datos por consola, conversión a tipos primitivos y fragmentación de cadenas por tokens.
2. Se crea la cabecera de la clase pública 'quicksort', la cual hace uso de la interface comparable y se define el método público 'sort' el cual recibe una colección del tipo lista. Se hace llamado al método 'sort', el cual, a diferencia del método que lo utiliza en el cuerpo del programa, recibe 3 parámetros, una lista y dos posiciones, con esto, podemos identificar una sobrecarga de métodos. Se define el método sobrecargado 'sort' que recibe los tres parámetros anteriormente indicados.

3. Se inicializa la variable entera ***i*** con el valor 'start' pasado como parámetro, y la variable entera k con el valor 'end'.
4. Se realiza una pregunta lógica para bifurcar el comportamiento dependiendo de la condición lógica expuesta en esta línea de código (deberías ponerle líneas de código…números), si la diferencia entre 'end' y 'start' es mayor o igual a 1, lo que se interpreta como que los limites aún no se han cruzado o encontrado (cuestión importante en el algoritmo quicksort), es la condición lógica que determinará el comportamiento del programa. De ser verdadero, la variable 'pivot' se inicializada con el valor d la lista en la posición 'start'.
5. Sentencia 'while' es utilizadad mientras se cumpla una condición es usada en esta línea de código, garantiza la repetición de un comportamiento hasta q se rompa la condición establecida, en este caso, que i sea menor que 'k', y se compara el valor en la posición i de la lista con el pivot y que i sea aún menor que end y a la vez menor que 'k' obliga a un incremento en 1 de la variable i; asicomo el valor en la posición 'k' de la lista con el pivot y que k sea mayor que start y a la vez mayor o igual que i obliga a un decremento en 1 de la variable ***k***. En este caso si i es menor que ***k***. Llamada al método intercambiar, el cual recibe como parámetros la lista, y las variables ***k*** e ***i***. Todo lo descrito se repite mientras se cumpla la condición indicada.
6. Se llama al método intercambiar, el cual recibe como parámetros la lista, 'start' y 'k' y al método 'sort' sobrecargado con 3 parámetros, en 2 oportunidades, de naturaleza recursiva, con la finalidad de ir ordenando tanto la "mitad" derecha como izquierda de la lista, recibiendo diferentes posiciones de inicio y fin. Se llama a 'else', en caso no se cumpla la condición de la pregunta del cruce de posiciones. Si hay acción de retorno, indica que ya se ordenó un sector de la lista.
7. Se define el método privado 'intercambiar', el cual solo puede ser accedido dentro de la clase y recibe como parámetros una lista y 2 números enteros.
8. Se inicializa la variable temporal 'temp', la cual recibe el dato que se encuentra en la lista en la posición i, pasada como parámetro; y se le asigna a la lista en la posición i el valor de la lista en la posición j pasada como parámetro, se realiza el intercambio.
9. Se le asigna a la lista en la posición j, el valor salvado en el temporal.
10. Se define el método statico main. Se prepara el ingreso de datos por la entrada estándar, haciendo uso de la clase scanner, y se crean dos variables enteras; y se manda a la salida estándar el mensaje "ingrese el número de valores: "

11. Se captura de la entrada estándar y asigna la captura a la variable numvals y se envían dos nuevos mensajes a la salida estándar, luego se captura entrada estándar y asigna valor a variable tipovals. Se crean variables tipo lista de distintos tipos de dato.
12. Se define una sentencia selectiva, la cual de cumplirse realiza el llenado de las listas, dependiendo del tipo de valores que se indicó por la entrada estándar, y dependiendo del tipo de valor que se indicó, se crea una instancia de quicksort el cual realizará el llamado al método ***sort***, recibiendo como parámetro la lista llenada en el caso anterior, y se muestra por la salida estándar la lista ordenada luego de la ejecución del método ***sort***. Se realiza esta acción de pendiendo de que instrucción case se deba ejecutar, la sentencia break interrumpe este bucle y termina el programa.

Lenguaje de Programación HASKELL

Lo que Java llama genéricos, Haskell lo obtiene a partir de variables de tipo, casi todas las funciones en Haskell incluyen el concepto «genérico» en el sentido de Java.

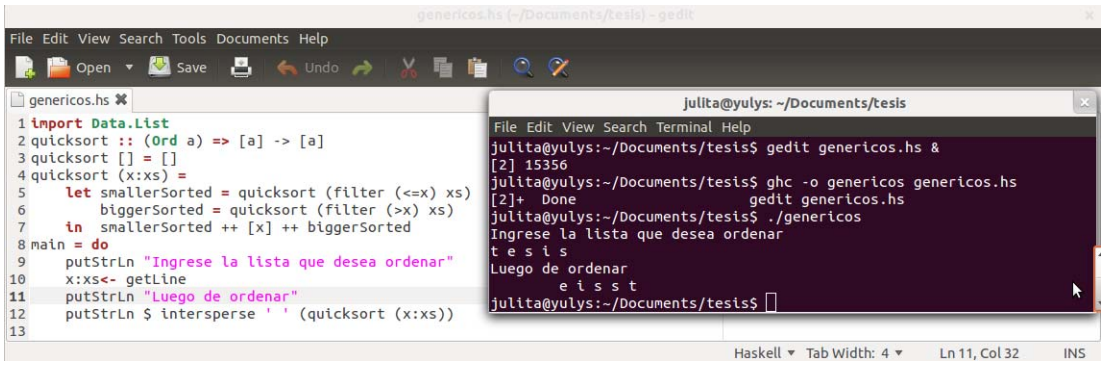
Cuando Haskell refiere "programación genérica", refiere a algo mucho más poderoso, como la búsqueda de datos a través de una gran estructura de datos compuesto.
Por ejemplo encontrar todos los salarios y aumentarlos en un 20%.
Esto se podría hacer con la reflexión de introspección profunda de todos los campos en busca de cosas de tipo 'salarial' y cambiar sus valores, para ello se pueden utilizar las librerías Data.Data y GHC.Generics.
Los genéricos en Java se detienen con argumentos de tipo único, pero en Haskell se puede hacer cosas como "genéricos en genéricos". Por ejemplo,

```
toList :: Foldable f => f a -> [a]
```

Es polimórfico en el tipo de "contenedor de 'a' no sólo en el tipo del contenedor tiene.

```
import Data.List
quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
    let smallerSorted = quicksort (filter (<=x) xs)
        biggerSorted = quicksort (filter (>x) xs)
    in  smallerSorted ++ [x] ++ biggerSorted
main = do
    putStrLn "Ingrese la lista que desea ordenar"
    x:xs<- getLine
    putStrLn "Luego de ordenar"
    putStrLn $ intersperse ' ' (quicksort (x:xs))
```

Figura 5.32.: Desarrollo, corrida y salida del programa que ordena elementos en Haskell



Lenguaje de Programación GO

```
package main

import "fmt"

func main() {
   list := []int{31, 41, 59, 26, 53, 58, 97, 93, 23, 84}
   fmt.Println("unsorted:", list)

   quicksort(list)
   fmt.Println("sorted!  ", list)
}

func quicksort(a []int) {
   var pex func(int, int)
   pex = func(lower, upper int) {
      for {
         switch upper - lower {
         case -1, 0:
            return
         case 1:
            if a[upper] < a[lower] {
               a[upper], a[lower] = a[lower], a[upper]
            }
            return
         bx := (upper + lower) / 2
         b := a[bx]
         lp := lower
         up := upper
      outer:
```

```
        for {
            for lp < upper && !(b < a[lp]) {
                lp++
            }
            for {
                if lp > up {
                    break outer
                }
                if a[up] < b {
                    break
                }
                up--
            }
            a[lp], a[up] = a[up], a[lp]
            lp++
            up--
        }
        if bx < lp {
            if bx < lp-1 {
                a[bx], a[lp-1] = a[lp-1], b
            }
            up = lp - 2
        } else {
            if bx > lp {
                a[bx], a[lp] = a[lp], b
            }
            up = lp - 1
            lp++
        }
        if up-lower < upper-lp {
            pex(lower, up)
            lower = lp
        } else {
            pex(lp, upper)
            upper = up
        }
    }
  }
  pex(0, len(a)-1)
}
```

Evaluación de soporte para genéricos

C no soporta programación genérica, para ser genérica no debe ser dependiente del tipo de dato, e incluso debería poder soportar estrucutras ‘structs’, y no lo hace. En este caso se ha tratado de codificar utilizando estructuras ‘case’, punteros y asignaciones de memoria para poder tratar los tipos de datos ‘int’, ‘float’ y ‘char’ de manera análoga como se hace cuando se utiliza programación genérica, que soporta cualquier tipo de dato.

Dada la estructura del programa presentado en C, el manejo de punteros es un tema en el cual se especializan programadores de nivel avanzado, caso contrario hay que tener cuidado para evitar punteros colgando, los que cancelan la asignación de un montículo de variables o que den muerte a las variables locales. Cuando se manejan muchos punteros, es difícil la depuración y no hay ninguna comprobación en tiempo de ejecución. En consecuencia se generan errores de programación (bugs) difícil de localizar y corregir a tiempo.

En el caso de asigancion de memoria, en C es muy fácil acceder a posiciones ilícitas de memoria (índices negativos o fuera de rango). Cada vez que C hace uso de memoria dinámica se debe tener en cuenta su liberación (malloc – free). La falta de controles en tiempo de ejecución permite que estos errores, y otros como indización de matrices fuera de rango, pasen desapercibidos hasta que ilimitados daños se han producido. Java es mucho más fiable en este caso, no pueden surgir punteros colgando, la comprobación del tipo en tiempo de compilación de forma automática y detecta una gran proporción de errores de programación garantizando un control automático de tiempo de ejecución de errores, como por ejemplo, la indización de matrices fuera de rango.

Haskell ha trabajado en los últimos años para incluir el concepto de genéricos en casi todas sus funciones. Tanto Java y Haskell, consideran esta característica como importante para mantenimiento de ambos proyectos como lenguajes de programación.

En Haskell, el concepto de genéricos es simplemente "polimorfismo paramétrico".

En la entrevista a un colaborador de la comunidad Haskell, Edward Kmett, indicó que uno de los creadores de la funcionalidad de polimorfismo en Haskell pasó a trabajar en los genéricos de Java.

Agregó que Haskell supera carencia de flexibilidad causados por la secuencialidad de las instrucciones en lenguajes imperativos como C y de su heredero Java.

CONCLUSIONES

1. Con respecto a los criterios de evaluación de los lenguajes de programación elegidos:

   El reto de los creadores y diseñadores de un lenguaje de programación es lograr un lenguaje de programación que permita que la máquina ejecute los requerimientos predefinidos para el desarrollo de un software o sistema de manera rápida y confiable. Los criterios como fácil lectura de código, fácil escritura de código y soporte para genéricos; permiten el desarrollo de software de manera rápida. Los criterios como confiabilidad y reflexión permiten, valga la redundancia un software confiable.

2. Con respecto a los paradigmas y lenguajes de programación evaluados:

   La teoría de los lenguajes de programación no se desliga a los paradigmas bajo los cuales se sustentan. Los diferentes paradigmas de programación permiten al programador resolver situaciones y brindar soluciones a los requerimientos de software de diferente manera. Los lenguajes C/C++, Java y GO han sido elegidos como representantes actuales, de alto nivel, pertenecientes a los paradigmas imperativo, orientado a objetos y multiparadigma, para ser comparados con el lenguaje Haskell, de paradigma funcional. Se ha demostrado en la tesis, el liderazgo y repercusión actual de estos lenguajes en los campos académico, industrial y científico.

3. Con respecto a la popularidad de los lenguajes evaluados:

   Una de las razones de la baja popularidad de Haskell en la actualidad es que no cuenta con respaldo de empresas, proyectos o plataformas de aplicaciones de renombre como Oracle o Android, como se ha dado en el caso de Java; o la compañía Google Inc, en el caso de GO. Haskell es un lenguaje de programación que fue creado en contemporáneo con Java; sin embargo la participación para aplicaciones Web con frameworks desarrollados con Java aparece desde mediados de 1999, con Servlet en Java 1.2. Frameworks desarrollados con Haskell como Happstack, Snap y Yesod recién son conocidos a partir del año 2011. De acuerdo a lo expuesto, se deduce que el lenguaje de programación Haskell tendrá un incremento de popularidad en los

futuros años, pero no llegará a tener gran trascendencia a menos que se involucre o asocie a un proyecto reconocido a nivel mundial.

4. Con respecto a la plataforma con la que se evaluaron los lenguajes de programación:

   Se debe utilizar un lenguaje de programación de acuerdo a la problemática y los entornos de desarrollo disponibles. Se ha demostrado en la tesis, que existen compiladores, entornos de desarrollo, IDEs y demás tecnologías propicias para programar en C, C++, Java, Haskell y GO en Linux. Los códigos empleados fueron compilados y corridos bajo las distribuciones Ubuntu 11.10 y Fedora 16. Se emplearon herramientas libres como Mysql, Mysql Workbench y NeatBeans IDE 7.0.1.

5. Ventajas y Desventajas de Haskell con respecto a C, C++, Java y GO:

   - ✓ Con respecto a la fácil lectura de código, se realizaron 62 encuestas a los alumnos de primer ciclo PUCP que tuvieron experiencia con el lenguaje de programacion Visual Basic for Applications, se consultó acerca del resultado del cálculo del BMI. Haskell ocupó un segundo lugar, luego del lenguaje de programacion GO. El resultado estuvo sujeto al tiempo de respuesta, acierto y evaluación del lenguaje de programacion que le tocó resolver a cada alumno.
     En un proyecto real, mientras más tiempo se demore un programador en leer el código, más tiempo se invierte en el mantenimiento del mismo. El tiempo invertido se traduce en costos monetarios y se refleja en cifras de satisfacción del cliente por el cumplimiento de entrega del software o por realización de cambios.

   - ✓ Con respecto a la fácil escritura, se escribieron códigos prácticos para utilizar funciones matemáticas y de manejo de cadena de caracteres, asi como el tratamiento de archivos de entrada/salida y la conexión a una base de datos en Mysql. Al desarrollar los cuatro códigos escritos para analizar este criterio, Haskell ocupó el primer lugar. Se obtuvo menos líneas de código con Haskell que con los demás lenguajes de programación.
     En un proyecto real, Haskell ofrece un código simple, conciso y ordenado (el compilador obliga a indentar código), sin embargo esta simplicidad tiene como contraparte la lectura de código. Al tener un código muy simple, resulta que se tiene una simbología casi críptica y afecta el factor tiempo en la lectura de código. Se debe tener un buen manejo y mucha práctica del lenguaje de programación

Haskell, para evitar sobreescribir sin causar bugs en otra parte del programa o en otros módulos. Esto requiere de muchas horas de entrenamiento y concentración.

- ✓ Con respecto a la confiabilidad, Haskell tiene un sistema de tipo fuertemente tipificado, y es así que cada expresión tiene un tipo que, puede ser definido por el programador o inferido por el compilador. GO también es un lenguaje de programación confiable que requiere la evaluación de cada entrada de una variable con respecto a su valor consistente.
  Un software debe ser exacto en los cálculos y tareas asignadas, no debe causar riesgos o equivocaciones en pagos u operaciones que afectan a millones de usuarios, o casos como operaciones quirúrgicas. Java aún presenta errores de programación cuando se tiene interacción entrada/ salida. Estos errores han sido superados por Haskell utilizando estructuras modulares denominadas mónadas.

- ✓ Con respecto a la reflexión, es un conjunto de funcionalidades que simplifica tareas como la ingeniería directa (forward engineering), depuración, realizar cambios de datos en tiempo de ejecución cuando no se especifíca o conoce una clase en especial o su método a través de la extensibilidad de clases. Es una propiedad avanzada que tienen Java y GO, C++ la tiene a través de librerías que crearon algunos programadores; pero ni C ni Haskell emplean. En Haskell no se tiene clases que examinar, ni métodos ni propiedades con las que se puede realizar una instrospección. Los tipos e inferencias de tipos ya vienen incluidos por defecto.

- ✓ Con respecto a la programación para genéricos, Haskell ocupa un primer lugar, seguido por C++, luego por Java y finalmente GO. En el presente trabajo se presentó un código que ordena valores sin importar el tipo numérico o cadena de caracteres, en el cual Haskell permitió el uso de una función que simplifica la escritura de funciones a partir de algoritmos sin importar el tipo de dato. Haskell tiene un tipo general denominado “a” el cual está incluído en funciones predeterminadas, facilitando la escritura de código, con menos tiempo y menos líneas de código.

RECOMENDACIONES

1. Con respecto a los criterios de evaluación de los lenguajes de programación:

   Existen muchos criterios por los cuales los lenguajes de programación pueden ser evaluados. Recomiendo que en un futuro estudio, los criterios a tomarse deben estar relacionados directamente al actual paradigma tecnológico, el nuevo modelo de Web 4.0, que incluye una capa de integración para explotar al máximo los beneficios de la Web semántica, y nuevo esquema de interacción con millones de usuarios a nivel mundial.

2. Con respecto a los paradigmas y lenguajes de programación evaluados:

   Existen muchos paradigmas de programación con los cuales muchos lenguajes de programación pertenecientes a ellos pueden ser evaluados. Recomiendo que un futuro estudio, los lenguajes evaluados deben pertenecer al mismo paradigma de programación. La tendencia es el uso de lenguajes de programación multiparadigma, el estudio debe abarcar liderazgo y repercusión actual de estos lenguajes en los ámbitos académico, industrial y científico.

3. Con respecto a la popularidad de los lenguajes evaluados:

   No existen fuentes confiables que nos puedan ofrecer estadísticas de uso de software con determinado lenguaje de programación en Perú. Recomiendo que en un futuro estudio, se proponga un sitio Web que pueda registrar tanto en el ámbito académico como industrial y científico, el uso de los diferentes lenguajes de programación y los propósitos para los cuales son utilizados.

4. Con respecto a la plataforma con la que se evaluaron los lenguajes de programación:

Existen muchos paquetes disponibles para Linux para las diferentes distribuciones como Ubuntu, Fedora y Open Suse, entre otras. Recomiendo que en un futuro estudio, se realicen pruebas de las diversas tecnologías y herramientas libres para contrastar desempeño y estabilidad con otros sistemas operativos.

5. Ventajas y Desventajas de Haskell con respecto a C, C++, Java y GO:

- Con respecto a la fácil escritura de código y lectura de código, se tiene precedente de estudios de pruebas de códigos a estudiantes que ingresan a la universidad para conocer la manera que conceptualizan la solución de problemas o algoritmos, y determinar el paradigma de programación que se debe impartir por primera vez en la clase hoy en día. Las pruebas consistieron en pruebas de asignaciones de valores y estructuras de control. En este trabajo se mezclaron dichas estructuras para realizar la prueba de comprensión de código. En un futuro estudio, recomiendo replantear y redefinir las estructuras de código que deben evaluarse y realizar la prueba de comprensión de lectura de código con alumnos de diferentes instituciones y universidades de Lima y provincias.

- Con respecto a la confiabilidad de código, se recomienda realizar un sistema de gestión de conocimiento del uso práctico de los lenguajes Java, Haskell, GO, C/C++ o en los lenguajes que se consideren en futuros estudios en el Perú; y comprobar con frameworks o códigos puros la estabilidad frente a tecnologías que auditan la seguridad informática.

BIBLIOGRAFIA

[ACM 08] Computer Science Curriculum 2008: An Interim Revision of CS 2001-Report from the Interim Review Task Force
Association for Computing Machinery - IEEE Computer Society. December 2008.

[Alvarez, Pow Sang 08] Alvarez, Marco A., Pow Sang, José A., Baiocchi, José.
Computing and higher education in Peru
SIGCSE Bulletin , Volume 40 Issue 2. Publisher: ACM. June 2008.

[Bernardy 10] Bernardy, Jean-Philippe; Jansson, Patrik; Zalewski, Marcin and Schupp, Sibylle. Generic programming with C++ concepts and Haskell type classes -a comparison. ACM SIGPLAN Workshop on Generic Programming WGP.

[Blake 11] Blake, Jonathan D.
Language considerations in the first year CS Curriculum
Journal of Computing Sciences in Colleges. Volume 26 Issue 6, June 2011 Consortium for Computing Sciences in Colleges , USA.

[Bloch 02] Bloch, Joshua.
Effective Java Programmming Language Guide, Addison- Wesley 2002. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[Budd 89] Budd, Timothy A., Pandey, Rajee K.
Never mind the Paradigm, what about Multiparadigm Languages?
SIGCSE BULLETIN. Vol. 27 No.2. June 1995.

[Cano 10] Cano Utrera, Andrés
Nuevas Tecnologías de la Programación. Módulo 3: Programación en Java en entorno UNIX. Dpto. Ciencias de la Computación e I.A. Universidad de Granada. 2010

[Cantwell 82] Cantwell Smith, Brian

Reflection and Semantics in a Procedural Language. PhD. Thesis. Massachusetts Institute of Technology. MIT-LCS-TR-272. Jan 1982.

[Cantwell 84] Cantwell Smith, Brian
Reflection and semantics in LISP. POPL '84 Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages ACM NY.

[Chung-chieh 05] Chung-chieh, Shan.
A Computational Interpretation of Classical S4 Modal Logic
Harvard University. (Draft of April 21, 2005)

[CoreGrid 07] CoreGrid NoE – Institute on Programming Model, Basic Features of the Grid Component Model, Deliverable Report D.PM.04, 2007.

[CSTA 11] The Voice of K–12 Computer Science Education and it s Educators
Volume 7, Issue 4. September 2011.

[De Raadt 02] De Raadt, Michael, Watson, Richard and Toleman, Mark.
Language Trends in Introductory Programming Courses.
University of Southern Queensland, Australia.
Informing Science InSITE - "Where Parallels Intersect" June 2002.

[Dehnadi 06] Dehnadi Saeed
Testing Programming Aptitude. School of Computing Middlesex University, UK. 2006.

[Floyd 07] Floyd, Robert W.
The paradigms of programming.ACM Turing award lectures, 2007 - portal.acm.org

[Garcia 03] Garcia, Ronald, Jarvi, Jaakko, Lumsdaine, Andrew, Siek, Jeremy G. y Willcoc, Jeremiah.
A Comparative Study of Language Support for Generic Programming
OOPSLA '03 Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. USA ©2003.

[Garcia 07] Garcia, Ronald, Jarvi, Jaakko, Lumsdaine, Andrew, Siek, Jeremy G. y Willcoc, Jeremiah.

An extended Comparative Study of Language Support for Generic Programming. Volume 17 Issue 2, March 2007. Cambridge University Press New York, NY, USA

[Getov 10] Getov, Vladimir.
Software Development Productivity: Challenges and Future Trends.
2010 IEEE 34th Annual Computer Software and Applications Conference (2010)

[Goldfinger 61] Goldfinger, Roy.
Problem-oriented programming language structure
Communications of the ACM , Volume 4 Issue 3. March 1961.

[González 04] González Benito.
Introducción al API Reflection (Reflexión) de Java. 27 Julio del 2004.

[Grogono 04] Grogono, Peter.
Principles of Programming Languages
Department of Computer Science and Software Engineering Concordia University Montreal, Quebec 2004

[Hartel 95] Hartel, Pieter H., Hertzberger, L.O.
Paradigms and laboratories in the core computer science curriculum: An overview
SIGCSE Bulletin Vol 27 No. 4 December 1995

[Henglein 10] Henglein, Fritz.
Technical Perspective Large-Scale Sound and Precise Program Analysis
Communications of the ACM Volume 53 Issue 8, August 2010. New York, NY, USA.

[Herzeel 08] Herzeel, Charlotte; Costanza, Pascal and D'Hondt, Theo.
Reflection for the Masses. Vrije Universiteit Brussel. Berlin Heidelberg 2008.
In: R. Hirschfeld and K. Rose (Eds.): S3 2008, LNCS 5146, pp. 87-122, 2008.

[HU 11] HU, Zhenjiang.
Special Issue on Programming Languages and Systems
NEW GENERATION COMPUTING Volume 29, Number 1, 1-2 (2011)

[Hutton 07] Hutton, Graham.

Programming in Haskell, Cambridge University Press, 2007. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[Jazayeri 98] Mehdi Jazayeri,Rüdiger Loos,David R. Musser.
International Seminar on Generic Programming Dagstuhl Castle, Germany April/ May 1998.

[Lipovača 11] Lipovača Miran.
Learn You a Haskell for Great Good!. April 2011, 400 pp.

[Löh 08] Löh, Andres; Jeuring, Johan; Van Noort, Thomas; Rodriguez, Alexey.
The Generic HASKELL User's Guide. Version 1.80 (Emerald). April 11, 2008. Institute of Information and Computing Sciences. Utrecht University. Netherlands.

[Luker 89] Luker, P. A.
Never mind the language, what about the paradigm?
SIGCSE '89: Proceedings of the twentieth SIGCSE technical symposium on Computer science education. February 1989.

[Maes 87] Maes, Pattie.
Concepts and experiments in computational reflection
ACM SIGPLAN Notices Volume 22 Issue 12, Dec. 1987. ACM New York, USA.

[Martin 08] Martin Sierra, Antonio J.
Programador Certificado Java 2: Curso Práctico. Segunda Edición, Octubre 2007.

[Mitchell 03] Mitchell, John C.
Concepts in Programming Languages. Cambridge University Press 2003. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[O'Sullivan 98] O'Sullivan, Bryan, Stewart, Don and Goerzen, John.
Real World Haskell. O'Really 2008. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[Pang 03] Pang, André T.H.
Binding Haskell to Object-Oriented Component Systems via Reflection.

Master's thesis, The University of New South Wales, School of Computer Science and Engineering, June 2003.
[Pike 10] Rob Pike. Another Go at Language Design. 28 Apr 2010. Google, Inc.

[Pou 94] Dick Pountaln
Functional programming comes of age, 1994

[Prashant 08] Prashant Kulkarni, Vaibhav Shankar, and Shashi Nagarajan, "Programming Languages: A Comparative Study", Mayo 2008. Technical Report.

[Pratt 98] Pratt, Terrence W., Zelkowitz, Marvin V.
Lenguajes de Programación, diseño e Implementación.
Tercera Edición. Prentice-Hall Hispanoamericana, S.A. 1998.

[Premchand 12] Premchand, Kumar Vasantha
Code Legibility by A Metric Method. Dept. of IT, Avanthi Institute of Engineering & Technology, Visakhapatman, AP, India. IJCST Vol. 3, Issue 1, Jan - March 2012

[Rao 91] Rao, Ramana
Implementational Reflection in Silica. ECOOP '91 Proceedings of the European Conference on Object-Oriented Programming. Springer-Verlag London, UK ©1991
[Schildt 11] Schildt, Herbert
Java The Complete Reference, 8th Edition

[Schildt 00] Schildt, Herbert
C – The complete Reference, 4th edition. McGraw Hill 2000.

[Scott 08] Scott, Michael L.
Programming Language Pragmatics, 3th ed. Morgan Kaufmann Publishers 2008. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[Sebesta 10] Sebesta, Robert.
Concepts of Programming Languages, 9th ed. Addison- Wesley 2010. Ubicación: Biblioteca de Ingeniería de la Pontificia Universidad Católica del Perú.

[Sestoft 08] Sestoft, Peter.

Programming Language Concepts for Software Developers
IT University of Copenhagen, Denmark 2008
[Van 04] Peter Van Roy, Seif Haridi
Concepts, Techniques, and Models of Computer Programming. MIT Press 2004

[Venkatreddy 10] Venkatreddy, Dwarampudi, Singh Dhilon, Shahbaz; Shah, Jivitesh, Sebastian, Nikhil Joseph; Satyanaran Kanigicharla, Nitin.
Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, PHP & Scheme - a Team 11 COMP6411-S10 Term Report. Revisión 1.0. Cornell University Library. Agosto 2010.

[Vujošević-Janičić 08] Vujošević-Janičić, Milena, Tošić, Dušan.
The role of Programming Paradigms in the First Programming Courses.
Teaching of Mathematics. 2008, Vol. XI, 2, pp. 63-83

[Washburn 07] Washburn, Geoffrey Alan
Principia Narcissus: How to Avoid Being Caught by Your Reflection.
University of Pennsylvania. November 2007

[Watt 04] Watt, David A.
Programming Language Design Concepts, John Wile & Sons, Ldt 2004. Ubicación: Biblioteca de Ingeniería - Pontificia Universidad Católica del Perú.

[Wu 98] Wu, Zhixue,
Reflective Java and A Reflective-Component-Based Transaction Architecture, in Proc. of OOPSLA'98 Workshop on Reflective Programming in C++ and Java (J.-C. Fabre and S. Chiba,eds.), 1998.

[Živanović 10] Živanović, Darko, Stefanović, Aleksandar.
A Survey of Programming Language Popularity: Is Java Dead?
Department of Computer Science. Faculty of Mathematics. University of Belgrade

ANEXOS

Anexo 1

A) Lenguajes de programación empleados en cursos en algunas universidades de Lima

Pontificia Universidad Católica del Perú

Ciclo	Curso	Lenguaje de Programación	Paradigma
Primer ciclo	Introducción a la Computación	Visual Basic for Applications	Multiparadigma
Cuarto ciclo	Técnicas de Programación	Pascal	Imperativo
Quinto ciclo	Fundamentos de Programación	Prolog, Haskell, Smalltalk	Declarativo, Funcional y OO
Sexto ciclo	Lenguajes de Programación 1	C/ C++	Imperativo/ OO
Séptimo ciclo	Lenguajes de Programación 2	C++,Java	OO

Fuente: Encuesta a profesores de Fundamentos de Programación y de Lenguajes de Programación: http://facultad.pucp.edu.pe/ingenieria/index.php?option=com_detalle&task=view&secc=14&cat=57&cont=141&btn_back=1&Itemid=88

Universidad Nacional Mayor de San Marcos

Ciclo	Curso	Lenguaje de Programación	Paradigma
Primer ciclo	Algorítmica 1	C/C++	Imperativo/OO
Segundo ciclo	Algorítmica 2 Estructura de datos	Java Java, C++	OO OO, Imperativo.
Tercer ciclo	Algorítmica 3	Java	OO
Cuarto ciclo	Computación Gráfica	Visual .NET Java	Multiparadigma Basado procedural
Sexto ciclo	Taller de Proyectos 1 Sistemas Operativos	Java C	OO Imperativo
Séptimo ciclo	Taller de Proyectos 2	Java/Scala	OO/Multiparadigma

Fuente: Encuesta a alumnos de Ingeniería de la Facultad de Ingeniería de Sistemas e informática – Ing. Sistemas

http://sistemas.edu.pe/ingenieria-sistemas/syllabus.html

Universidad Nacional de Ingeniería

Ciclo	Curso	Lenguaje de Programación	Paradigma
Tercer ciclo	Lenguajes algortimicos	C/Java	Imperativo/OO
Cuarto ciclo	Lenguaje estructurado	C	Imperativo
Quinto ciclo	Leng. Programación OO	Java	Orientado a Objetos
Séptimo ciclo	Inteligencia Artificial	Prolog/LISP	Declarativo/Funcional

Fuente: Encuesta a alumnos de Ingeniería de la Facultad de Ingeniería de Sistemas e informática

Universidad Nacional del Callao

Ciclo	Curso	Lenguaje de Programación	Paradigma
Segundo ciclo	Algoritmo y Estructura de Datos	Pascal	Imperativo
Tercer ciclo	Lenguajes de programación 1	C/ Java	Imperativo /OO
Cuarto ciclo	Lenguajes de programación 2	Java	OO
Séptimo ciclo	Lenguajes de programación 3	Power Builder	Multiparadigma
Noveno ciclo	Inteligencia Artifical	Prolog	Lógico

Fuente: Syllabus y encuesta a alumnos de Ingeniería de la Facultad de Ingeniería de Sistemas e informática

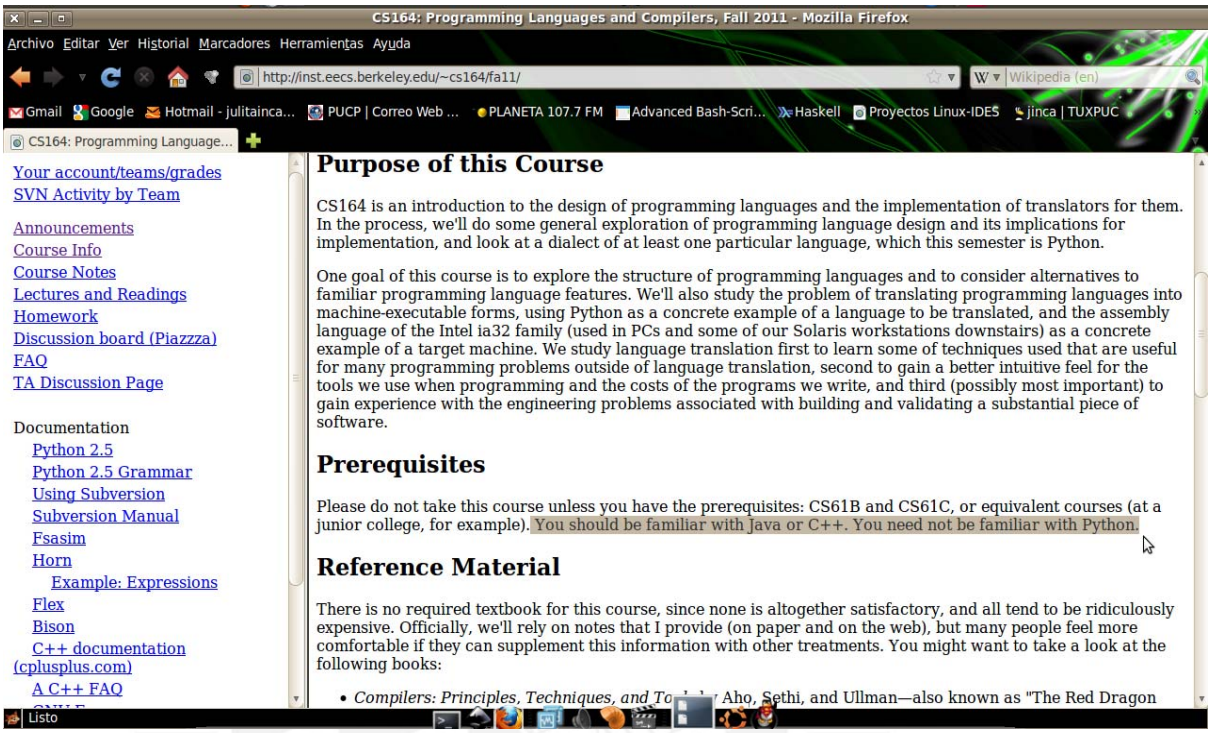
Universidad Inca Garcilazo de la Vega

Ciclo	Curso	Lenguaje de Programación	Paradigma
Primer ciclo	Fundamentos de Informática	PHP, Visual Basic	Multiparadigma
Tercer ciclo	Lenguaje de programación 1	C/ C++	Estructurado/OO
Cuarto ciclo	Lenguaje de programación 2	Java	OO
Quinto ciclo	Lenguaje de programación 3	PHP	Multiparadigma

Sexto ciclo	Tópicos programación	.NET	Multiparadigma

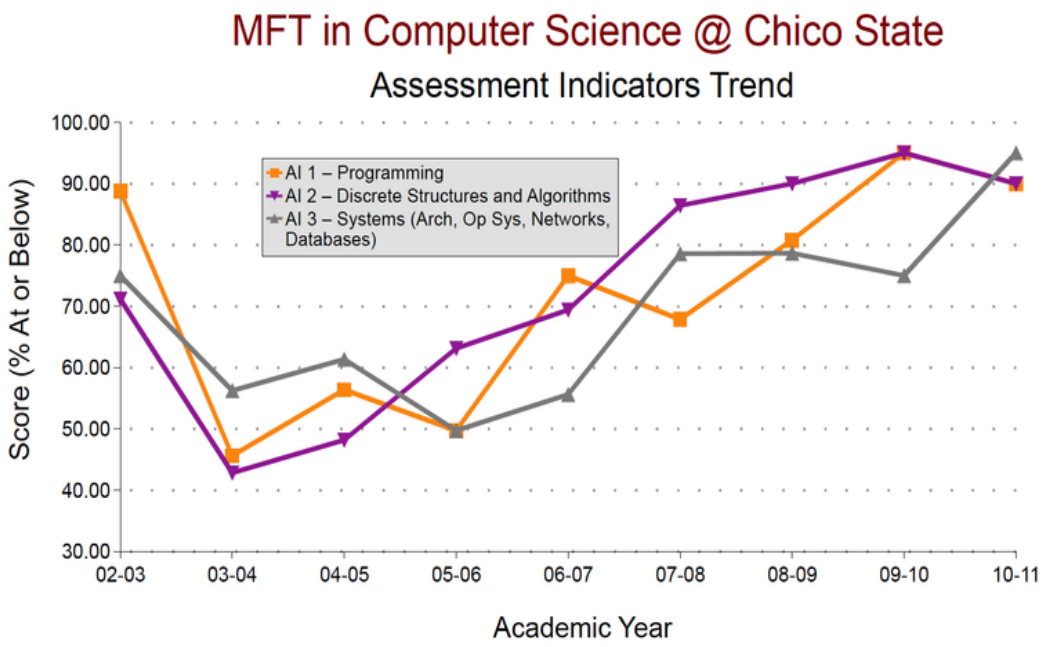
Fuente: Encuesta a alumnos de Ingeniería de la Facultad de Ingeniería de Sistemas e informática

B) Sitio Web de la documentación del curso Lenguajes de Programación en la Universidad de Berkeley



Fuente: http://inst.eecs.berkeley.edu/~cs164/fa11/

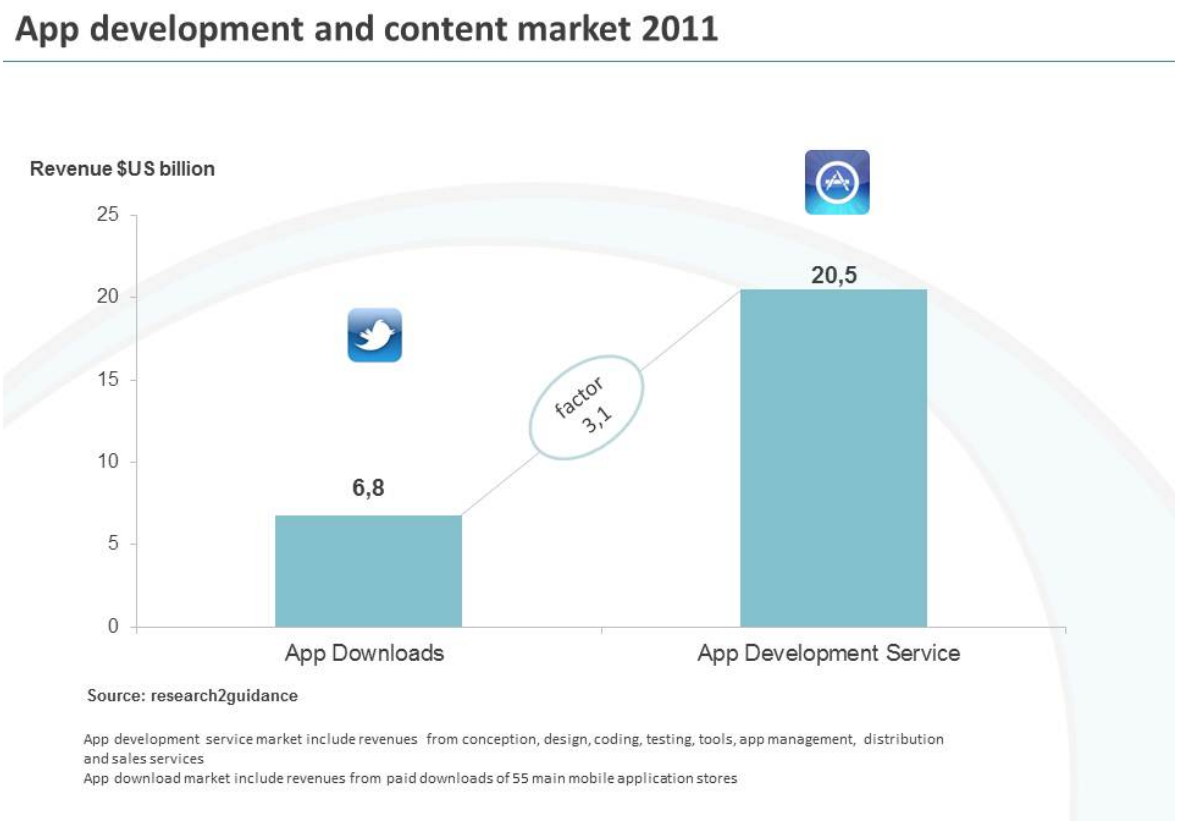
C) Calificaciones obtenidas en programación desde el 2002 al 2011 en EEUU.



Fuente: http://csci.ecst.csuchico.edu

Anexo 2

El Mercado para el servicio de desarrollo de aplicaciones móviles ascendió a $US 20.5 billion en 2011



Fuente: http://www.research2guidance.com/the-market-for-mobile-app-development-services-reached-us-20.5-billion-in-2011/

Oracle sues Google over Android and Java

Google's Android software infringes on several patents related to Java, which Oracle acquired from Sun earlier this year, Oracle says in suit filed Thursday.

by Tom Krazit | August 12, 2010 5:18 PM PDT

Two Silicon Valley heavyweights are about to reenact the Java wars: this time, in a court room.

Oracle issued a press release late Thursday saying it has filed suit against Google for infringing on copyrights and patents related to Java, which Oracle acquired along with Sun Microsystems earlier this year. The terse release claimed Google "knowingly, directly and repeatedly infringed Oracle's Java-related intellectual property."

A copy of the complaint (PDF), which was filed in the U.S. District Court for the Northern District of California, says that "Android (including without limitation the Dalvik VM and the Android software development kit) and devices that operate Android infringe one or more claims of each of United States Patents Nos. 6,125,447; 6,192,476; 5,966,702; 7,426,720; RE38,104; 6,910,205; and 6,061,520."

A Google representative said the company had not yet been served with the lawsuit, and therefore couldn't comment until it had a chance to review it. An Oracle representative declined to comment beyond the complaint.

FEATURED POSTS

Stephen Hawking: I lost a $100 bet over Higgs boson discovery
Technically Incorrect

Exoskeleton 3D-printed shoes look alien, awesome
Crave

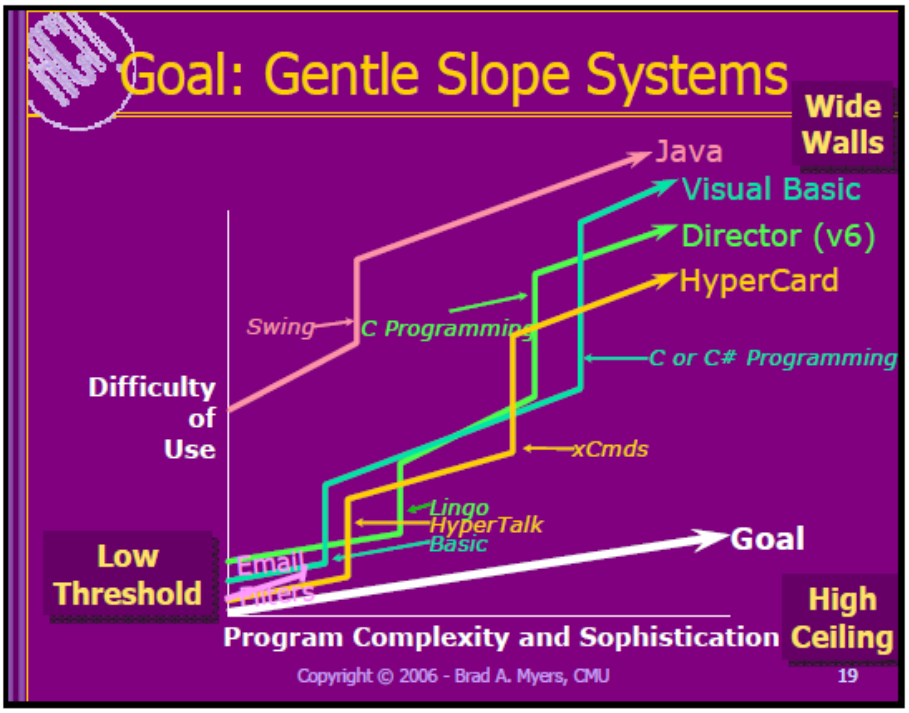
Did iPhone 'explode' in 17-year-old's pocket?
Apple

The three phases of Steve Ballmer's tenure at Microsoft
Microsoft

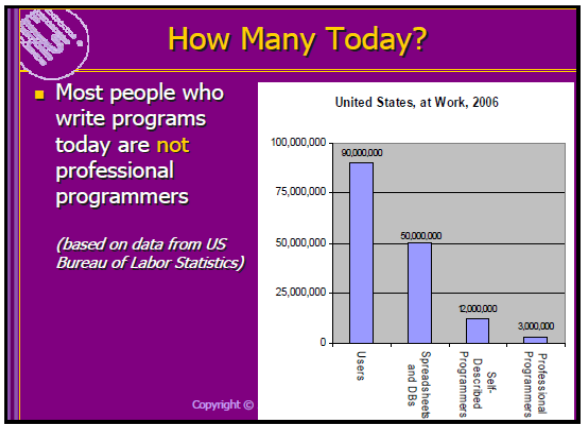
Fuente: www.news.cnet.com

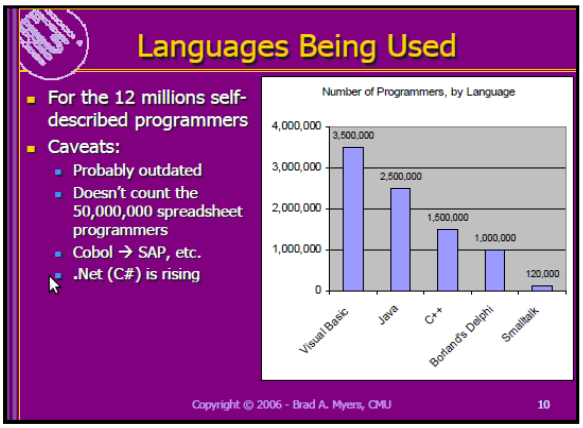
Anexo 3

Ejemplo de las Investigaciones acerca de los lenguajes de programación donde se toman a Java y C como lenguajes de programación a considerar:



Fuente: http://www.cs.cmu.edu/~bam/papers/EUPchi2006overviewColor.pdf





Anexo 4

Comprensión de lectura de un Lenguaje de Programación (A)

Estimado alumno, favor de seguir en orden las siguientes instrucciones:

1. Coloque la hora de inicio de la encuesta: _____

2. Asumir que el valor de peso es 60 y el valor de la estatura es 1.60

Resolver:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
      const float bajo_peso=18.5, normal = 25.0, sobre_peso =
      30.0;
      float peso, altura, bmi;

      printf("Ingrese su peso en kg:");
      scanf("%f",&peso);

      printf("Ingrese su altura en m:");
      scanf("%f",&altura);

      bmi = peso / pow(altura,2);

      if (bmi <= bajo_peso)
            printf("Tiene un peso debajo del promedio: %3.4f
      \n",bmi);
      else
            if (bmi <= normal)
                  printf("Tienes un peso promedio:
            %3.4f\n",bmi);
            else
                  if (bmi <= sobre_peso)
                        printf("Tiene sobrepeso: %3.4f\n",bmi);
                  else
                        printf("Ud. está obeso: %3.4f\n",bmi);
      getchar();

      system("PAUSE");

      return 0;
}
```

Respuesta que arrojaría la computadora: _____
3. Si ya escribió la respuesta, coloque la hora: _____

4. Evaluar del 1 al 5 (1 es más fácil y 5 más complicado)

1 2 3 4 5

5. ¿Qué parte del código le pareció complicada o no la comprendió?

_____

_____

6. Lenguaje(s) de programación que conoce:

_____

_____

7. Nota que sacó en IC:

Examen Parcial: _____

Promedio de prácticas: _____

Examen Final: _____

8. Edad: _____

9. Sexo: (F) (M)

¡Muchas gracias por su tiempo!

Favor de NO llenar esta parte

a) Resolvió el programa? Sí No

b) Contestó correctamente? Sí No

c) Tiempo de resolución:

De 1 – 3 minutos

De 3 – 6 minutos
Más de 6 minutos

Comprensión de lectura de un Lenguaje de Programación (B)

Estimado alumno, favor de seguir en orden las siguientes instrucciones:

1. Coloque la hora de inicio de la encuesta: _____

2. Asumir que el valor de peso es 60 y el valor de la estatura es 1.60

Resolver:

```
package bmi;
import java.util.Scanner;

public class Main {

 public static void main(String[] args)
 {
      double bajo_peso=18.5, normal=25.0, sobre_peso=30.0;

      Scanner sc=new Scanner(System.in);

      double peso, altura, bmi;

      System.out.print("Ingrese su peso: ");
      peso=sc.nextDouble();

      System.out.println("Ingrese su altura");
      altura=sc.nextDouble();

      bmi = peso/(Math.pow(altura,2));

      if(bmi <= bajo_peso)
            System.out.println("Tiene un peso debajo del
      promedio");
      else
            if(bmi <= normal)
                  System.out.println("Tiene un peso dentro del
            promedio");
            else
                  if(bmi <= bajo_peso)
                        System.out.println("Tiene un peso superior
                  del promedio");
                  else
                        System.out.println("Ud. tiene problemas de
                  obesidad");
      }
}
```

Respuesta que arrojaría la computadora:

_____

3. Si ya escribió la respuesta, coloque la hora: _____

4. Evaluar del 1 al 5 (1 es más fácil y 5 más complicado)

1 2 3 4 5

5. ¿Qué parte del código le pareció complicada o no la comprendió?

_____

_____

6. Lenguaje(s) de programación que conoce:

_____

_____

7. Nota que sacó en IC:

Examen Parcial: _____

Promedio de prácticas: _____

Examen Final: _____

8. Edad: _____

9. Sexo: (F) (M)

¡Muchas gracias por su tiempo!

Favor de NO llenar esta parte

a) Resolvió el programa? Sí No

b) Contestó correctamente? Sí No

c) Tiempo de resolución:

De 1 – 3 minutos
De 3 – 6 minutos
Más de 6 minutos

Comprensión de lectura de un Lenguaje de Programación (C)

Estimado alumno, favor de seguir en orden las siguientes instrucciones:

1. Coloque la hora de inicio de la encuesta: _____

2. Asumir que el valor de peso es 60 y el valor de la estatura es 1.60

Resolver:

```
main :: IO ( )
main = do

     putStr “Ingrese su peso:\n”
     peso <- getLine

     putStr “Ingrese su altura:\n”
     altura <- getLine

     print (bmiTell (read peso::Float) (read altura::Float))

bmiTell :: Float -> Float -> String

bmiTell peso estatura

     | bmi <= bajo_peso = "Tiene un peso debajo del
promedio"

     | bmi <= normal = "Tiene un peso dentro del promedio"

     | bmi <= sobre_peso = "Tiene un peso superior del
promedio"

     | otherwise = "Ud. tiene problemas de obesidad"

where bmi = peso /altura ^ 2

(bajo_peso, normal, sobre_peso) = (18.5, 25.0, 30.0)
```

Respuesta que arrojaría la computadora:

_____

3. Si ya escribió la respuesta, coloque la hora: _____

4. Evaluar del 1 al 5 (1 es más fácil y 5 más complicado)

1 2 3 4 5

5. ¿Qué parte del código le pareció complicada o no la comprendió?

_____

_____

6. Lenguaje(s) de programación que conoce:

_____

_____

7. Nota que sacó en IC:

Examen Parcial: _____

Promedio de prácticas: _____

Examen Final: _____

8. Edad: _____

9. Sexo: (F) (M)

¡Muchas gracias por su tiempo!

Favor de NO llenar esta parte

a) Resolvió el programa? Sí No

b) Contestó correctamente? Sí No

c) Tiempo de resolución:

De 1 – 3 minutos
De 3 – 6 minutos
Más de 6 minutos

Comprensión de lectura de un Lenguaje de Programación (D)

Estimado alumno, favor de seguir en orden las siguientes instrucciones:

1. Coloque la hora de inicio de la encuesta: _____

2. Asumir que el valor de peso es 60 y el valor de la estatura es 1.60

Resolver:

```
package main
import "fmt"

const (
    bajoPeso  = 18.5
    normal    = 25.0
    sobrePeso = 30.0
)
func main() {
    var peso, altura float64
    fmt.Print("Ingrese su peso en kg: ")
    fmt.Scanln(&peso)

    fmt.Print("Ingrese su altura en m: ")
    fmt.Scanln(&altura)

    switch bmi := peso / (altura * altura); {

     case bmi <= bajoPeso:
          fmt.Println("Tiene un peso debajo del promedio")
     case bmi <= normal:
          fmt.Println("Tiene un peso dentro del promedio")
      case bmi <= sobrePeso:
          fmt.Println("Tiene un peso superior al promedio")
     default:
          fmt.Println("Ud. tiene un problema de obesidad")
    }
}
```

Respuesta que arrojaría la computadora:

_____

3. Si ya escribió la respuesta, coloque la hora: _____

4. Evaluar del 1 al 5 (1 es más fácil y 5 más complicado)

1 2 3 4 5

5. ¿Qué parte del código le pareció complicada o no la comprendió?

_____

_____

6. Lenguaje(s) de programación que conoce:

_____

_____

7. Nota que sacó en IC:

Examen Parcial: _____

Promedio de prácticas: _____

Examen Final: _____

8. Edad: _____

9. Sexo: (F) (M)

¡Muchas gracias por su tiempo!

Favor de NO llenar esta parte

a) Resolvió el programa? Sí No

b) Contestó correctamente? Sí No

c) Tiempo de resolución:

De 1 – 3 minutos
De 3 – 6 minutos
Más de 6 minutos

Resultados del Horario 1

Tiempo	Evaluacion	Acierto	Sexo	Edad	Lenguaje de programacion
5	3	No	M	19	Java
4	5	No	M	26	Java
3	5	No	M	17	C
10	4	No	M	18	Java
6	5	No	M	18	C
5	1	Si	M	16	Haskell
7	4	Si	M	17	C
5	2	No	M	18	C
4	4	SI	M	18	Haskell
5	3	SI	M	16	Haskell
4	3	SI	M	16	Haskell
5	3	SI	M	18	GO
6	3	No	M	17	Java
3	4	Si	M	17	GO
1	5	No	M	17	Haskell
3	5	No	M	17	Java
3	4	No	M	18	Haskell
2	2	Si	M	17	GO
10	4	No	M	19	Java
3	4	Si	F	18	GO

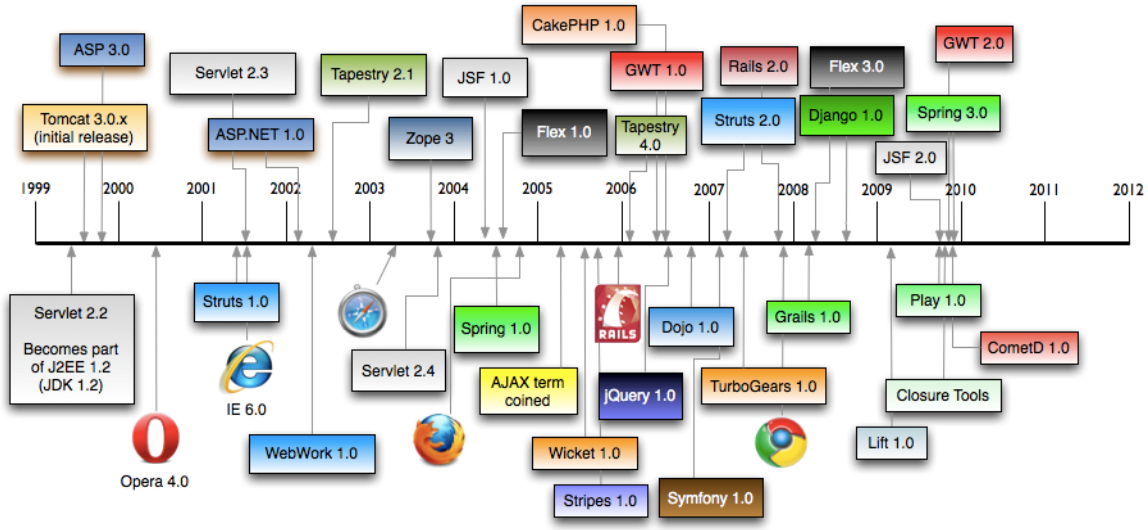
Resultados del Horario 2

4	5	Si	M	17	Haskell
10	3	No	M	18	C
3	2	SI	M	18	GO
4	2	No	M	17	Java
18	4	No	F	18	Java
3	4	Si	M	17	Haskell
7	3	Si	M	18	Java
5	3	No	M	19	Java
4	4	Si	M	18	Haskell
5	2	Si	F	19	GO
3	2	Si	M	18	GO
5	2	Si	F	18	GO
5	4	No	M	18	Haskell
7	2	No	M	19	Java
4	2	Si	M	17	GO
2	3	Si	F	18	Java
3	3	No	M	18	Haskell
2	3	No	M	18	Haskell
2	2	No	F	18	C
4	5	No	M	19	Java
3	4	No	F	18	Java
5	4	No	M	21	C
5	3	Si	M	19	GO
5	1	Si	F	18	Haskell
5	4	No	M	22	GO
3	4	No	M	18	C
3	3	No	M	18	Haskell
5	4	No	M	18	Java
10	3	Si	F	18	GO
6	3	No	M	18	Java
7	5	Si	M	18	C
2	4	Si	M	20	Haskell
2	3	Si	M	21	C
5	4	Si	M	17	Haskell
3	2	No	M	18	GO
7	4	No	M	18	Java
4	3	No	F	18	GO
2	5	No	M	19	C
2	2	Si	M	18	GO
3	3	No	M	17	Haskell
3	4	No	F	18	C

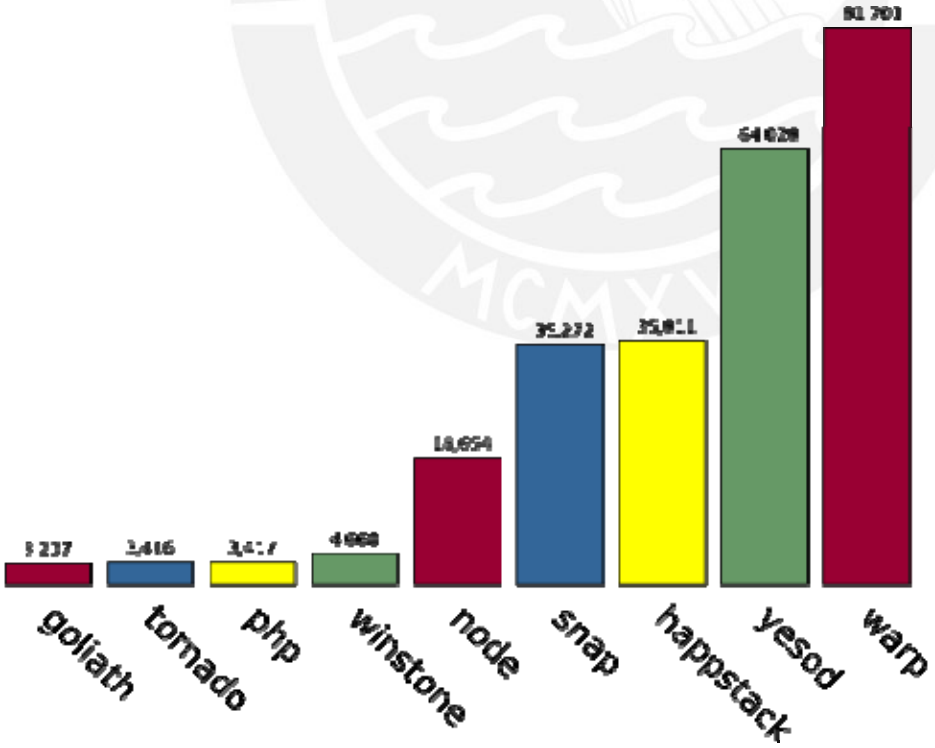
7	3	Si	F	18	C

Anexo 5

Frameworks timeline, comienza con Servlet 2.2 con JDK 1.2 a mediados del año.



Fuente: http://java.dzone.com/articles/my-future-web-frameworks



Fuente: http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks