DEFINICIÓN DE UN FRAMEWORK PARA LA APLICACIÓN DEL PENSAMIENTO COMPUTACIONAL EN LA CONSTRUCCIÓN Y PRUEBA DE ALGORITMOS

RICARDO LEÓN ISAZA DAVID

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERÍA

MAESTRÍA EN INGENIERÍA

ESPECIALIDAD EN TECNOLOGÍAS DE INFORMACIÓN PARA LA EDUCACIÓN

MEDELLÍN

2015

DEFINICIÓN DE UN FRAMEWORK PARA LA APLICACIÓN DEL PENSAMIENTO COMPUTACIONAL EN LA CONSTRUCCIÓN Y PRUEBA DE ALGORITMOS

RICARDO LEÓN ISAZA DAVID

Proyecto de investigación para optar el título de Maestría en Ingeniería con especialidad en Tecnologías de Información para la Educación

Director

MAURICIO TORO BERMÚDEZ

UNIVERSIDAD EAFIT

ESCUELA DE INGENIERÍA

MAESTRÍA EN INGENIERÍA

ESPECIALIDAD EN TECNOLOGÍAS DE INFORMACIÓN PARA LA EDUCACIÓN

MEDELLÍN

2015

DEDICATORIA

AGRADECIMIENTOS

Muy especialmente a mi asesor, ya que me tendió su mano y me orientó con paciencia y dedicación, cuando muchos otros solo se limitaron a mirar hacia otro lado.

A la Alianza Futuro Digital, por su apoyo económico en el logro de este nuevo peldaño en mi carrera profesional.

TABLA DE CONTENIDOS

LISTA DE ILUSTRACIONES

LISTA DE TABLAS

"*Minino de Cheshire, empezó Alicia tímidamente – pues no estaba del todo segura de si le gustaría este tratamiento – pero el Gato no hizo más que ensanchar su sonrisa, por lo que Alicia decidió que sí le gustaba. Minino de Cheshire, ¿podrías decirme, por favor, qué camino debo seguir para salir de aquí?*

Esto depende en gran parte del sitio al que quieras llegar - dijo el Gato.

No me importa mucho el sitio... -dijo Alicia.

Entonces tampoco importa mucho el camino que tomes - dijo el Gato.

... siempre que llegue a alguna parte - añadió Alicia como explicación.

¡Oh, siempre llegarás a alguna parte - aseguró el Gato -, si caminas lo suficiente!"

Caroll (2003).

Introducción

El desarrollo de programas orientados al procesamiento de información a través de computadoras se constituye en el pilar central de muchos perfiles de formación en las instituciones de educación superior que ofrecen programas formativos afines a las ciencias computacionales, haciendo de este un conocimiento necesario en cualquiera de los niveles de formación posterior a la educación secundaria, e incursionando inclusive en los niveles de media técnica (grados décimo y undécimo) siempre con el objetivo de brindar al estudiante una fundamentación sólida que le permita desenvolverse en el desarrollo del software, indistintamente de la herramienta o el entorno de trabajo seleccionado para tal fin.

A pesar de que en la teoría este objetivo sea alcanzable, a lo largo de los años se evidencia que la programación – centro del universo correspondiente al perfil de egresado del estudiante – se convierte en su más acérrimo contendor. Puede observarse en escritos como los presentados por Galdeano (2002), Ala-Mutka (2004) y Haden (2003), que la programación se erige como el némesis de cualquier estudiante que opte por orientar su proceso formativo en carreras ligadas a las ciencias computacionales, siendo una piedra en el zapato su apuro en el desarrollo de unas competencias fundamentales en la solución de problemas algorítmicos, sin el logro de unos conocimientos básicos en sus estructuras de control, su aplicabilidad en problemáticas reales, o el incipiente desarrollo de unas habilidades de análisis requeridas. Lo anterior conlleva según Ohland (2014) a que los estudiantes se desgasten en su aprendizaje, obligándoles en algunos casos a abandonar sus estudios con miras a incursionar en otros horizontes académicos, bien sea por decisión propia o simplemente forzados a hacerlo como consecuencia de un retiro académico.

Existen diversos trabajos que han indagado en posibilidades de acción para la enseñanza de la algoritmia, un ejemplo de ello sería el caso presentado por Mendes (2012). Mientras algunos otros se han enfocado en analizar el aprendizaje de un lenguaje o el uso de herramientas de información para soportar el análisis de problemas, como es el caso de Mahnic (1997). Por medio de este trabajo se busca establecer un punto medio entre ambos enfoques, ya que se pretende observar la posibilidad de orientar el proceso de enseñanza-aprendizaje a través del entendimiento del problema y su disgregación en elementos funcionales interrelacionados, empleando para ello un enfoque de análisis denominado *Pensamiento Computacional*.

El *Pensamiento Computacional* en su forma actual es una estructura de desarrollo cognitivo propuesta de Jeanette Wing (2006) encaminada al desarrollo de habilidades que permitan la solución de problemas a través del uso de conceptos fundamentales de las ciencias computacionales. Esto implica tomar las características y requisitos del problema a través de una visión algorítmica de este, con sus respectivas secuencias de pasos y la orientación a un conjunto de soluciones posibles, logrando articular enfoques y niveles de abstracción diversos, que en

consideración de los recursos disponibles y los resultados esperados, permitan la elección de aquella opción que se considere adecuada, en conformidad a su eficiencia, confiabilidad y seguridad.

Es importante aclarar que si bien el Pensamiento Computacional no surge con la finalidad específica de desarrollar competencias en programación, es posible emplearlo para formar capacidades de abstracción y formulación de soluciones con corte algorítmico a problemas donde pueda emplearse un soporte intensivo de la informática. De tal suerte que el presente trabajo confía en estructurar su aplicabilidad al proceso de enseñanza-aprendizaje, orientándole específicamente al entendimiento de problemas y la construcción de algoritmos mediante el fortalecimiento de la capacidad cognitiva del estudiante, facilitando el análisis y la abstracción de las características del problema.

Este trabajo se divide en **cuatro** capítulos, cada uno de los cuales permite sustentar de manera evolutiva el desarrollo de las preguntas de investigación y los objetivos planteados presentando algunas conclusiones preliminares. Así, el **capítulo 1** presenta de manera general la problemática identificada en el proceso de enseñanza-aprendizaje de los conceptos y el pensamiento algorítmico, enumera los objetivos que delimitarán el desarrollo del trabajo y su campo de acción, brinda un acercamiento a los conceptos y referencias que darán soporte al logro de los objetivos planteados, delimitando finamente los aspectos metodológicos y de proceso que orientarán el proceso investigativo. **El capítulo 2** establece las prácticas más comunes en la enseñanza algorítmica, facilitando así la extrapolación de sus actividades a las planteadas para el Pensamiento Computacional, para ello se identifica por medio de un ejercicio de intersección los principios que le dan sustento a través de los enfoques de diversos autores, estableciendo para ello la aplicabilidad de los principios hallados en el proceso de enseñanza-aprendizaje de algoritmos, basándose en planteamientos de diversos autores con énfasis en: Algoritmia, Lógica de Programación e Ingeniería de Software, definiendo finalmente la manera en que la aplicación de las premisas identificadas con anterioridad orientan el desarrollo de las competencias algorítmicas en el estudiante. El **capítulo 3** define la estructura por fases del Framework que articulará el desarrollo de las competencias algorítmicas, su entorno y necesidades particulares para la aplicación el Framework en ambientes académicos, por lo que se presenta además un acercamiento breve al impacto que se espera en las instituciones en articulación con el Politécnico Jaime Isaza Cadavid. El **capítulo 4** denota algunas consideraciones finales del trabajo, sus conclusiones y el trabajo futuro derivado. Al final se presentan las referencias bibliográficas que dan soporte al desarrollo del trabajo.

1. Planteamiento del Problema

1.1. Planteamiento del Proyecto

1.1.1. Generalidades

La "Sociedad de la Información" considera de vital importancia aspectos como la generación y el uso que se puede dar a la información. En ella, los actores públicos y privados impulsan políticas de apropiación tecnológica y masificación en el uso de las Tecnologías de Información y Comunicación (TIC) aplicadas al mejoramiento de los procesos productivos y de servicios. Algunos enfoques como los presentados por James (2012) y Popescu (2013) acreditan al desarrollo para dispositivos móviles, los servicios web, los intercambios de información y la construcción de conocimiento colectivo, como algunas de las áreas de mayor impacto social en los años por venir, que al integrarse con lo descrito por el Consorcio de Habilidades Indispensables para el Siglo XXl (2007), pueden adicionarse – entre otras – las competencias en pensamiento crítico y la solución a problemas a través de la separación de sus rasgos, cualidades o características más relevantes, con el objetivo de analizarles de forma separada y considerarlos de manera directa en su esencia a través de la interacción (proceso conocido como abstracción), como obligantes de una transformación en el modelo pedagógico y los objetivos de aprendizaje actuales.

Sin embargo, a pesar de lo vanguardista y coherente que pueda resultar este horizonte, existen diversas dificultades en los procesos de enseñanza-aprendizaje con relación a las asignaturas o módulos que soportan las competencias ligadas a la abstracción y la solución de problemas, principalmente mediante la algoritmia, conforme lo expresan Lahtien (2005), Mow (2008) y Milne (2002).

Estos problemas trascienden los modelos de formación y hacen que los futuros profesionales en ciencias computacionales encuentran laborioso y complejo el proceso de desarrollo de software, manifestándose a través de limitaciones en la apropiación de los conceptos algorítmicos fundamentales para la resolución de problemas. Dicha problemática es generalizada, y evidente a nivel mundial según pude extractarse lo descrito por Lee (2006) y Pérez (2013). Sus textos sugieren que numerosos estudiantes deben repetir los cursos relacionados con algoritmia, fundamentos de programación, o pensamiento analítico y sistémico. Por otro lado se encuentran aquellos estudiantes que según Ohland (2014) abandonan simplemente la carrera por no lograr superar satisfactoriamente dichos cursos, encontrando que la comprensión algorítmica que les es esquiva, sin detenerse a pensar que ésta hace parte de su cotidianidad

En un contexto más cercano, es posible tomar como referencia datos relacionados con estudiantes inscritos, sus respectivas notas y las cancelaciones al interior de la Media Técnica Articulada, la Técnica Profesional en Programación de Sistemas de Información, la Tecnología en Sistematización

de Datos y la Ingeniería Informática al interior del Politécnico Colombiano Jaime Isaza Cadavid para evidenciar casos en los cuales los estudiantes no completan dichos cursos, o lo hacen en algunos casos haber evidenciado un logro de los objetivos de aprendizaje requeridos, acarreando en muchos de ellos una desconsideración al desarrollo de software como una alternativa viable en su desempeño profesional. En este contexto se pretende emplear el Pensamiento Computacional como base en el desarrollo de un Framework que permita orientar el desarrollo de actividades que motiven al estudiante en el desarrollo de sus competencias algorítmicas, complementando los objetivos de aprendizaje propuestos en sus respectivos módulos o asignaturas en el Politécnico Colombiano Jaime Isaza Cadavid.

1.1.2. Descripción del Problema

En los diferentes programas de formación técnica, tecnológica y profesional ligados a las ciencias computacionales, las asignaturas o módulos relacionados con algoritmia hacen parte del ciclo de formación básico. Conforme lo presentan Mahnic (2003) y Mow (2008) dichas asignaturas son orientadas a lograr en el estudiante la capacidad de resolver problemas de programación a través del uso de estructuras algorítmicas que soporten la construcción de código en lenguajes de alto nivel. En este aspecto, Garner (2002) explica que la construcción de dichas estructuras algorítmicas requiere la integración de conocimientos de diversa índole, siendo el razonamiento lógico, el modelamiento matemático y el pensamiento abstracto aquellos con mayor nivel de relevancia.

Sin embargo, se presentan situaciones donde los estudiantes no alcanzan a cubrir la totalidad de los objetivos de aprendizaje al culminar los módulos algorítmicos. Esto, según lo presentado por Pérez (2008) conlleva a situaciones donde los educandos deben repetirlos en diversas ocasiones, así como el caso de otros que obtienen calificaciones satisfactorias, pero en todo caso con poco aprendizaje significativo, y que según puede inferirse de lo propuesto por Navarridas (2002)consecuencia del uso de una escala evaluativa y la definición de un umbral mínimo necesario para considerar como aprobada una asignatura.

Además, el cambio de paradigmas al que se enfrenta la industria ha llevado da la academia a emplear lenguajes que iniciaron con una estructura lineal y compacta de expresión (p.ej., Pascal, C) migrando hacia otros soportados en estructuras donde los objetos determinan la arquitectura de construcción (siendo el caso de C++ y Java), hasta llegar a lenguajes con una gran cohesión a las estructuras visuales y un enfoque más integrado con su entorno de desarrollo (como sería el caso de Microsoft Visual Basic) planteando la inquietud acerca de cuál sería el más adecuado para lograr la representación codificada de las estructuras algorítmicas enseñadas. En este respecto, Mahnic (2003) presenta un dilema aún mayor al considerar que la educación en la programación debe decidir entre enseñar los procesos algorítmicos aplicados a la programación estructurada, orientada a objetos, aspectos, orientada a la web, o integrada a ambientes de desarrollo, siendo este último el más complejo de todos los escenarios, ya que desliga al desarrollador de la carga de generación de código específico, así que cuestiona acerca de ¿Cuál es realmente el aprendizaje significativo en la generación de código basado en algoritmos a ese nivel, si el entorno de

desarrollo genera la mayoría del código necesario para el logro de los objetivos? O quizás ¿si el enfoque de enseñanza se centra en el aprendizaje en el uso de un lenguaje de programación, más que en el desarrollo de las competencias lógicas ligadas a la algoritmia?

Si bien no existe un estudio formal que soporte dicha afirmación, Milne (2002), Ala-Mutka (2004), Mahnic (1997) y Mow (2008), entre otros autores, presentan algunas iniciativas de análisis que permiten identificar las vicisitudes que rodean la problemática del aprendizaje en las estructuras algorítmicas y de programación, pudiendo traducirse en un sentir generalizado entre estudiantes y profesores que las asignaturas o módulos relacionados con algoritmia son aquellos con mayor nivel de complejidad de la carrera, o son los "cedazos" o el "filtro" para los estudiantes, en su mayoría por que los estudiantes no logran definir modelos mentales de comportamiento y aplicación de los algoritmos, de su representación, o de la manera en que dichas construcciones operan en la memoria de los dispositivos computacionales, desencadenando problemáticas colaterales como la incomprensión de los conceptos algorítmicos más complejos, la deserción temprana de los programas de formación o el incubamiento de un desinterés por asignaturas relacionadas con la programación, siendo este último suficientemente complejo si se considera el hecho que el énfasis de muchos de estos programas de formación se orienta al desarrollo de software.

Adicionalmente, al analizar algunas iniciativas como las presentadas por Burkhardt (1997) y Blackwell (1996) es posible establecer la necesidad de construir modelos mentales que faciliten la realización de validaciones a las estructuras algorítmicas construidas, lo cual es verdaderamente complicado si se toma por sentado el planteamiento de Burgess (1994) al formular que la falta de experiencia de los estudiantes al enfrentarse a situaciones de resolución de problemas no solo limita su capacidad de construir estructuras algorítmicas, sino que impide que estas puedan ser validadas en sus calidades sistémicas de manera eficiente.

Estos y otros interrogantes han buscado solución a través de la exploración de diversos enfoques pedagógicos que permitieran concentrar sus esfuerzos en direccionar los procesos de enseñanza-aprendizaje. Puede tomarse como punto de referencia inicial el modelo instruccional, mencionado por Bárcena (2003) y aplicado en muchas instituciones educativas de diverso nivel, en el cual se define una relación de transmisión de información magistral de los conceptos a tratar en algoritmia con una participación pasiva del estudiante enfocada principalmente a un proceso de recepción y repetición, más que en la comprensión de la información y construcción de un conocimiento duradero.

Dada la problemática que aqueja al proceso formativo y que fundamenta el desarrollo de este trabajo, podría entenderse la necesidad que los procesos de enseñanza-aprendizaje logren de igual forma una evolución natural en relación con la solución de problemas y la algoritmia. Cobijados en esta orientación, Ertmer (1993) lo manifiesta como algo que debiera involucrar al estudiante para permitirse obtener resultados significativos. Enfoques como los presentados por

Barriga en sus trabajos de (2003) y (2002) entrevén la necesidad de facilitar al estudiante un conjunto de bases que le permitan establecer un enfoque propio para la resolución de problemas en un enfoque constructivista, de manera que pueda desarrollarse un conocimiento aplicado y duradero, bajo el contexto cognitivista, orientándose al desarrollo de conocimientos al centrarse en la posibilidad que el educando sea un participante activo de dicho proceso, bien sea al lograr un cuestionamiento del comportamiento fenomenológico de lo que estudia (p.ej., interrogantes acerca de la manera en que puede ser construido el algoritmo idóneo para la solución de un problema) o involucrándose con sus pares en la construcción de un conocimiento social fundamentado en la retroalimentación de sus actos (p.ej., mediante una construcción de código y/o su revisión en pares).

Tomando como fundamento un enfoque constructivista, surge un modelo de pensamiento que busca desarrollar en el estudiante la capacidad de enfrentar diversas problemáticas a través de una estructuración similar a la formulación de un problema computacional, orientando la definición de su solución a estructuras algorítmicas. Este enfoque, planteado en su forma moderna por Wing (2006) se conoce como Pensamiento Computacional, y será el fundamento en el desarrollo del presente trabajo.

Para abordar el análisis de los principios del Pensamiento Computacional, será necesario identificar aquellos cuya aplicabilidad al aprendizaje en el desarrollo de algoritmos tenga soporte a la luz de las competencias esperadas en el estudiante. Algunos autores como Joyanes (2003), sostienen que un programador debe estar en capacidad de resolver problemas a través de un modo sistémico y riguroso, empleando algoritmos y estructuras de datos; convirtiéndose en una habilidad fundamental en la construcción de estructuras de control sobre un lenguaje de programación, y de este modo, el problema sobre el que se desarrollará el presente trabajo corresponde a identificar aquellos puntos clave en la formación básica del estudiante que pueden ser fortalecidos a través del Pensamiento Computacional, permitiéndole realizar una construcción y verificación efectivas de soluciones algorítmicas.

1.2. Objetivos

1.2.1. General

Construir un Framework que permita la aplicación de los principios del pensamiento computacional a la construcción y verificación de algoritmos en procesos de aprendizaje.

1.2.2. Específicos

- Caracterizar prácticas para la enseñanza de los algoritmos a nivel internacional.
- Identificar aquellos principios del pensamiento computacional aplicables al desarrollo del trabajo.
- Plantear el Framework de aplicación de los principios del pensamiento computacional

- Definir un marco de aplicación del Framework planteado.
- Definir la aplicabilidad del Framework planteado al interior del proceso de articulación de la Media Técnica con el Politécnico Colombiano Jaime Isaza Cadavid.

1.2.3. Preguntas de Investigación

- ¿Cómo aplicar los principios del pensamiento computacional al proceso de aprendizaje en la construcción y verificación de las soluciones planteadas a través de algoritmos?
- ¿De qué manera puede aplicarse los principios del pensamiento computacional al proceso de articulación de la Media Técnica con el Politécnico Colombiano Jaime Isaza Cadavid?

1.3. Marco Referencial

A continuación se presenta un recorrido por diversos elementos que permitirán cimentar el presente trabajo y la manera en que estos pueden ser empleados para desarrollar en el estudiante competencias orientadas a la construcción y prueba de algoritmos.

Así, la **sección 1.3.1** busca explorar diversas definiciones del concepto de modelo y abstracción, evidenciando la necesidad de establecer puntos de referencia para su aplicación en la solución de problemas. La **sección 1.3.2** presenta una definición de algoritmo a través de la observación de diversas concepciones, su representación, estructuras de control y proceso de verificación de los resultados obtenidos. La **sección 1.3.3** relaciona al algoritmo con el concepto de programación, sus objetivos y la manera en que dan solución a problemáticas de usuario. La **sección 1.3.4** habla acerca del concepto de competencias algorítmicas y la manera en que estas deben ser vistas en los estudiantes. Finalmente, la **sección 1.3.5** habla acerca del pensamiento computacional y sus principios.

1.3.1. El concepto de modelo

El ser humano siempre se ha cuestionado acerca de la realidad que le rodea. Observación y análisis de las situaciones cotidianas han sido las herramientas que ha empleado para lograr descripciones que le permiten explicar comportamientos o situaciones que le resultan atractivas, interesantes o útiles, dependiendo del contexto en el que se desenvuelve. De esta forma se plantea entonces la necesidad de definir puntos de referencia que le permitan analizar los componentes del universo circundante, construyendo un plano de representación cuyas características sean en proporción, idénticas a la del elemento de la realidad que es observado, esta caracterización de sus puntos más relevantes le permite capturar la esencia y facilita su estudio.

Wiener (1945) enfatiza en la importancia de emplear abstracciones para representar elementos pertenecientes al universo con el fin de lograr el entendimiento de sus características, la manera en que se compone y la interacción que tiene con los demás elementos que le rodean. La abstracción puede entenderse entonces como la posibilidad de enumerar cada una de las

características de un elemento del universo (p.ej., un sistema) que sean relevantes a un contexto particular, logrando diversos niveles de fidelidad frente al original, permitiendo su manipulación al tomar como base los aspectos más sobresalientes del sistema y omitiendo aquellos que se consideran superfluos. De esta forma es posible describir o entender su comportamiento en situaciones con características controladas, algo que se conoce como "proyección del modelo". En la **Ilustración 1** se representa el concepto de niveles de abstracción para un reloj de pulso. Allí se toma como base el elemento con mayor cantidad de detalles (izquierda) y se avanza hacia un nivel de abstracción mayor (derecha) o viceversa. De esta forma, la cantidad de detalles tienden a aumentar o disminuir, pero en todo caso se conservan solo aquellos más relevantes a un contexto particular.

Ilustración 1. Representación de los niveles de abstracción de un elemento "Reloj de Pulso". La cantidad de detalles disminuye en la medida que aumenta el nivel de abstracción. (Fuente: http://mydrawingblog.wordpress.com)

Dicha proyección del modelo se cimenta en el hecho que la manipulación de sus características admita la obtención de resultados cuyas conclusiones sean aplicables al ámbito real, permitiendo realizar un abordaje analítico de los resultados y facilitando su modificación en consideración de las necesidades del medio en el que se desenvuelve. Es en este punto que el modelo se convierte en un compendio de aspectos que deben ser tenidos en cuenta para llevar a cabo una tarea o conjunto de ellas, siendo quizás una definición más simple la entregada por la Real Academia de la Lengua Española (2013), que en su sitio web habla del modelo como un "*Arquetipo o punto de referencia para imitarlo o reproducirlo*" o de una "*Representación en pequeño de alguna cosa*".

Ahora bien, si se pudiese alterar el enfoque de los modelos y se les tomara como punto de llegada en lugar de como punto de partida, como punto de comparación para la evaluación en lugar de como punto de referencia para la construcción de elementos basados en las características que representa, puede decirse que su utilidad radica en el hecho de permitir valorar los niveles de proximidad con otros elementos existentes en la realidad, siendo el nivel de generalidad del modelo la posibilidad que más elementos de características similares puedan ser representados como un universo bajo la misma estructura del modelo. Este esquema de representación podría considerarse una generalización del dominio del modelo, y permite ampliar su espectro de representación sacrificando la existencia de características aplicables a ámbitos particulares. En la **Ilustración 2** se presenta este concepto al incorporar visualmente el tiro parabólico empleando como trasfondo el juego "Angry Birds", el modelo de tiro altera la trayectoria del pájaro al evidenciar cambios en las variables de la ecuación matemática. Dicha ecuación es aplicable a

cualquier contexto que cumpla con las características básicas de movimiento horizontal y vertical a través de la aplicación de fuerzas sobre un objeto. En la medida que sea más específico al contexto del juego, la ecuación tomará un papel menos amplio, y solo será aplicable a sus características propias, como lo es por ejemplo, la necesidad de derribar construcciones y obstáculos con tiros no necesariamente directos causando la mayor cantidad de daño colateral posible; caso que sería impensable en el contexto armamentista, ya que en este último se pretende lograr niveles superiores de precisión que eviten al máximo el daño colateral, conforme se expresa en la teoría y principios de la balística, expresados en los escritos de Swan que datan de (1983) y (1991).

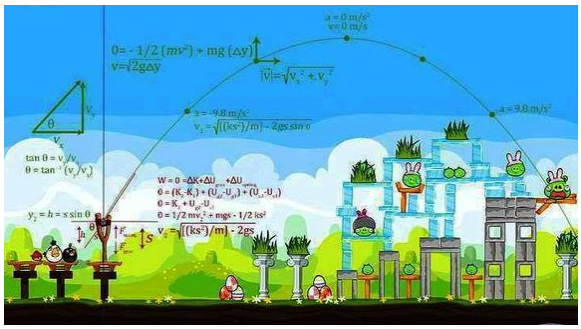


Ilustración 2. Representación del modelo matemático del tiro parabólico representándolo en el juego AngryBirds. (Fuente: http://tarea-info-emmanuelsaltos.blogspot.com/)

Como resultado es posible decir entonces que los modelos surgen de la necesidad de explicar hechos o acciones en el universo, que pueden servir como punto de referencia para definir un comportamiento esperado bajo unas circunstancias controladas, siendo sus resultados objeto de análisis ya que no son ciertos ni falsos en sí, sino que obedecen a la posibilidad de enfrentar resultados de características específicas toda vez que se realice una alteración de las características relevantes al modelo, y que según Forisek (2012) se debe validar si los resultados obtenidos – o esperados, según sea el caso – son aplicables a la realidad particular que se define o simplemente requiere de un afinamiento del modelo conforme se obtiene una retroalimentación positiva o negativa.

1.3.2. El concepto de Algoritmo

Hasta este punto se ha descrito la manera en que el proceso de observación directa de un sistema permite describir su comportamiento a través de la definición de un modelo, y como la enumeración y manipulación de sus características facilita la predicción de la manera en que este se comportará. Ahora es necesario considerar el hecho que muchos de estos modelos son extrapolados a un plano tecnológico a través de la construcción de dispositivos digitales, los cuales son diseñados para realizar gran cantidad de actividades: desde operaciones matemáticas simples, manipulación de grandes cantidades de datos, realizar la mediación en procesos de comunicación, hasta realizar el apoyo en el control de vuelo en aeronaves tripuladas. Cada uno de estos dispositivos tiene su propia estructura física, y en algunos casos su diseño obedece al logro de un fin específico, separándose de las arquitecturas genéricas compatibles con la de los

microcomputadores disponibles hoy en día. Sin embargo todas tienen algo en común: realizan la transformación de operaciones triviales de entradas de datos a través de secuencias de control que pueden resumirse en algoritmos.

Según Harel (2004), los algoritmos pueden ser observados desde la perspectiva de una receta culinaria para la preparación de un pastel. Así, los datos de entrada corresponderían a los ingredientes necesarios para la preparación, mientras que el pastel sería el resultado obtenido o la salida esperada del proceso, siendo entonces el algoritmo la receta que orienta la transformación de los ingredientes, estableciendo cada uno de los pasos a seguir y permitiendo la obtención del resultado final. Los algoritmos que son relevantes a un proceso en particular y que hacen parte de un universo funcional en ocasiones tangible, como es el caso de las aplicaciones inmersas en un teléfono celular inteligente o un dispositivo de control del entorno, o en ocasiones intangible, como el establecimiento de enlaces de intercambio de datos entre dispositivos, se consideran parte de un software que les da vida. Así, puede entenderse al software como la sumatoria de un conjunto de algoritmos, desde su perspectiva más simple. En la **Ilustración 3** es posible evidenciar la correspondencia existente entre la descripción natural de los pasos necesarios para cocinar un pastel, el algoritmo construido para representarlo y su respectiva codificación.

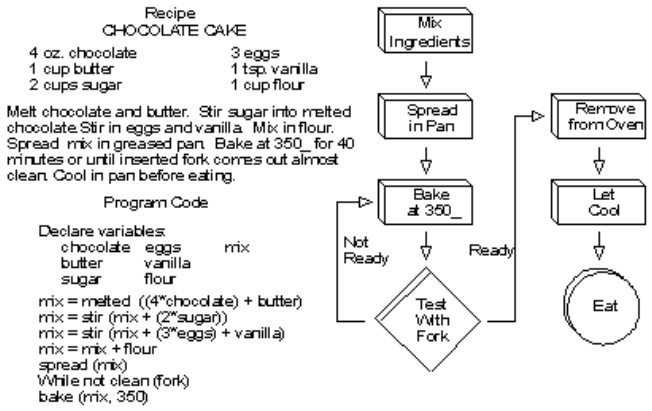


Ilustración 3. Tres enfoques para la representar el proceso de cocinar un pastel: la receta, el algoritmo y su codificación. (fuente: www.gmtel.net)

Así, los algoritmos pueden asociarse a gran cantidad de actividades cotidianas ya que constituyen una serie de pasos finitos, que siguen un orden específico, y que permiten lograr un objetivo discreto. Tómese la definición presentada por Weisstein (2013) donde los algoritmos se componen de una serie de instrucciones, las cuales orientan el desarrollo de un procedimiento o facilitan la solución de un problema, siempre enfocados en el hecho que las instrucciones deben ser completadas en algún punto. Esto hace que procesos como el reemplazo de una llanta desinflada, la preparación de un café, leer un libro, construir un modelo a escala, llenar un formulario, hasta inclusive adquirir una entrada a cine, sean poseedores de una "receta" que facilite su realización y demarquen un punto de inicio y otro de terminación. Dicha receta puede ser relacionada de

manera directa con el uso de tecnologías de información que le soporten, otras requieren de un esfuerzo mental mayor en su asociación, pero lo realmente importante radica en la comprensión de las necesidades de cada caso particular y la manera en que esto hace que se evidencien los pasos necesarios para dar una solución adecuada. Entonces, la composición de un algoritmo puede estructurarse a partir de la descripción de un conjunto de acciones que conlleven a obtener la solución a un problema identificado, haciendo que su esencia sea independiente de su implementación en un lenguaje de programación.

De igual forma, Moschovakis (2001) presenta un planteamiento en el cual especifica claramente que la definición rigurosa de algoritmos en la literatura de las ciencias computacionales se identifica generalmente con la idea de las máquinas abstractas y modelos matemáticos de computadores, sin relación directa con un lenguaje de implementación o entorno de desarrollo particular. Eso no quiere decir que no pueda lograrse una representación de un algoritmo a través del uso de un lenguaje de programación particular de alto nivel (p.ej., C++ o JAVA), sino que estas representaciones, conforme lo plantea Sedgewick (2011) facilitan la labor de verificar su efectividad requerida, sobre la premisa que todo código perteneciente a un lenguaje de programación es solo una de las posibles representaciones de un algoritmo. Para tal efecto obsérvese como en la **Ilustración 4** se presenta el algoritmo de Euclides – empleado para encontrar el máximo común divisor de dos números – de una manera descriptiva (en inglés) y su correspondiente representación algorítmica implementada en el lenguaje de programación JAVA. En el primer caso, el algoritmo de Euclides explica la manera en que se deben llevar a cabo las acciones para lograr obtener el resultado, y su correspondiente representación en el lenguaje de alto nivel es solo una de las posibles maneras de implementar dicho algoritmo, ya que su estructura puede verse sustancialmente modificada en el momento que se elija implementarlo en un lenguaje diferente.

Compute the greatest common divisor of two nonnegative integers **p** and **q** as follows: If **q** is **0**, the answer is **p**. If not, divide **p** by **q** and take the remainder **r**. The answer is the greatest common divisor of **q** and **r**.	`public static int gcd(int p, int q)`
	`{`
	`if (q == 0)`
	`  return p;`
	`int r = p % q;`
	`return gcd(q, r);`
	`}`

Ilustración 4. Comparativa de la descripción de un algoritmo y su implementación en un lenguaje de alto nivel como C++. (Fuente: Sedgewick (2011))

Entonces, toda definición de una secuencia de eventos debe ser una tarea estructurada, que permita una comprensión real de las características del problema y la definición de una solución acorde. En ese sentido, Baeza (1997) indica que la construcción de algoritmos es una labor que se compone de diversas etapas: en primer lugar, se hace necesario construir un modelo que esquematice las características más relevantes del problema a resolver, diseñando posteriormente la secuencia de pasos que conlleve al logro de una posible solución, representándola según sus

necesidades particulares. Posteriormente, dicha solución es analizada con el fin de establecer su nivel de eficiencia y efectividad, siendo traducida finalmente a un conjunto de instrucciones que serían interpretadas por un computador.

En este orden de ideas, la definición del modelo debe ser lo suficientemente robusta para abarcar todos y cada uno de los posibles datos de entrada y las funcionalidades de transformación que establezcan su capacidad computacional, el diseño de la solución se fundamenta en la utilización de diversos métodos de representación que faciliten su documentación, el análisis del algoritmo implica el seguimiento a la distribución que se haya dado a las diferentes secuencias de control que le componen y la manera en que estas son empleadas.

1.3.2.1. Secuencias de Control en los Algoritmos

Desde la idea fundamental del algoritmo como colección ordenada de pasos, es la secuencialidad lo que ha permitido hasta ahora esquematizar el conjunto de acciones que conllevan a logro de un objetivo discreto, enmarcados al interior de un inicio y un resultado final. Sin embargo, existen acciones que si bien deben participar de forma ordenada en la construcción del resultado, se hace necesario realizarlas en más de una ocasión con el fin de lograr un hito intermedio al interior de la secuencia, y por otra parte, existen casos en los que se hace evidente la necesidad de validar ciertas condiciones de igualdad, con el fin de determinar variaciones en el curso de acción que normalmente se seguiría.

En este sentido Gayo (2013) y Toledo (2002) definen la existencia de una secuencia de acciones lineales como la base de construcción de dos estructuras de control complementarias: los ciclos o iteraciones y las condicionales o alternativas. Tómese como ejemplo una acción simple, como sería realizar la suma de dos números y obtener de ellos un resultado. Definir una secuencia de pasos ordenados que permita satisfacer dicha necesidad se logra mediante acciones atómicas o instrucciones, enmarcadas entre un inicio y un final. La **Ilustración 5** presenta una aproximación a dicha secuencia.

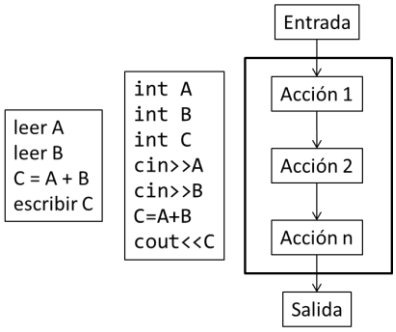


Ilustración 5. Representación de una secuencia de pasos lineal para la suma de dos números. (Fuente: el autor)

Y conforme lo menciona Sedgewick (2011), existen casos en los cuales las secuencias ordenadas son insuficientes para representar las acciones requeridas y dar solución a una problemática con

características especiales. Obsérvese por ejemplo la necesidad de saber si el número resultante de la operación anterior es par, en este caso se requiere de validar una igualdad o el cumplimiento de una determinada condición mediante una pregunta del tipo: *si<se cumple una condición> entonces [realice una acción]*, de allí el nombre de "estructuras condicionales", pudiendo alterar la ejecución de la secuencia de pasos y reorientarla, o simplemente crear una bifurcación para una acción opcional y permitir luego continuar con la secuencia normal de eventos.

Realizando un barrido por los textos escritos por Sedgewick (2011), Gayo (2013), Joyanes (2003), entre otros autores, es posible identificar tres tipos de estructuras condicionales: el *condicional simple*, el *condicional doble* y el *condicional anidado*, cada uno de ellos con características bien demarcadas y una utilidad específica.

El condicional simple verifica el cumplimiento de una y solo una condición específica, creando un apéndice en la ejecución y permitiendo posteriormente continuar con la secuencia normal de pasos a través de una composición del tipo: *si <se cumple una condición> entonces [realice una acción]* (p.ej., si el número resultante de la suma es múltiplo de cuatro, entonces multiplíquelo por dos). Esta acción solo se realizaría en la medida que la condición se cumpla a cabalidad, siendo a su vez ignorada en caso que no lo sea. El condicional doble por su parte, plantea dos posibilidades: el cumplimiento de la condición o la posibilidad de presentarse su caso contrario a través de una composición del tipo: *si <se cumple una condición> entonces [realice una acción] sino se cumple [realice esta otra acción]* (p.ej., si el número resultante de la suma es par, entonces divídalo por dos, sino divídalo por tres). Obsérvese que la verificación contempla solo una de las dos posibilidades, y su no cumplimiento conlleva inmediatamente a que la otra sea tomada como cierta, lo cual es útil siempre que no existan más de dos posibles estados que puedan ser evaluados como ciertos y requieran ser tratados de manera independiente. Ahora bien, en el caso anterior el resultado de la suma puede ser un cero, así que el cero no es par, no es impar, no es positivo ni es negativo, como consecuencia una condicional doble sería insuficiente para representar su identificación al interior del algoritmo. Es en este punto donde la condicional anidada presenta su valía a través de una estructura del tipo: *si <se cumple una condición> entonces [realice una acción] sino si <se cumple otra condición> entonces [realice otra acción] sino se cumple ninguna de las anteriores [realice esta otra acción]*, si se observan las posibilidades arrojadas por el hecho que se valide que el número resultante de la suma sea cero; si no lo es, se valida que sea par, y si finalmente no es ninguno de los dos anteriores, puede decirse entonces que se trata de un número impar. La **Ilustración 6** permite diferenciar las características fundamentales de las 3 secuencias de control principales.

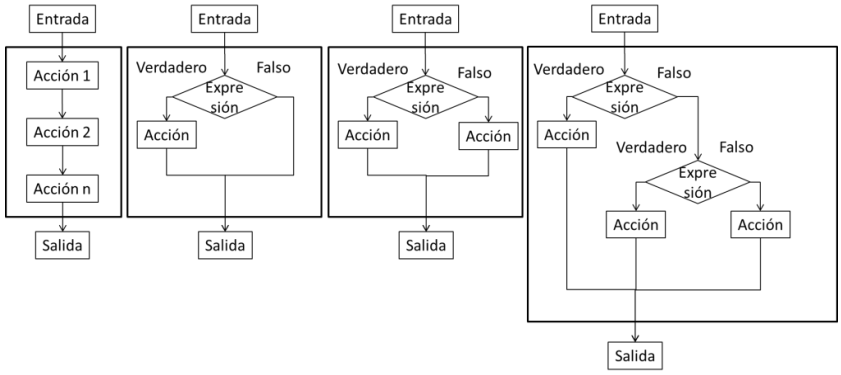


Ilustración 6. Representación comparativa de los condicionales simple, doble y anidado frente a la secuencia ordenada de pasos. (Fuente: el autor)

De la igual manera debe analizarse aquellas secuencias de pasos que al ejecutarse una sola vez no permiten obtener el resultado esperado (p.ej., acumular el valor 1 tantas veces como sea necesario hasta llegar a 10) y surge la necesidad de repetir dichas secuencias un número determinado de veces, cuya cantidad puede variar entre ejecución y ejecución, a través de ciclos o iteraciones. En este caso es posible identificar tres tipos de ciclos: el ciclo *para*, el ciclo *mientras*, y el ciclo *haga mientras*.

El ciclo *para* es empleado cuando es requerido ejecutar una instrucción o conjunto de ellas por un número determinado de repeticiones mediante una estructura del tipo *para <una condición de inicio; hasta una condición de terminación; de paso en paso> [ejecute un conjunto de instrucciones]*. La condición específica a evaluar en este punto es realmente el número de repeticiones del conjunto de instrucciones y no la verificación de cualidades que puedan verse modificadas durante la ejecución de las instrucciones. En el caso que la verificación de las condiciones sea lo necesario para dar continuidad a las iteraciones, los otros dos casos son los ideales.

En este punto surge la inquietud ¿Cuándo hacer uso de uno o el otro? Sedgewick (2011) explica a través de ejemplos que la iteración basada en una estructura del tipo *mientras <se cumpla una condición> haga [la ejecución de un conjunto de instrucciones]* implica necesariamente dos cosas: primero, que la condición a validar debe verse cumplida antes de entrar al ciclo y ejecutar sus instrucciones, ya que en caso contrario su conjunto de instrucciones interno será ignorado; y segundo, al interior del conjunto de instrucciones debe presentarse un cambio de estado en las condiciones a validar, de forma tal que el ciclo pueda finalizar en algún momento del tiempo o luego de un número finito de repeticiones, condición fundamental para la construcción del algoritmo como solución. Por otra parte, las iteraciones basadas en una estructura del tipo *haga [la ejecución de un conjunto de instrucciones] mientras <se cumpla una condición>* implica que el conjunto de instrucciones son ejecutadas al menos por una vez antes de validar si la condición a validar es cierta o no. La **Ilustración 7** permite evidenciar las características principales de los ciclos *mientras* y *haga mientras*.

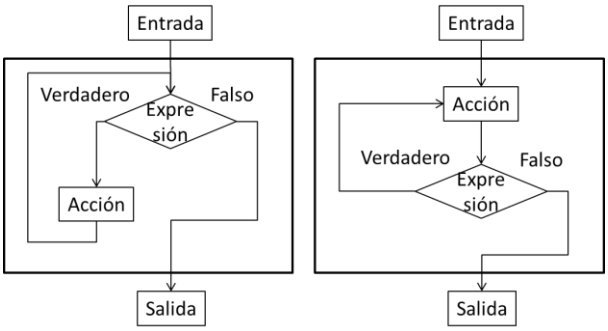


Ilustración 7. Representación comparativa de la composición para los ciclos MIENTRAS y HAGA MIENTRAS. (Fuente: El autor)

1.3.2.2. Representación de los Algoritmos

Desarrollando la idea que los algoritmos constituyen la posibilidad de realizar acciones guiadas para la resolución de problemas ligados al tratamiento de la información y que estas acciones pueden ser ejecutadas de manera secuencial empleando ramificaciones o ciclos para el logro de sus objetivos, se hace indispensable tener en cuenta la necesidad de plasmarlo de una manera simple, y entendible para cualquier persona que requiera de dicha solución. En este sentido, la manera en que se representa dicho algoritmo puede variar según el contexto en el que se desenvuelva quien acceda a su colección de pasos, bien sea con la finalidad de aplicarlo o de validar sus resultados en diversas colecciones de datos de entrada, siempre con la premisa que sus características o la problemática que le sustenta debe conservar su validez a lo largo del tiempo.

Una serie de representaciones relacionadas son enumeradas por Gayo (2013) y a su vez compartidas en su mayoría por otros autores, como es el caso de Rodríguez (2008), Toledo (2002) y Joyanes (2003), a saber: el Lenguaje Natural, los Diagramas de Flujo, el Pseudocódigo y los Lenguajes de Programación.

El Lenguaje Natural (LN) sería la primera opción cuando se habla de realizar una descripción del conjunto de pasos necesarios para el logro de un objetivo discreto. Sin embargo, no necesariamente corresponde a la elección más sencilla, ya que aspectos como: divergencia en el significado de las palabras, el uso de expresiones, modismos o regionalismos, la vaguedad o ambigüedad en las descripciones y la dificultad para la expresión escrita, pueden dificultar su construcción y uso ampliado. La **Ilustración 8** muestra la forma en que el lenguaje natural puede ser empleado para expresar un conjunto de pasos entendibles en el cálculo del área de un rectángulo.

Para calcular el área de un rectángulo es necesario solicitar el valor de la base y luego solicitar el valor de la altura, verificando que ambos sean positivos se emplea posteriormente la fórmula donde el área es equivalente a la multiplicación de la base por la altura. Dicho resultado se asigna a otra variable antes de ser presentada al usuario.

Ilustración 8. Representación de una descripción en lenguaje natural para el algoritmo que permite calcular el área de un rectángulo. (Fuente: el autor)

Los Diagramas de Flujo (DF) corresponden a representaciones gráficas de acciones pre-establecidas, las cuales son situadas de manera secuencial con el fin de proveer transformaciones que permitan la obtención de resultados, dicha representación gráfica reduce algunas de las problemáticas presentadas por el lenguaje natural, como es el caso de la ambigüedad, el uso de expresiones no universales y la dificultad para la expresión escrita. Sin embargo, existen dificultades frente a la representación de estructuras de transformación complejas, llevando a que la construcción de algoritmos empleando lenguaje gráfico sea más perjudicial que benéfica si no se conocen las representaciones o convenciones para cada una de las acciones a realizar, sin contar que hasta este punto, tanto el lenguaje natural como el visual no son traducibles directamente a construcciones que permitan implantarse en sistemas informáticos sin requerir grandes cantidades de esfuerzo en su transformación. La **Ilustración 9** muestra una estructura algorítmica construida a partir de representaciones gráficas para el ejemplo del cálculo de área de un rectángulo, presentado con anterioridad.

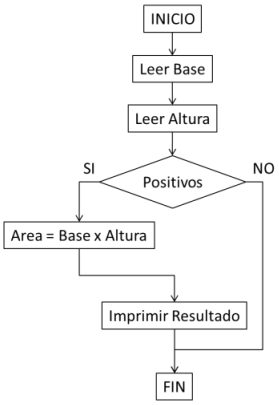


Ilustración 9. Representación de una descripción en Diagrama de Flujo para el algoritmo que permite calcular el área de un rectángulo. (Fuente: el autor)

El Pseudocódigo (PSC) por su parte, pretende integrar los aspectos de sencillez del lenguaje natural con la estructura y el orden de los diagramas de flujo, obviando del primero la libertad de "definir" el conjunto de instrucciones y limitándolo a un conjunto reducido de ellas, mientras resta del segundo el uso de representaciones gráficas que obligan a la construcción de mapas y esquemas de difícil comprensión, compactándolo y delimitando claramente el campo de acción

del algoritmo, su estructura y acciones. En esta medida se facilita su traducción a un lenguaje de programación, ya que por lo general, el conjunto de acciones tienen una correspondencia representativa con las instrucciones nativas de control de la mayoría de los lenguajes de programación. La **Ilustración 10** alecciona un ejemplo simple de este concepto sobre el cálculo del área de un rectángulo, desarrollado en los ejemplos anteriores.

```
INICIO
    definir Base, Altura, Area como enteros
    leer base
    leer altura
    si base y altura son positivos entonces
        area ← base x altura
    imprima area
FIN
```

Ilustración 10. Representación de una descripción en Pseudocódigo para el algoritmo que permite calcular el área de un rectángulo. (Fuente: el autor)

Finalmente se ilustra el caso de la representación del algoritmo a través del uso de instrucciones pertenecientes a un lenguaje de programación. Dicha representación posee las ventajas de estructura inherentes al pseudocódigo, con el adicional que no solo es comprensible por las personas, sino por las computadoras, pudiendo llevarse a un proceso de ejecución directa. Sin embargo, dicha representación es propia del lenguaje, y en algunos casos puede diferir de la estructura que se esperaría obtener partiendo del pseudocódigo, conforme a las necesidades propias del lenguaje para la preparación de la ejecución. Obsérvese el ejemplo presentado en la **Ilustración 11** y compárese con su equivalente en pseudocódigo.

```
int main(void){
    int base = 0;
    int altura = 0;
    int area = 0;
    cin>>base;
    cin>altura;
    if ((base>0)&&(altura>0))
        area = base * altura;
    cout<<area;
}
```

Ilustración 11. Representación de una descripción en código C++ para el algoritmo que permite calcular el área de un rectángulo. (Fuente: el autor)

1.3.2.3. Verificación de Resultados: las Pruebas de Algoritmos

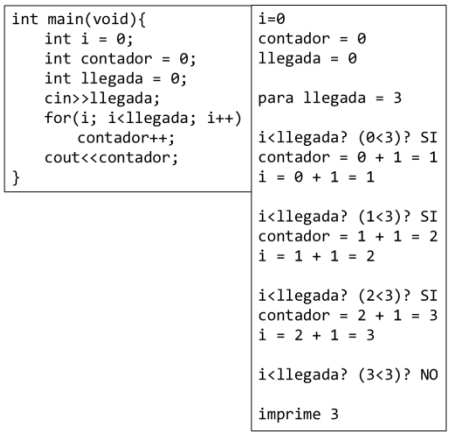
Hasta este punto se ha tomado como premisa la existencia de una definición del problema, de la cual se identifican sus aspectos más relevantes, facilitando la estructuración de un conjunto de pasos que conllevarían a una solución adecuada y su representación algorítmica. El algoritmo es el citado resultado de este proceso de construcción, y conforme lo describe Harel (2004) en los algoritmos se presenta un comportamiento constante en la medida que sus entradas de datos son constantes y legales, es decir, que mientras no exista variación en las características de las

entradas asignadas a los problemas algorítmicos, y cumplan con las características esperadas para alimentar el algoritmo, sus resultados son invariantes independientemente del número de ocasiones en que sean ejecutados. Puede entones decirse que un problema ha sido resuelto en la medida que se ha construido el algoritmo que brinde una solución genérica y adecuada al problema durante cada ejecución del proceso, bajo la condición que se brinden entradas pertenecientes a un conjunto considerado permitido o legal.

Sin embargo, el comportamiento resultante del algoritmo puede no ser el esperado, bien sea por que la definición del problema sea ambigua o errónea, porque la estructuración de sus pasos no brinda una solución adecuada al problema, porque las estructuras de control que le conforman tienen algún tipo de desvío en su secuencia normal o por que los datos que le han sido ingresados pueden considerarse incorrectos o ilegales. De cualquier modo, puede entenderse que es interés y responsabilidad del problema algorítmico brindar tratamiento a cada una de estas vicisitudes con el fin de conservar su estructura y evitar resultados no contemplados o incorrectos, de acuerdo a lo manifestado por autores como Bond (2007) y Meinke (2004), entre otros.

Pero en este nivel surgen inquietudes diversas, por ejemplo: ¿Cómo es posible que un algoritmo no haga correctamente aquello para lo cual ha sido construido? Y si el algoritmo constituye una construcción mental orientada a la solución de un problema ¿Cómo esperar que tenga un comportamiento diferente al definido en su construcción y que sus resultados no sean los esperados? Si bien los algoritmos reflejan la manera en que su creador identifica las características de un problema, es él quien construye alrededor de la solución una estructura mental que le soporte, pero en pocas ocasiones, dicha estructura mental toma en consideración las diferentes variables que pueden ocasionar comportamientos no controlados o arrojar resultados no válidos. Podría decirse que la estructuración del algoritmo se orienta a que haga lo que tiene que hacer, pero en ningún momento a validar que no haga lo que no tiene que hacer. Es en este sentido que la verificación de un algoritmo debe ser concebida como un proceso a realizarse en diferentes etapas, permitiendo generar la trazabilidad de la solución desde el momento de su concepción hasta llegar a su aplicación en la vida diaria mediante su implementación en un lenguaje de programación.

La primera etapa en la verificación de un algoritmo corresponde a su “seguimiento mental” en el momento mismo de su concepción, haciendo un recorrido a través de su colección de pasos e identificando posibles inconsistencias en su estructura, de manera que permita evidenciar su idoneidad en la solución al problema planteado. En la **Ilustración 12** puede observase que este tipo de verificación, si bien es simple en su aplicación, se hace compleja en la medida que el algoritmo se hace más denso, ya que el manejo de diversas características y sus transformaciones pueden estar más allá de la capacidad mental de quien lo construye. Así, el incremento en la complejidad limita el seguimiento mental, surgiendo la verificación a través de un registro escrito o “prueba de escritorio”.

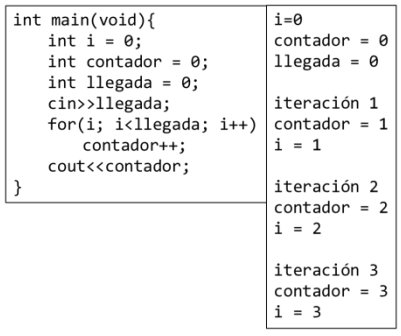


```
int main(void){
    int i = 0;
    int contador = 0;
    int llegada = 0;
    cin>>llegada;
    for(i; i<llegada; i++)
        contador++;
    cout<<contador;
}
```

```
i=0
contador = 0
llegada = 0

para llegada = 3

i<llegada? (0<3)? SI
contador = 0 + 1 = 1
i = 0 + 1 = 1

i<llegada? (1<3)? SI
contador = 1 + 1 = 2
i = 1 + 1 = 2

i<llegada? (2<3)? SI
contador = 2 + 1 = 3
i = 2 + 1 = 3

i<llegada? (3<3)? NO

imprime 3
```

Ilustración 12. Simplificación del proceso de seguimiento mental de un algoritmo. (Fuente: el autor)

La prueba de escritorio permite no solo observar el comportamiento del algoritmo paso a paso, sino que deja tras de sí un rastro físico de dicho comportamiento. Posee evidentes ventajas frente a la verificación mental, ya que no depende completamente de la memoria para el almacenamiento de los datos, facilitando en algunos casos devolverse sobre los pasos ya realizados, o inclusive partir de un resultado esperado y construir la serie de pasos necesarios para obtenerlo, algo que según Brandt (2002) puede entenderse como Backtracking. Este nivel de verificación requiere un mayor nivel de concentración en el seguimiento de los pasos ejecutados, dejando de lado la memorización del estado de las variables que son manipuladas por el algoritmo. Pero muy a pesar de sus ventajas, puede ser bastante monótono cuando se habla de algoritmos que poseen estructuras demasiado complejas o con un número alto de iteraciones. Tómese como ejemplo el contador presentado en la **Ilustración 13**, ¿Qué pasaría si en lugar de contar hasta tres lo hiciera hasta 100 o hasta 1’000.000? este es el motivo por el cual tiende a emplearse conjuntos pequeños de datos. El éxito de la prueba de escritorio consiste en registrar de manera detallada el comportamiento de las variables del algoritmo en cada paso.



```
int main(void){
    int i = 0;
    int contador = 0;
    int llegada = 0;
    cin>>llegada;
    for(i; i<llegada; i++)
        contador++;
    cout<<contador;
}
```

```
i=0
contador = 0
llegada = 0

iteración 1
contador = 1
i = 1

iteración 2
contador = 2
i = 2

iteración 3
contador = 3
i = 3
```

Ilustración 13. Simplificación del proceso de seguimiento de un algoritmo empleando prueba de escritorio. (Fuente: el autor)

Posterior a su construcción, la verificación del algoritmo corre por cuenta de su ejecución en un ambiente controlado. Dicha ejecución debe ser observada detalladamente con el fin de corroborar

que los resultados obtenidos sean correspondientes con lo esperado. Este tipo de verificación se conoce como pruebas funcionales o testing.

Las pruebas funcionales, conforme son explicadas en los escritos de Patton (2001), Sommerville (2005) y Pressman (2005) se encaminan a la búsqueda de errores en la estructura funcional de un algoritmo, permitiendo identificar los puntos en los cuales ocurren y el impacto que pueden tener en su ejecución y los resultados obtenidos. En sus escritos es posible identificar al menos cuatro aproximaciones al proceso de ejecución de las pruebas: dos en consideración de la posibilidad de acceder al código (las pruebas de caja negra y las pruebas de caja blanca), y otras dos orientadas a la posibilidad de acceder a estructuras descriptivas o funcionales del software (las pruebas estáticas y las pruebas dinámicas). Ambos enfoques pueden ser complementarios en su aplicación, de forma que se logre una revisión más a profundidad de las capacidades del conjunto de algoritmos que soporta al software, partiendo de la premisa que cualquier algoritmo podrá ser probado de manera tan amplia como se desee, sin que por ello se considere que su verificación es exhaustiva.

En el caso de las pruebas estáticas, podría decirse que están más relacionadas con las etapas tempranas de definición del algoritmo que con la ejecución del mismo, ya que se orientan en mayor grado a observar que la especificación del problema sea consistente y que la estructuración de su solución pueda en sí, satisfacer las necesidades identificadas. Las pruebas dinámicas por su parte requieren de unidades algorítmicas que puedan ser ejecutadas o al menos verificables en su comportamiento a través de un seguimiento (p.ej., con una prueba de escritorio o unas pruebas funcionales).

En términos de ejecución de soluciones algorítmicas, las pruebas de caja negra solo permiten validar el comportamiento del algoritmo a partir de la relación entrada/salida, ignorando completamente el proceso de transformación intermedio y fundamentando la comparativa de éxito o fracaso simplemente en la definición de lo que el software está destinado a hacer. Por otra parte, en las pruebas de caja blanca es posible acceder al algoritmo que soporta la ejecución, facilitando la definición de estrategias de verificación en su estructura (p.ej., frente al uso de tipos de datos) conforme a la relación entrada/proceso/salida, haciendo evidente el proceso de transformación intermedio. Se esperaría entonces que las pruebas de caja blanca sean más efectivas que las pruebas de caja negra, sin embargo demandan de un mayor cuidado ya que es posible perder la objetividad durante su ejecución, ya que en ocasiones las pruebas pueden realizarse de una forma demasiado ajustada a la estructura de codificación, enfocándose simplemente en observar que el código haga lo que se supone debe hacer.

1.3.3. Relación entre Algoritmos y Programación

Mucho se ha mencionado acerca de la importancia de los algoritmos y la manera en que estos brindan estructura a la construcción de soluciones desligadas de la sintaxis y las restricciones de un lenguaje de programación. Es así que Shen (2010) presenta un enfoque de algoritmia

considerado independiente de la existencia de computadores capaces de ejecutar instrucciones, y se orienta a la definición de estrategias y modelos aplicables al entendimiento de la naturaleza del problema y su solución genérica, algo que en términos computacionales es poco práctico en la medida que las máquinas no comprenden las representaciones algorítmicas basadas en lenguaje natural, diagramas de flujo e inclusive pseudocódigo. Sin embargo, dicha independencia representa una gran ventaja estratégica en la medida que se cimenta la estructura lógica en el desarrollo de la solución, permitiendo entender de primera mano y a fondo las características del problema, validando su secuencia de pasos antes de invertir tiempo en la codificación de una solución no probada o que no satisface sus necesidades totalmente.

Tómese como punto de partida la existencia de un problema algorítmico específico como el presentado en la **Ilustración 14**, sobre el cual es construida una solución algorítmica genérica en cualquiera de sus posibles representaciones (p.ej., pseudocódigo) y que ha sido validada en un primer punto a través del seguimiento mental y el registro de una prueba de escritorio.

```
INICIO
    lea numero
    para (i=0; i<numero; incremente i en 1)
        lea valor
        vector(i) ← valor
    fin para
FIN
```

Ilustración 14. Representación en pseudocódigo para el llenado de un vector de números. (Fuente: el autor)

Para que dicha estructura algorítmica sea empleada en la construcción de una solución software es necesario traducirla a un lenguaje de programación de alto nivel mediante un proceso conocido como programación. En este sentido, la descripción de programación presentada por Joyanes (2001) difiere de un conjunto de acciones triviales de sustitución de palabras, ya que debe considerarse – entre otras cosas – aspectos como estructuras gramaticales y de sintaxis, menos flexibles que las empleadas por las representaciones tempranas del algoritmo. Obsérvese la comparación presentada por la **Ilustración 15**, donde las estructuras de representación de un algoritmo en pseudocódigo y en un lenguaje de programación de alto nivel como C++ hacen evidentes sus diferencias.

```
INICIO
    lea numero
    para (i=0; i<numero; incremente i en 1)
        lea valor
        vector(i) ← valor
    fin para
FIN
```

```
int main(void){
    int numero = 0;
    Int vector[10];
    cin >> numero;
    for (int i = 0; i < numero; i++){
        cin >> vec tor[i];
    }
}
```

Ilustración 15. Representación en código para el llenado de un vector de números y comparación con su pseudocódigo. (Fuente: el autor)

En este respecto, el proceso de programación debe partir de la identificación y caracterización de las entradas que alimentarán y las salidas que entregará el algoritmo, ya que de esta manera es posible establecer las necesidades de declaración de variables y constantes en función de su tipo (p.ej., públicas, privadas, globales o locales) conformando así el conjunto de datos válidos del algoritmo. Es necesario ser muy cuidadoso al seleccionar los tipos de dato de las variables de entrada y su correspondencia con las salidas, ya que como puede derivarse del análisis del trabajo presentado por Bond (2007) la incompatibilidad en el manejo de tipos de datos dispares puede llevar a que el algoritmo presente fallos en su implementación aunque su lógica sea correcta bajo cualquier punto de vista. Un ejemplo de ello puede observarse en la **Ilustración 16**.

```
float dividir(int a, int b){
    return a / b;
}
```

Ilustración 16. Representación del manejo dispar de tipos de dato, obsérvese que la función retorna un flotante mientras que opera con enteros. (Fuente: el autor)

El siguiente paso en el proceso de programación del algoritmo es tomar acciones atómicas de manipulación de los datos, como es el caso de las asignaciones (p.ej., por valor o por referencia), verificación de desigualdades (p.ej., magnitudes o proporciones), verificación de expresiones y operaciones (p.ej., prioridad en los operadores y sobrecarga de los mismos), de forma tal que el proceso de traducción conserve el sentido original sobre el cual fue construido el algoritmo.

Finalmente, la programación del algoritmo se centra en la sustitución de las instrucciones evidenciadas en el método de representación por aquellas que hacen parte de la estructura propia del lenguaje seleccionado para la construcción del programa. Estas instrucciones se conocen generalmente como “palabras reservadas” y requieren de un conocimiento amplio del lenguaje de programación y su sintaxis.

1.3.4. Competencias Algorítmicas

Es evidente el hecho que, la construcción de estructuras algorítmicas, su verificación, y posterior traducción a un lenguaje de programación son actividades que demandan de quienes les construyen un conjunto de conocimientos y habilidades aplicados a la solución de problemáticas específicas. Estas habilidades son desarrolladas y solidificadas a través de un proceso continuo de aplicación del conocimiento y se conocen como competencias. Pero ¿Qué son? ¿Cuáles pueden considerarse competencias propias del proceso de solución de problemas algorítmicos?

1.3.4.1. Definición de Competencias

Las competencias son descritas por Perrenoud (2000) como la capacidad para hacerse de un conjunto de conocimientos con el fin de aplicarles al hacer frente a situaciones problemáticas de la vida real, pudiendo inferirse de esto que alguien competente es aquel que tiene autonomía y capacidad para aprender a resolver problemas, logrando un crecimiento en la construcción de nuevo conocimiento aplicado, y no simplemente una acumulación de teorías sin orden o

estructura. En este respecto: Churches (2009), Andeson (2000), Füller (2007) y Stucki (2000) presentan una clasificación de los objetivos del proceso de aprendizaje que tuvo su origen en los planteamientos de Benjamin Bloom y se conoce como la *Taxonomía de Bloom*. Su idea fundamental se orienta a entender el hecho que todo estudiante debe adquirir no solo conocimientos teóricos, sino que debe estar en capacidad de evidenciar su aplicación mediante el desarrollo de conocimientos y habilidades prácticas. Es importante tener en cuenta que dicha taxonomía no define una secuencia de instrucciones a seguir, ni la manera de llevarlas a cabo, pero si establece los niveles esperados de desempeño de los estudiantes, asumiendo que en la medida que se alcanza los niveles más altos, aquellos niveles más bajo han sido apropiados y pueden ser aplicados sin problema alguno, logrando una estructura acumulativa e incremental de conocimiento.

Así, el desarrollo de las competencias se evidencia como la apropiación evolutiva del conocimiento a través de la formación de destrezas y su aplicación, orientada a la solución de problemáticas de áreas específicas. Se toma como foco principal el desarrollo de perspectivas del universo que rodea al estudiante, permitiéndole abstraer sus características más relevantes. De la misma forma, su valoración debe orientarse a la demostración del desempeño del estudiante en su actividad de pericia. La **Ilustración 17** presenta visualmente la manera en que el proceso de apropiación de un conocimiento particular logra ser aplicado en la solución de problemas cotidianos, lo cual hace que se convierta en parte del comportamiento natural del individuo, aconduchándole.

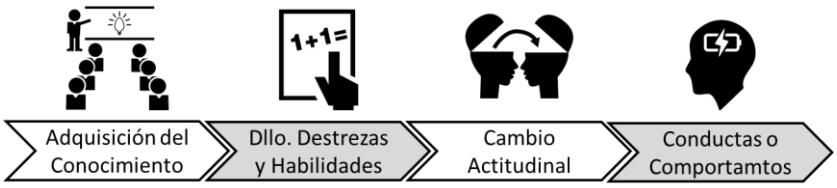


Ilustración 17. Estructura de adquisición de las competencias en un proceso evolutivo. (Fuente: El Autor)

1.3.4.2. Competencias de cara a la Algoritmia

Tomando la definición de competencias planteada con anterioridad, es posible observar que la solución de problemas a partir del análisis de su información relacionada, la abstracción de sus características y la aplicación práctica del conocimiento para la toma de decisiones, es una de las principales metas en el desarrollo de las habilidades que requieren los profesionales de las nuevas generaciones.

Pero ¿de qué manera es posible fortalecer en el estudiante las habilidades orientadas a la solución de problemas? es posible encontrar pistas en lo expresado por Zsakó (2012), quien expresa que el desarrollo de competencias en el uso de Tecnologías de Información y Comunicación debe orientarse a la aplicación de herramientas y métodos, tanto en el aprendizaje como en la vida diaria, incluyendo no solo un enfoque de usuario en el uso de las herramientas

ofimáticas, sino en la comprensión de la estructura de origen, procesamiento y almacenamiento de información.

Es así que en su documento plantea diez componentes principales en el desarrollo de las competencias, de las cuales se extractarán las siguientes:

1. Pensamiento algorítmico, como la posibilidad de diseñar algoritmos a partir de series de actividades y construcción de flujos de información en lo cotidiano.
2. Modelamiento de datos, permitiendo la descripción de los elementos de la realidad a partir de sus características más relevantes.
3. Modelamiento del mundo real, como un medio para entender la composición e interacción de los elementos que participan de un fenómeno en particular.
4. Solución de problemas, como parte de un proceso creativo que, fundamentado en los planteamientos de Polya (2004), permita descomponer el problema en sus partes principales y analizarles de manera sinérgica.
5. Pensamiento sistémico, de forma que le sea posible entender la interrelación de las partes de un problema y la manera en que afectar una de ellas pueda verse reflejado en su equilibrio.

Los cuatro elementos restantes de este listado hacen parte del desarrollo de competencias de corte transversal, donde la capacidad de comunicación, el desarrollo de la creatividad y el trabajo en equipo, son los principales protagonistas.

1.3.4.3. Niveles en el desarrollo de las Competencias Algorítmicas

El proceso de "*algoritmización*" no es en ningún momento dependiente de una implementación computacional, y por el contrario, busca que su construcción sea franqueada por la rutina y la cotidianidad, de manera que las secuencias de pasos que los componen puedan ser entendibles por sí mismos y por los demás, transformándose en una actividad natural, analítica de las acciones, y previa a un proceso de estructuración en un lenguaje de programación.

Es así que se partirá de la propuesta descrita por Zsakó (2012), quien estipula siete niveles de desarrollo de la capacidad algorítmica, a saber:

1. Reconocimiento y entendimiento de los algoritmos como secuencias de actividades que pueden ser utilizados para resolver problemas con características claramente identificadas, diferenciando aquello que debe ser realizado y las diferentes posibilidades para lograr los resultados esperados.
2. La implementación de algoritmos como conjuntos de instrucciones, descritas de manera clara para que puedan ser comprendidas sin error o ambigüedad.
3. El análisis de los algoritmos como una secuencia de acciones construidos a partir de reglas básicas de sintaxis y gramática, unificando estructuras de pensamiento y orientándole a una estandarización en su construcción.

4. La construcción de algoritmos como secuencias de pasos que pueden ser adaptables a nuevas necesidades, extrapolando sus características esenciales con las de otros problemas de corte similar.
5. La descripción de los algoritmos como estructuras de pasos pertenecientes a un lenguaje de programación.
6. Modificación y cambio a los algoritmos planteados, como secuencias de acciones estructuradas con un fin particular, pero que deben ser actualizadas a las necesidades cambiantes del medio en que se desenvuelven.
7. El diseño de algoritmos complejos, como secuencias de acciones que permiten resolver partes más reducidas de un problema, y que por disgregación del mismo, requieren además de secuencias de acciones que faciliten la intercomunicación de las partes solucionadas.

Obsérvese que al final es posible establecer que las competencias relacionadas con la algoritmia se reducen en cuatro fases: la identificación de las secuencias de pasos, el uso de dichas secuencias en la construcción de conjuntos de instrucciones, extrapolación y comparación de las características del problema con otros similares, facilitando la reutilización, y finalmente la modificación y adaptación del algoritmo.

1.3.5. Pensamiento Computacional

Hasta ahora se ha visto que el desarrollo de las competencias algorítmicas requiere de diversos niveles de abstracción, partiendo desde la necesidad de conocer los fundamentos teóricos que rodean a los algoritmos hasta llegar a su aplicación en diferentes contextos a lo largo del tiempo. Durante el tránsito de estos eventos, es el análisis del problema y su entendimiento lo que permitirá el desarrollo de una solución adecuada, y la manera en que se aborde, será el instrumento que evidenciará claramente el logro de las competencias.

Pensar de forma algorítmica es algo natural, todas y cada una de las actividades que son realizadas cotidianamente tienen un corte algorítmico ya que pueden dividirse en secuencias de pasos con un orden particular, partiendo desde un estado inicial y permitiendo lograr un objetivo discreto al culminarse. Puede tomarse como ejemplos la preparación requerida para ir a trabajar, tomar el desayuno, desplazarse hacia el trabajo, realizar actividades académicas, entre otras. Sin embargo, el hecho que estas acciones hagan parte del día a día torna su realización en algo monótono, rutinario y automático, desdibujando aspectos como el orden y la secuencialidad. Aun así, tomar como base la premisa que toda actividad puede ser expresada en términos de pasos ordenados posibilita inferir de igual forma que cualquier problema puede ser abordado y analizado en forma algorítmica. Dicha afirmación constituye en sí la cimentación del concepto de Pensamiento Computacional.

El Pensamiento Computacional es una teoría propuesta en su forma actual por Wing (2006) quien lo describe como una habilidad básica, equiparable con leer, escribir y realizar cálculos

matemáticos, planteando la posibilidad de afrontar los problemas cotidianos por medio de la abstracción de sus características más relevantes y considerando además del conjunto de pasos necesarios para lograr una solución adecuada, las restricciones dadas por el ambiente y su nivel de complejidad. Como resultado, toda solución se obtendría a partir de un análisis del problema haciendo uso de la premisa "divide y vencerás", ampliamente aceptada en las ciencias computacionales como la base para abordar de manera recursiva cualquier solución.

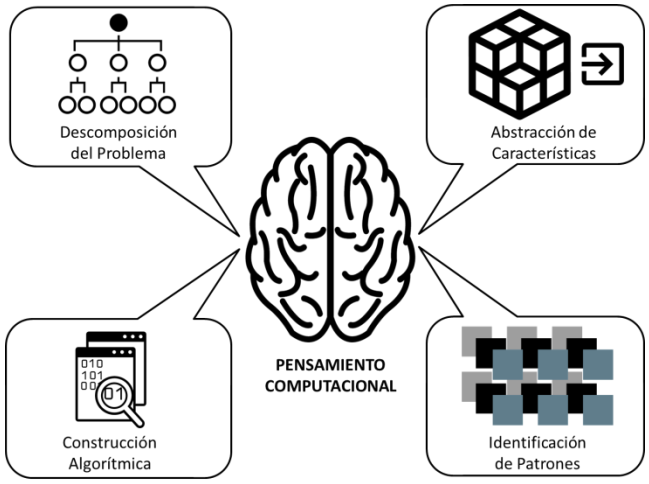


Ilustración 18. Aspectos clave relacionados con el Pensamiento Computacional. (Fuente: el autor)

La **Ilustración 18** muestra que la descomposición de un problema en fragmentos más pequeños, la abstracción de sus características y la identificación de patrones son tareas propias del análisis algorítmico ligado al Pensamiento Computacional. Al identificar los aspectos fundamentales del problema, es posible encontrar una representación de la solución que les satisfaga de la mejor manera posible, así el pensamiento computacional se orienta a hacer que el enfoque de análisis propio de las ciencias computacionales haga parte de la cotidianidad, facilitando el enfrentarse a problemáticas que no necesariamente involucren tecnologías de información en su solución, pero que requieran de la definición de secuencias algorítmicas de pasos para llegar a ella. De esta manera, la abstracción se torna el punto central al realizar el análisis del problema, permitiendo identificar escenarios diversos, con características similares o intereses transversales, en los cuales la solución pueda ser aplicada de manera general.

1.3.5.1. Origen del Pensamiento Computacional

Puede identificarse un primer acercamiento al concepto de Pensamiento Computacional a través de la Teoría Computacional de la Mente (TCM), propuesta por Hilary Putnam en los años 60. Conforme se describe en el trabajo realizado por Alvarado (2000) dicha teoría toma el funcionamiento del cerebro y la mente humana de manera similar a la de un computador, siendo su fundamentación el hecho que el cerebro esté en capacidad de procesar un conjunto de entradas o estímulos empleando el pensamiento como esquema de análisis y estructuración de la información.

Este enfoque realiza una clara descripción de la manera en que funciona la mente a través de la conexión de las ideas con las acciones y la obtención de un significado más amplio para el concepto del pensamiento y la percepción como fuente información, sin embargo el cerebro funcionaría más como repositorio de información y fuente de su procesamiento a través de reglas sintácticas establecidas, y no como el punto central donde se desarrolla el análisis de sus variables y la construcción de los algoritmos que le permiten realizar labores específicas.

Entonces, la interpretación es en sí una de las grandes problemáticas inherentes al análisis de los estímulos del medio que alimentan al cerebro, siendo fundamental contar con habilidades más allá de lo interpretativo. La construcción de estructuras de control a través del uso de reglas sintácticas debe provenir entonces del análisis situacional, conforme lo presenta Simari (2013). A pesar de lo anterior surge la inquietud ¿de dónde proviene la construcción de dichas estructuras de control? Si se analiza de manera estricta la Teoría Computacional de la Mente (sin adentrarse demasiado en el contexto cognitivo y su discusión) no es el computador quien genera sus propias instrucciones, por el contrario, interpreta de manera inequívoca aquellas que le son suministradas y sin mayor reparo. En este punto puede hacerse un paréntesis para lograr entender la explicación dada por Ala-Mutka (2004) acerca del como un modelo educativo tradicional puede ser empleado en las teorías de la ciencia computacional para formar estudiantes en capacidad de repetir estructuras de control y secuencias aprendidas, pero sin advertir con claridad la manera de aplicarlas a situaciones problémicas cambiantes. Surge una inquietud adicional, relacionada con identificar ¿hasta qué punto es desarrollada en el estudiante la capacidad de analizar e interpretar de las necesidades que obligan a su uso? El conjunto de instrucciones necesarias para la realización de una actividad surge entonces de un entendimiento de la realidad y la abstracción de sus características más relevantes.

Así, el entendimiento se deriva de concienciar al estudiante acerca de la importancia de pensar de manera lógica y coherente, no en términos de codificación o instrucciones ligadas a un lenguaje de programación, sino en la posibilidad de construir soluciones que sean factibles en el ámbito de las capacidades y limitaciones computacionales. En este punto sería sencillo pensar que la solución a las inquietudes planteadas se limita al desarrollo de la lógica y la capacidad de evidenciar las características fundamentales del problema con una orientación netamente computacional. Sin embargo se gesta una nueva inquietud acerca de ¿Qué pasaría cuando la problemática a abordar no se orienta a un esquema computacional tradicional? ¿Es posible pensar que el enfoque de trabajo que es útil en la realización de tareas computacionales pueda ser aplicado de igual manera a cualquier tipo de problema? En por ello que Wing (2006) equipara la formación en habilidades y métodos computacionales al nivel de los conocimientos fundamentales.

1.3.5.2. Principios del Pensamiento Computacional

Basándose en la definición de Pensamiento Computacional descrita en Wing (2013), su cimentación sería la formulación de problemas y sus respectivas soluciones a través de una

representación que pueda ser entendible por un agente de procesamiento de información, sea humano, máquina o una combinación de ambos.

Pero para lograr dicha representación sería necesario identificar los principios que rigen al pensamiento computacional, permitiendo así identificar las actividades requeridas para orientar el proceso de enseñanza-aprendizaje del estudiante. Cabe entonces plantear la inquietud acerca de ¿Cuáles son los principios del pensamiento computacional? ¿Cómo puede orientarse el trabajo a realizar con los estudiantes, si se pretende desarrollar en ellos la capacidad para afrontar diversas problemáticas a través de esta perspectiva? En este respecto Wing (2006) no es muy específica al describir los principios que soportan la estructura del pensamiento computacional, aunque su principal argumento seria el establecimiento de una necesidad transversal a cualquier disciplina: el desarrollo de la capacidad de abstracción y su relación con la descomposición del problema en partes más pequeñas pero igualmente representativas. Con este punto de partida es posible derivar de su disertación las siguientes capacidades a desarrollar en el estudiante:

1. Capacidad para abstraer las características más relevantes del problema, identificando intereses verticales y transversales.
2. Capacidad para dividir el problema en estructuras significativamente más pequeñas.
3. Capacidad para elegir la representación más adecuada para un problema y realizar trazabilidad de sus partes.
4. Capacidad para identificar no solo las variables del problema y su comportamiento, sino las invariantes que dan soporte a las transformaciones durante el proceso.
5. Capacidad para determinar un conjunto de posibles soluciones a un problema y establecer cuál de ellas sería la más adecuada en consideración de las características del problema y las heurísticas empleadas para su análisis.

Por otro lado, si se indaga en los escritos de Barr (2011) puede encontrarse otro conjunto de principios que sustentarán la idoneidad del pensamiento computacional como estructura cognitiva necesaria en disciplinas diferentes a las que conforman las ciencias computacionales en sí, a saber:

1. Capacidad para analizar datos y organizarlos de manera lógica.
2. Capacidad para modelar datos, construir abstracciones y realizar simulaciones.
3. Capacidad para formular solución a problemas que empleen asistencia por computadora.
4. Capacidad para identificar, implementar y probar posibles escenarios de solución.
5. Capacidad para automatizar soluciones a través del uso del pensamiento algorítmico.
6. Capacidad para generalizar y aplicar este mismo proceso a diversos problemas cotidianos.

Por su parte, Brennan (2012) menciona la inexistencia de una definición unificada de Pensamiento Computacional, permitiendo encontrar un abanico de posibilidades solventadas en la interpretación; así que podría esperarse igual disyuntiva con relación a la definición de sus características y principios. En su planteamiento es posible identificar tres dimensiones clave: conceptos computacionales, prácticas computacionales y perspectivas computacionales. La

primera se orienta a conocimientos específicos en estructuración de instrucciones y uso de operadores para la construcción de expresiones, mientras que la tercera permite expresar la manera en que se desarrollan habilidades transversales y de comunicación. La segunda dimensión – las prácticas computacionales – se orientan al desarrollo de habilidades necesarias para el análisis de los problemas y diseño de las soluciones requeridas, pudiendo interpretarse como principios en su concepción más general las siguientes:

1. Capacidad para diseñar y construir un proyecto de manera iterativa e incremental.
2. Capacidad para realizar un proceso de depuración a través de la verificación y validación de los programas.
3. Capacidad para identificar elementos pre-construidos y que puedan ser empleados para satisfacer necesidades actuales.
4. Capacidad de abstraer las características más relevantes del problema con el fin de conceptualizarlo.
5. Capacidad de dividir o modularizar el problema en segmentos más simples y manejables, permitiendo la definición de alcances intermedios o hitos en la construcción de la solución.

Finalmente, es importante considerar iniciativas que se han presentado con el fin de buscar el establecimiento de un marco común para el entendimiento del Pensamiento Computacional. Puede tomarse como referencia la definición operativa presentada por la Sociedad Internacional para la Tecnología en Educación y la Asociación de Docentes en Ciencias de la Computación (2012) en la cual se establece un conjunto de principios y características fundamentales para el desarrollo de la capacidad del estudiante, como serían:

1. Capacidad para formular problemas de forma tal que sea posible resolverlos haciendo uso de herramientas computacionales y no computacionales de apoyo.
2. Capacidad para organizar datos de manera lógica y realizar su análisis.
3. Capacidad para representar datos a través de abstracciones, como sería el caso de modelos y simulaciones.
4. Capacidad para la automatización de soluciones construidas mediante la definición de un conjunto de pasos ordenados a través del pensamiento algorítmico.
5. Capacidad para identificar, analizar e implementar diversas soluciones a una problemática específica, con el fin de seleccionar la combinación de pasos más eficiente y efectiva.
6. Capacidad de generalizar el proceso de solución construido, permitiéndole transferirlo a otros problemas mediante adaptación.

Este conjunto de principios conformaría entonces el punto de partida para el análisis a realizar en este trabajo, haciendo manifiesta la importancia de desarrollar en los estudiantes capacidades de abstracción, representación y evaluación de diversas soluciones a un mismo problema. Obsérvese que siempre se mantiene coherencia con lo planteado por WING en su disertación inicial.

Como resultado de este ejercicio será necesario realizar una actividad de unificación de los principios que orientaran el desarrollo de este proyecto, seleccionando en primera instancia aquellos que sean compartidos en mayor medida, brindándoles sustento mediante argumentos que permitan relacionarlos con las actividades propias del proceso algorítmico y de desarrollo de software. En un análisis preliminar es posible ver que la capacidad de abstracción sería aquella con mayor presencia en los diferentes enfoques, seguido de la posibilidad de división del problema en partes significativas, permitiendo finalmente enfrentarlos con construyendo diversos escenarios de solución. Sin embargo, Dicha selección y clasificación se profundizará en el Capítulo 7.

1.4. Planteamiento Metodológico

1.4.1. Metodología

Tomado como punto de partida el planteamiento de Tamayo (2004) es posible determinar que el presente trabajo se orientará bajo una estructura descriptiva, al interior de una forma aplicada.

En su desarrollo se espera realizar una exploración de elementos teóricos que permitan fundamentar y correlacionar aspectos generales del pensamiento computacional y de la enseñanza de algoritmos, por medio del análisis de la realidad actual en los procesos de enseñanza-aprendizaje y su solapamiento a la estructura de un marco de trabajo definido para tal fin.

1.4.2. Proceso de Investigación

El proceso de investigación estará orientado por las siguientes seis etapas:

1.4.2.1. Caracterización del estado actual en la enseñanza de los algoritmos

En esta etapa se pretende recopilar información literaria relacionada con la manera en que se llevan a cabo los procesos de enseñanza-aprendizaje ligados a la algoritmia. Como resultado se espera determinar no un conjunto significativo de prácticas que se llevan a cabo, sino el enfoque pedagógico que les sustenta y les convoca.

1.4.2.2. Identificación de los principios que cimentan el pensamiento computacional

Posterior a la identificación, recopilación y clasificación de un conjunto representativo de prácticas y metodologías pedagógicas orientadas al aprendizaje algorítmico, se hace necesario contrastar los planteamientos de diversos autores con relación a los que consideren, sean los principios claves del pensamiento computacional. Esta etapa busca seleccionar aquellos que conformarán el conjunto orientador del presente trabajo y como resultado se espera obtener un conjunto que esté alineado y sustentado con las prácticas actuales en desarrollo de software.

1.4.2.3. Establecer la aplicabilidad de los principios del pensamiento computacional en el proceso de enseñanza de algoritmos

Una vez seleccionados los principios del Pensamiento Computacional que orientarán el desarrollo del presente trabajo, es necesario determinar la aplicabilidad de dichos principios al proceso de enseñanza-aprendizaje, a la luz de las teorías y prácticas que, en un recorrido literario, sean consistentes con los planteamientos de diversos autores sobre algoritmia y desarrollo de software. Esta etapa busca relacionar las premisas seleccionadas con competencias fundamentales que el estudiante debe desarrollar en el campo algorítmico, obteniendo como resultado una justificación clara de la manera en que se articulan.

1.4.2.4. Definición de las premisas de aplicación de los principios de pensamiento computacional

Ya que se han seleccionado los principios orientadores del trabajo y se ha sustentado su aplicabilidad en un proceso de enseñanza-aprendizaje, es necesario definir el conjunto de competencias algorítmicas que serán desarrolladas y la manera en que las premisas del Pensamiento Computacional aportan a su logro. Esta etapa busca enumerar los diferentes niveles de acercamiento del estudiante al proceso de formación algorítmica, en consideración de las competencias que se espera lograr en él, obteniendo como resultado un conjunto de hitos evolutivos de las competencias algorítmicas con relación a la solución de problemas.

1.4.2.5. Plantear el Framework de aplicación de los principios del pensamiento computacional

Posteriormente se definen los conjuntos y secuencias de actividades que estructurarán el marco de trabajo propuesto para el desarrollo de las competencias algorítmicas en el proceso de enseñanza-aprendizaje. Esta etapa busca presentar la propuesta estructural del Framework, al igual que cada una de las actividades que lo conforman, sus elementos de entrada y resultantes. Al final se espera obtener una estructura de aplicación de las premisas del pensamiento computacional que sea independiente del nivel formativo del estudiante.

1.4.2.6. Definición de un marco de aplicación del Framework planteado

Como consecuencia se establecen las pautas de aplicación del Framework definido: sus requisitos para la aplicación, herramientas que pueden ser empleadas y sus resultados esperados. Esta etapa pretende delimitar los alcances del Framework, la manera de aplicarse y lo que se puede esperar como resultado de su uso, al final se obtendrá un "mapa de carretera" relacionado con el Framework y su uso.

1.4.2.7. Aproximación a la aplicación del Framework en el proceso de articulación

Finalmente se presenta un caso de aproximación a la manera en que se realizaría el proceso de articulación de la media técnica con el politécnico, a través de la aplicación del Framework en aquellos módulos orientados a formar en el estudiante competencias algorítmicas. Esta etapa pretende brindar un posible escenario en el cual se esboce los resultados que podrían esperarse al aplicar el Framework en un escenario real.

1.4.3. Recursos y Técnicas de recolección de información

Para el desarrollo de este trabajo se recopilará de manera organizada información bibliográfica relacionada con Pensamiento Computacional, empleando fuentes como: libros, artículos, revistas y sitios web, que contengan prácticas de enseñanza-aprendizaje en las ciencias de la computación, resultados de investigaciones y exploraciones realizadas por algunos de los autores citados, con el fin de establecer las bases para la construcción de la propuesta y su sustentación, a la luz de planteamientos realizados por diferentes autores, acompañados de su respectiva sustentación.

1.5. Conclusiones del capítulo

A este punto se ha presentado un panorama general de las temáticas que sustentan el desarrollo del proyecto, partiendo de la necesidad de entender la importancia de la abstracción y la construcción de modelos en el análisis de problemas como núcleo de toda actividad ligada a la solución de problemas. Es así que cada uno de los elementos que le siguen se consideraría una acumulación de saberes, necesarios para la aplicación del pensamiento computacional.

La esquematización de los pasos requeridos para hallar la solución se traduciría entonces en una estructura algorítmica, que debería estar acompañada de los puntos de verificación que permitan certificar sus resultados. Éstos deberán finalmente ser traducidos a un lenguaje de programación específico, establecido por las necesidades mismas del problema planteado, y requiriendo para ello el logro de un conjunto de saberes o competencias que puedan superponerse a la aplicación de metodologías instructivas como el pensamiento computacional.

Sin embargo, de la lectura consciente de estos referentes teóricos es posible establecer una dependencia entre el nivel de abstracción (o especificidad, cuestión de perspectiva) en la descripción del problema y la variación en el conjunto de características disponibles para plantear la solución, llevando a pensar que mientras más detallada sea la descripción, es posible obtener conjuntos de atributos más dicientes y significativos en la naturaleza del problema, evitando de esta manera que pudiera omitirse algún tipo de característica que fuese apreciable, y obteniendo una solución de mejores prestaciones.

Es así que se hace evidente la necesidad de unificar el entendimiento de los principios que fundamenten al pensamiento computacional en el desarrollo del presente trabajo, de modo que sea posible orientar el entendimiento de las características del problema y facilitar el necesario proceso de abstracción, permitiendo estudiante definir aquellos enfoques que darán como resultado las posibles soluciones al problema, siendo consistentes con su simplificación a través del principio de “divide y vencerás”, base conceptual en el desarrollo de software tradicional.

2. Revisión Literaria.

2.1. Caracterización de las prácticas para la enseñanza de la algoritmia

Existen diferentes enfoques pedagógicos empleados en la enseñanza de la algoritmia, y en general no podría hablarse de cuál sería el más o menos usado, ya que no se logró ubicar estudios globales que permitan establecer condiciones de comparación o puntos de referencia homogéneos. Sin embargo, puede observarse en lo escrito por el Consorcio de Habilidades Indispensables para el Siglo XXl (2007), al igual que en lo expresado por Aktunc (2013), Futscek (2011), Dagiené (2011) y Ala-Mutka (2004), entre otros autores, que existen esfuerzos por dejar atrás las prácticas tradicionales de transmisión de conocimientos de forma magistral. Dicha transformación ha resultado ser un proceso lento, y aunque cada día más docentes generan la inquietud de innovar los métodos de enseñanza en sus aulas, aún son una gran minoría frente al total.

Es importante resaltar que en la actualidad el uso de muchas de las prácticas pedagógicas se relaciona con el nivel académico al cual se orienta la enseñanza de los conceptos algorítmicos. Así podría decirse que las prácticas empleadas en educación básica no serían las mismas a usar en secundaria, de igual forma ocurriría a nivel de pregrado universitario frente a los dos niveles anteriormente descritos. Esto puede tener su origen en la diversidad de profesiones y profesionales involucrados en el proceso de enseñanza-aprendizaje, ya que en los niveles de formación de base (primaria y secundaria) cuentan con una planta docente donde, según lo expresa Barrera (2012), son mayoría los docentes con formación normalista o en licenciaturas ligadas a áreas específicas del conocimiento (con un 75%) frente a aquellos con formación profesional en diversas áreas (con un 25%), mientras que para el nivel universitario es obligatorio el nivel de formación profesional e inclusive acompañado de formación complementaria en especializaciones, maestrías y/o doctorados.

Para los licenciados, el proceso pedagógico hace parte de su base formativa, y como puede evidenciarse en las descripciones disponibles en el Observatorio de la Universidad Colombiana (2013), su labor a desempeñar equilibra el conocimiento en áreas específicas del conocimiento y se orienta específicamente al quehacer docente. La inquietud por experimentar y transmitir conocimientos de manera comprensible y haciendo uso de modelos pedagógicos con participación activa del estudiante en la temática o contexto particular hacen parte de su haber, y por tanto, la práctica docente se realiza en un ambiente donde la transmisión del conocimiento debe diferir del modelo pedagógico magistral tradicional. Sin embargo, su conocimiento en áreas técnicas es limitado, y se orienta principalmente a la formación en la aplicación tecnológica, más que en la práctica técnica. Por otra parte, los profesionales tienen un mayor énfasis en los aspectos técnicos de la práctica, pero todo el contexto pedagógico les es ajeno en su formación de base, siendo necesaria una formación complementaria (p.ej., cursando un diplomado en docencia) para ejercer

su labor como docentes. Es en este caso que se puede observar con mayor frecuencia el uso de modelos de transmisión ligados a la explicación de acciones de tipo instructivo.

A continuación se presentan algunos modelos que, por su naturaleza difieren de las prácticas magistrales tradicionales, siendo aplicados con éxito en instituciones de todos los niveles a través de herramientas que serían el soporte para los modelos pedagógicos que involucran activamente al estudiante en su proceso de formación algorítmica.

2.1.1. Enseñanza de la algoritmia a través de modelos, representaciones y analogías

Para el caso de los estudiantes más jóvenes, Bischof (2011) plantea la posibilidad de realizar una introducción al ámbito de las ciencias de la computación a través de la definición simple de los conceptos de control: autómatas y máquinas de estado finito. En ambos casos, la representación de objetos de la realidad (p.ej., máquinas expendedoras de golosinas) brinda la explicación de los conceptos informáticos más simples mediante la descripción de su comportamiento: acción, reacción, secuencia, paralelismo, inicio y fin. Otros conceptos computacionales, como serían la estructuración de instrucciones para el logro de un objetivo, el ordenamiento de elementos en una colección, codificación de información, entre otros, son evidenciados a partir de la realización de juegos, dramatizaciones y ejercicios de comparación con objetos reales, con el fin de facilitar su comprensión.

Frente al uso de modelos y metáforas como medio para presentar a los estudiantes los conceptos computacionales y facilitar la enseñanza de los algoritmos, Forisek (2012) explica que su utilidad va más allá de la representación conceptual, llegando inclusive a lograr la construcción de cimientos que soporten estructuras de pensamiento, por demás complejas, a través de la abstracción de sus características representativas, siendo estas de mayor solidez cuando se trata de transferencia de conocimiento entre participantes de culturas divergentes, ya que incitan al cuestionamiento de la interacción y la participación de las partes como pertenecientes a un todo. En la **Ilustración 19** puede observarse la BLUE BALL MACHINE, el resultado de un concurso planteado por el sitio web SOMETHING AWFUL (http://www.somethingawful.com/) para representar la manera en que pequeñas partes pueden interactuar entre sí para obtener resultados más complejos. La idea es que se entregaban pequeños escenarios de interacción de las bolitas azules, siendo construidos de manera independiente y posteriormente integrados en un esquema macro. Este mismo enfoque para el desarrollo de un trabajo colaborativo puede encontrarse en los trabajos de robótica realizados por Chernova (2008), enfoques de inteligencia computacional como los descritos por McManus (1996) y el enfoque algorítmico presentado por Voyiatzaki (2004) entre otras disciplinas.

Otra perspectiva como la presentada por Allan (1997) establece una relación directa en las habilidades de pensamiento a través del uso de modelos matemáticos, la ejemplificación en solución de problemas lógicos y de pensamiento, permitiendo al estudiante evidenciar sus avances mediante el auto-cuestionamiento, la recapitulación y la documentación de lo que hace para

alcanzar la solución al problema. Existen además enfoques orientados a presentar los conceptos ligados a las ciencias computacionales de manera didáctica, como sería el enfoque presentado por Tim Bell (2010) en su ya conocido Computer Science Unplugged, el cual ilustra el manejo de los números binarios y la construcción de algoritmos a través de la realización de actividades lúdicas que faciliten el entendimiento del concepto sin necesidad de entrar en los detalles de su implementación.

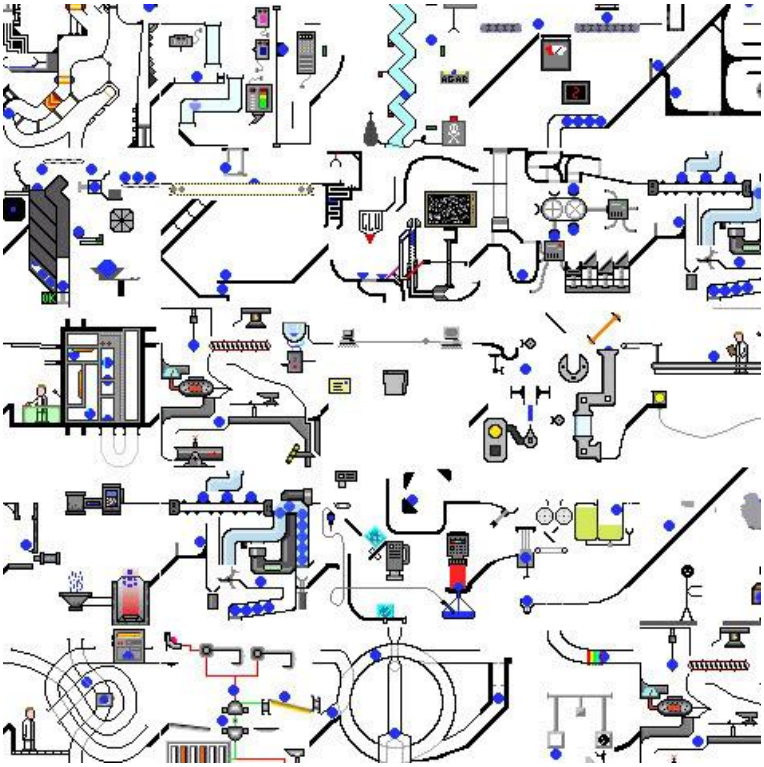


Ilustración 19. Representación de los conceptos de estado y secuencia a través de la Blue Ball Machine (Fuente: http://www.somethingawful.com/)

Existen además enfilamientos conceptuales como el provisto por Hromkovik (2011), quien plantea el hecho de hacer entender a los estudiantes las características del mundo a partir de la posibilidad de representarlos como algoritmos. Su aplicación transversal con otras áreas del conocimiento – como sería el caso de la física, la economía o las ciencias sociales – no debe ser simplemente el conocimiento en el uso de las herramientas computacionales de oficina o de paquetes complementarios, sino en entender como el análisis de la información y la representación de sus flujos de transformación pueden llevarse a cabo a través de modelos, estructuras y simulaciones. En la **Ilustración 20** puede evidenciarse la perspectiva del caricaturista ***Bill Amend*** acerca de la interiorización de la solución algorítmica a problemas cotidianos para las nuevas generaciones.

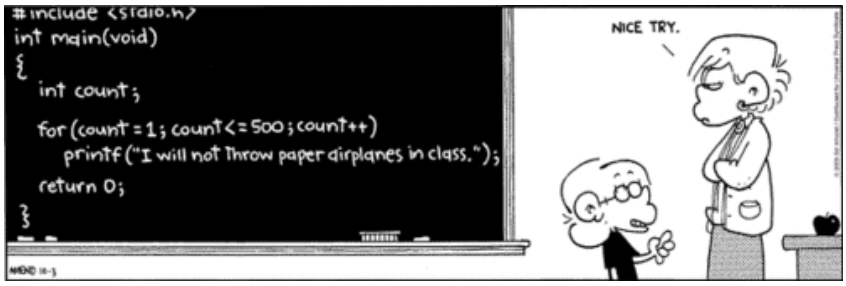


Ilustración 20. Representación del análisis algorítmico del estudiante frente al problema que representa escribir en el tablero una frase en repetidas ocasiones. (Fuente: http://www.foxtrot.com/)

2.1.2. Enseñanza de la algoritmia a través de mini-lenguajes y simulaciones

Los mini-lenguajes, conforme lo presenta Brusilovsky (1997) corresponden a una representación reducida de los lenguajes de programación de alto nivel, compuestos por un conjunto reducido de instrucciones que facilitan la manipulación de un entorno limitado y realizar en su interior conjuntos de tareas específicas a través de un actor. Su idea fundamental se centra en el hecho que si los estudiantes aprenden el concepto de estructuras básicas de control y la manera de aplicarlas en ambientes reducidos y controlados, sería más sencillo extrapolarlo a situaciones donde los lenguajes y los ambientes tienen a aumentar en complejidad.

Quizás uno de los mini-lenguajes más famoso es el LOGO, propuesta formulada por Papert (1980) que contiene estructuras básicas de control que le permiten a una entidad – conocida como tortuga – realizar acciones en la pantalla y construir figuras a través de su recorrido por ella. Este lenguaje no fue construido como un medio para enseñar programación, sino para permitir a los estudiantes el desarrollo de habilidades en la solución a problemas específicos y a la revisión de sus resultados a partir del seguimiento a las acciones mediante la realización de pruebas de escritorio y retroalimentación de sus resultados. Las secuencias de control incluyen el manejo de condiciones, ciclos, archivos, entre otras facilidades.

LOGO fue el precursor de otras iniciativas en la enseñanza, como la presentada por LEGO al incorporarlo en un juego que permitía controlar construcciones de fichas a través de la programación de sus movimientos mediante la manipulación de motores eléctricos. La programación se realizaba directamente en computadores haciendo uso del lenguaje LOGO. Esta herramienta fue utilizada ampliamente durante los años 80 y principios de los 90, siendo el precursor del área de robótica experimental y didáctica de LEGO, conocida como MINDSTORMS, algunas experiencias significativas en el uso de esta herramienta pueden evidenciarse en los escritos de Brauner (2010), McWhorter (2009), Lui (2010) y Hamada (2010).

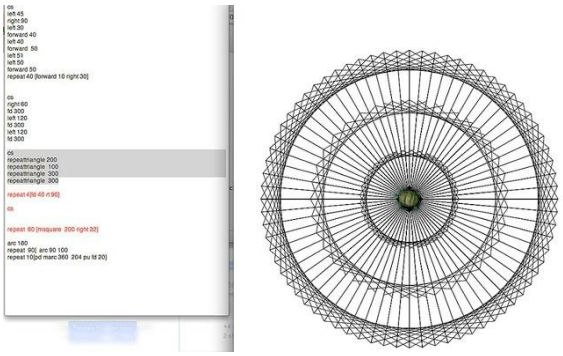


Ilustración 21. Uno de los muchos ambientes de trabajo para LOGO (Fuente: www.flickr.com)

Otra iniciativa similar al LOGO sería la presentada por Rich Pattis en los años 80: ***KAREL el Robot***. Según Pattis (1995), KAREL posee fundamentación en el lenguaje de programación PASCAL, y conforme es descrito por Roberts (2005) eso hace que sus programas posean un enfoque estructurado por medio del cual los estudiantes debían enseñar a un actor a resolver problemas simples de razonamiento espacial y deducción lógica a través de la definición de estructuras de control secuenciales y su programación mediante un conjunto de comandos simples.

Con el pasar de los años surgirían versiones de KAREL más acordes a los paradigmas de programación predominantes en la industria. Es así como por ejemplo, Joseph Bergin describe los casos de KAREL++, (1996), KAREL learns JAVA (2013) y Monty KAREL (2013), mientras Hvorecky (1992) presenta una visión similar para KAREL 3D. En el primero, KAREL vive en un mundo rodeado de objetos con los cuales puede interactuar a través de comandos equivalentes a los empleados por C++ y JAVA. En el segundo caso, si bien se mantiene la esencia del actor, el conjunto de instrucciones son derivadas únicamente del lenguaje de programación JAVA. El tercero por su parte maneja los comandos de control en PYTHON, mientras que el último se desenvuelve en un mundo donde se expande el área de trabajo de forma tridimensional. Tal fue la acogida del KAREL como medio para la enseñanza de la algoritmia, que ***Charles Schultz***, creador de la tira cómica **PEANUTS** (Carlitos y Snoopy en algunos países latinos) dedicó una tira a la aplicación de la lógica de KAREL para la solución de problemas cotidianos, como puede evidenciarse en la **Ilustración 22**.

Ilustración 22. Caricatura de PEANUTS que habla del uso de la lógica de KAREL en situaciones reales (Fuente: http://jinkchak.wordpress.com/tag/karel/)

Otras iniciativas como las propuestas por Loyarte (2006) se enfocan en flexibilizar el proceso de aprendizaje, fundamentándose en el desarrollo de la lógica de programación sin adentrarse en la construcción de estructuras sofisticadas de código, manejando un conjunto de instrucciones básico y de acción limitada en estructuras de datos, que complementándose con las experiencias esbozadas por Chiarani (2013) se apoyen en estructuras visuales que permitan al estudiante enfocarse en la solución del problema, más que en construir segmentos de código funcionales al abreviar aspectos que, a vista general de código, serían demasiado "abstractos". Un ejemplo claro de este enfoque es PSeInt[1]. Según Rosas (2014), esta herramienta se ha abierto paso como medio de aprendizaje no solo en las ciencias computacionales, ya que su simplicidad y el uso de pseudocódigo para la construcción de los algoritmos, facilita el acercamiento y apropiación de su funcionamiento por parte del estudiante.

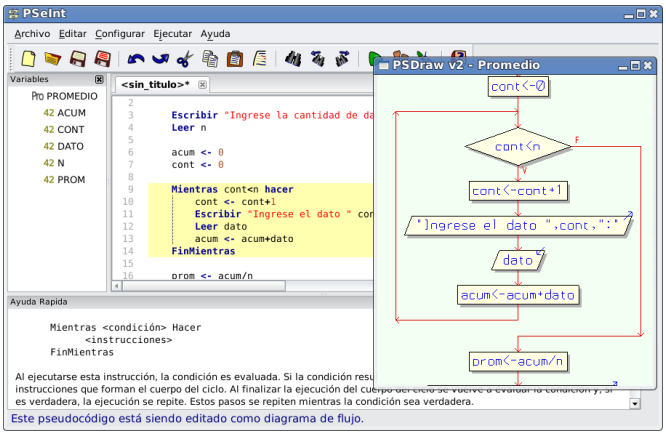


Ilustración 23. Presentación de la interfaz de PSeInt como ambiente de desarrollo acompañado de estructuras visuales. (Fuente: http://www.bitacorainformatica.com/)

Como un complemento a la labor realizada por los mini-lenguajes puede encontrarse la posibilidad de representar visualmente el comportamiento de las instrucciones de control impartidas para un conjunto diverso de elementos – y no necesariamente uno solo, como ha sido hasta ahora – es en este contexto que las simulaciones muestran su utilidad, ya que permiten retroalimentar el trabajo realizado, validando su estructura y efectividad. Según Cernuda (2004) las simulaciones son representaciones de comportamiento que buscan reproducir situaciones o acciones de la forma más real posible, y en el caso de la algoritmia, presentar al estudiante la visualización del comportamiento de estructuras algorítmicas construidas – normalmente – en un mini-lenguaje, apelando a su aspecto kinestésico.

Al interior de este enfoque puede destacarse una propuesta basada en entornos gráficos que permite la manipulación de objetos al interior de un entorno de trabajo: SQUEAK. Ésta propuesta permite al estudiante emplear estructuras de control orientadas a objetos para manipular los elementos que se ubiquen al interior del área de trabajo, partiendo de un guion de

[1] Siglas correspondientes a PSEudo INTérprete.

comportamiento de dichos objetos en esencia similar a las animaciones de FLASH. En este orden de ideas, Fernández (2006) manifiesta que el lenguaje se orienta a la construcción de secuencias de acción que permiten reflejar la construcción de los conceptos y no solo la representación de sus cualidades.

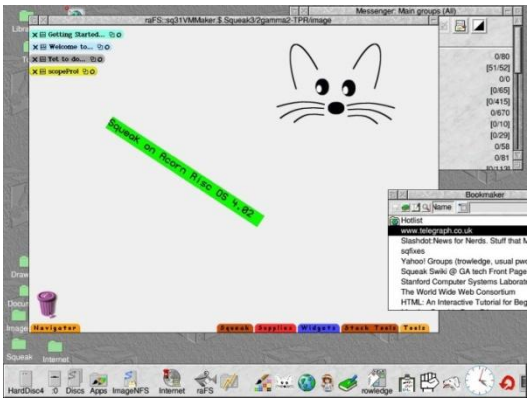


Ilustración 24. Presentación de la pantalla de SQUEAK como ambiente de desarrollo simplificado (Fuente: wiki.squeak.org)

Otro enfoque en el uso de las simulaciones sería el provisto por ALICE, presentado por Aktunc (2013) como un entorno software construido con el fin de acercar a los estudiantes a una experiencia de programación sencilla mediante el uso de estructuras algorítmicas y secuencias de control básicas aplicadas a objetos sumergidos en un universo discreto. Su enfoque inicial se orienta a estructuras de control simples, soportando elementos como el razonamiento abstracto, el pensamiento algorítmico y los principios fundamentales de la programación. Para su tercera versión, la herramienta realiza un acondicionamiento del estudiante que le facilite una transición al paradigma orientado a objetos y el aprendizaje del lenguaje de programación JAVA.

Ahora bien, en el contexto de la simulación y el uso de entornos gráficos, SCRATCH sería otra herramienta que permitiría acercar al estudiante conceptos de secuencia de control, ciclos y reutilización de bloques de código de manera más simple, y de la misma manera que ALICE, y que según Resnick (2011) obvia el ahondar en estructuras algorítmicas complejas.

SCRATCH tiene un campo de acción diverso, permitiendo crear historias interactivas, juegos y simulaciones a través de la interacción con los objetos que sitúan en su área de trabajo, disminuye la complejidad de interactuar con conjuntos densos de instrucciones y sin embargo permite obtener resultados interesantes inclusive para quienes no han tenido un contacto prolongado con las tecnologías de información, soportando un modelo de pensamiento, análisis de casos y diseño de soluciones cimentado en el enfoque del pensamiento computacional. Experiencias como las exhibidas por Brauner (2010) permitieron evidenciar que SCRATCH facilitaba a los estudiantes la apropiación del conocimiento básico necesario para crear secuencias de control simples para controlar un pequeño robot simulado. En la **Ilustración 25** puede observarse lo intuitivo de su

distribución, simplificando la identificación de las estructuras de trabajo y evidenciando la sencillez de las construcciones requeridas para desarrollar estructuras algorítmicas.

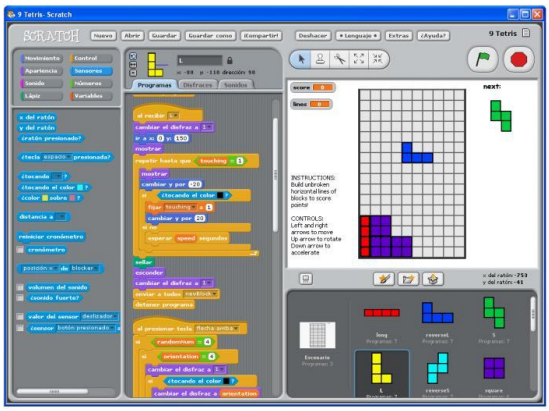


Ilustración 25. Presentación de la pantalla de SCRATCH como ambiente de desarrollo (Fuente: mit-scratch.softonic.com)

Ha sido tal el éxito conseguido por SCRATCH debido a la simplicidad de su entorno de trabajo y lo intuitivo en la construcción de estructuras algorítmicas que GOOGLE, adoptando estos mismos principios, ha creado un editor de programación que funciona a través del apilamiento e interconexión de bloques visuales: BLOCKLY.

BLOCKLY puede ser empleado a través de un browser, lo cual le brinda una ventaja inmensa frente a SCRATCH, ya que este último debe ser instalado como una aplicación de escritorio, y como ventajas adicionales podría enumerarse la posibilidad de emplearse para construir juegos simples que hagan uso de conceptos básicos como condicionales y ciclos, y al igual que SCRATCH, posee una interfaz que le permite interactuar con las estructuras robóticas de Lego Mindstorms. Marron (2012) explica que su acogida ha sido tal, que sistemas comerciales como SNAPP [2](orientado al desarrollo de prototipos de apps móviles) hace uso de la interfaz de BLOCKLY para facilitar la construcción de prototipos funcionales empleando metodologías rápidas de desarrollo de software. Además, la interfaz permite la traducción del código generado a lenguajes como Python y JavaScript. La **Ilustración 26** presenta la interfaz de BLOCKLY, obsérvese las grandes similitudes que guarda con SCRATCH.

[2] Accesible desde la URL: https://snapp.click/

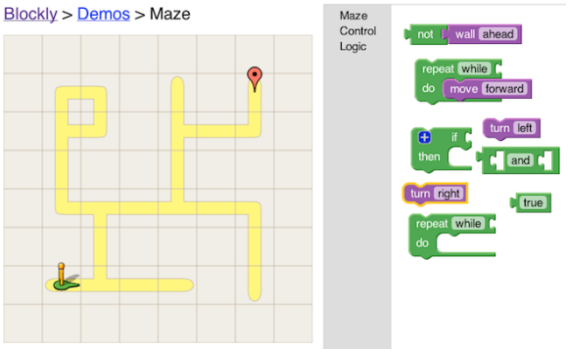


Ilustración 26. Presentación de la interfaz de BLOCKLY, obsérvese las similitudes con SCRATCH. (Fuente: http://www.wired.com)

2.1.3. Enseñanza de la algoritmia a través de juegos

La participación de los juegos en el proceso de enseñanza de la algoritmia es relatada por Prensky (2001) como una manera de capturar la atención del estudiante en relación a la obtención de aprendizaje significativo al involucrar actividades de planeación, análisis y resolución de problemas que involucren la lógica. Los juegos, debido a su componente lúdico, han sido por excelencia la manera más simple de capturar la atención de las personas. De igual forma Futsek (2011) manifiesta que es precisamente la curiosidad de su funcionamiento lo que permite lograr atención a los detalles sin necesidad de ahondar en los aspectos teóricos. En este aspecto, Shabanah (2009) presenta los modelos de aprendizaje de BLOOM, GAGNE y el ENFOQUE CONSTRUCTIVISTA como opciones a la enseñanza de algoritmos, brindando facilidades al estudiante para analizar el comportamiento del juego con el objetivo de obtener una estructura algorítmica de su propia autoría, siendo validada posteriormente contra el juego y permitiéndole retroalimentarse de los eventos de éxito o fracaso como criterios de análisis de opción. En la **Ilustración 27** se puede observar la estructura general de los tres modelos, que en todos los casos basan su estrategia en el análisis del comportamiento del juego una vez que ha sido jugado, permitiendo la construcción del algoritmo y su verificación a priori o a posteriori.

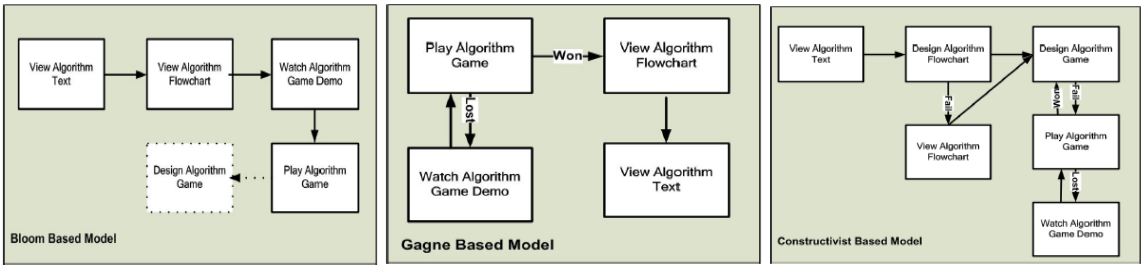


Ilustración 27. Simplificación de los modelos de Bloom, Gagne y el enfoque Constructivista. (Fuente: Shabanah, 2009)

Retomando el concepto de abstracción, y extrayendo al estudiante de un entorno de trabajo computacional para llevarlo a un entorno más simple, existe una experiencia en la implementación del enfoque lúdico a través del juego TIM THE TRAIN el cual permite fortalecer el pensamiento algorítmico de los estudiantes a través de la aplicación de instrucciones simples a la manipulación

de los objetos en su interior, de esta forma las secuencias, los condicionales y las iteraciones, permiten llenar los compartimientos de los vagones del tren empleando unas figuras similares a las usadas en TETRIS sin desbordar su carga o desperdiciar figuras. Así, el aprendizaje de los conceptos algorítmicos constituyen en primer lugar a la definición de secuencias de instrucciones que permitan llenar los vagones del tren, o por el contrario, a la identificación de los pasos que fueron necesarios para obtener un resultado de figuras apiladas presentado al estudiante con anterioridad, como puede evidenciarse en la **Ilustración 28**.

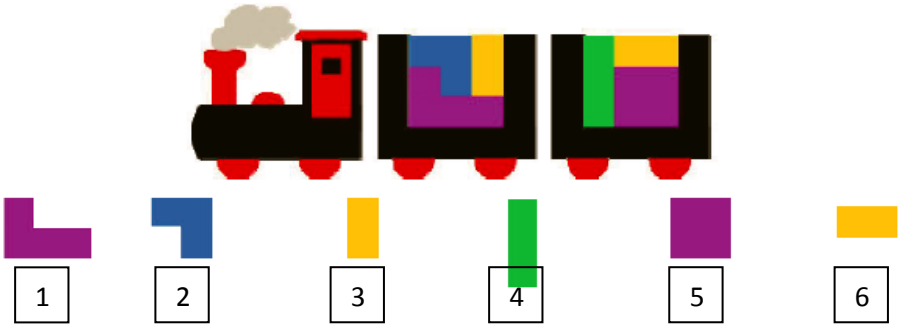


Ilustración 28. Representación de los elementos que componen el juego TIM THE TRAIN, los vagones de almacenamiento y las fichas que deberán ser situadas en un orden específico. (Fuente: Futschek, 2011)

Como se mencionó, TIM THE TRAIN se cimenta en uno de los juegos por excelencia para el desarrollo de la lógica y la habilidad mental: TETRIS. Creado en la Rusia Comunista como medio para probar la capacidad de procesamiento de los equipos de cómputo de la época, permite desarrollar las estructuras de pensamiento algorítmico desde la fundamentación misma del juego: minimizar el número de filas existentes en el tablero al maximizar el número de filas borradas empleando cualquier secuencia de fichas dada en forma aleatoria, todo esto en el menor tiempo posible.

TETRIS posee un conjunto de acciones (o instrucciones) que pueden ser ejecutadas sobre las fichas que se encuentren en juego, éstas instrucciones deben considerar no solo la situación de la ficha en desplazamiento, sino la situación de las fichas que se encuentran actualmente en el fondo del tablero y la siguiente ficha por jugar, de esta manera, el jugador deberá entrelazar los conceptos de causa-efecto con el fin de establecer secuencias de pasos que, realizadas "en vivo" alteran el comportamiento de las fichas y permiten obtener un resultado particular. En la **Ilustración 29** se presenta un acercamiento al algoritmo de análisis de la ficha en juego, donde el punto de partida sería la existencia del juego en sí, obsérvese que las decisiones, las iteraciones y las instrucciones simples hacen parte de la naturaleza del juego, y el estudiante puede identificarlas de manera intuitiva, sin establecer el concepto técnico que les da soporte.

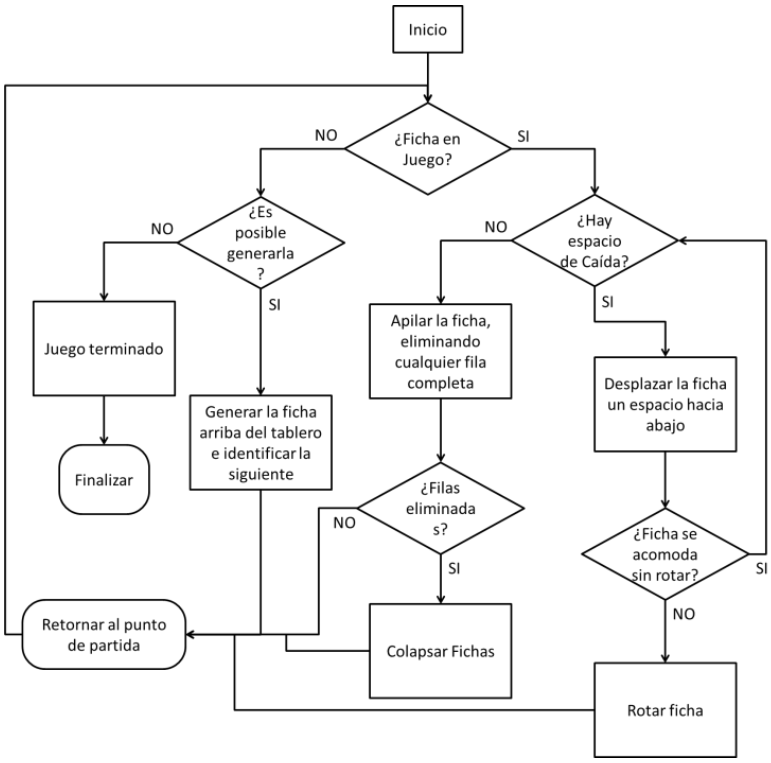


Ilustración 29. Representación simplificada del algoritmo de juego para TETRIS (fuente: el autor)

Ahora bien, alejándose un poco del concepto netamente lúdico y hablando de los juegos con una orientación marcada hacia lo educativo, las posibilidades de aplicación y la cantidad de elementos disponibles serían demasiado grandes como para reseñar detalladamente cada uno de ellos. Podría abarcarse desde los juegos que permiten a los niños más pequeños la identificación de figuras, colores, animales, posicionamiento y relación causa-efecto (p.ej., JENGA) siendo este el primer acercamiento a un pensamiento analítico y de resolución de problemas, y que según Shabanah (2009), le permite llegar de manera más simple a los simuladores de mayor complejidad como los de aviación y manejo de maquinaria pesada.

Tómese por ejemplo al juego de las RANAS SALTARINAS, en éste se busca que el estudiante logre intercambiar las ranas que se encuentran a un lado del rio con las que se encuentran al otro lado. Para completar dicha tarea es necesario acatar y cumplir con restricciones que elevan su nivel de complejidad (avanzar solo un espacio, no poder saltar dos ranas consecutivas, entre otras) que obligan al estudiante a observar y analizar bien la problemática y sus restricciones antes de aventurarse a mover las ranas. En este orden de ideas, se esperaría como resultado que el estudiante defina una serie de pasos ordenados que le permitan lograr exitosamente el intercambio sin bloquear las ranas, además, que el estudiante esté en capacidad de explicar que fue lo que hizo y como dio solución al problema, porque no tiene sentido que el estudiante encuentre la solución y no sea capaz de reproducirla. La **Ilustración 30** permite evidenciar la estructura del juego y la secuencia de solución, expresada de una manera simple.

Ilustración 30. Representación básica del JUEGO DE LAS RANAS SALTARINAS y su solución. (Fuente: ajedrezyrazonamiento.bligoo.es/juego-de-logica-las-ranas-saltarinas)

Otros juegos clásicos para la definición de secuencias lógicas que permitan obtener resultados similares serían CRUZA EL RIO y las TORRES DE HANOI. El objetivo del primero sería permitir a un grupo de 4 actores cruzar un rio, donde para poder hacerlo es necesario cumplir una serie de condiciones, entre las que se encuentran: la existencia de dos tipos de actores que no pueden estar solos, de los cuatro actores disponibles solo tres pueden cruzar a la vez, y siempre debe haber alguien que controle el medio de transporte empleado para transportar a través del rio. Para este juego pueden encontrarse variaciones con familia (padre/madre/hijo/hija), granjero (granjero/lobo/cabra/col), entre otros. El objetivo del juego de las TORRES DE HANOI seria transportar un grupo de 3 o más discos, ordenados de menor a mayor, desde la torre 1 hasta la torre 3 en el menor número de movimientos posible, teniendo cuidado de no situar discos de mayor tamaño sobre discos de menor tamaño. Este es un ejercicio clásico en la enseñanza de algoritmos y complejidad algorítmica, conforme es presentado por Minsker (2008), Stadel (1984), Maniccam (2012) y Konopasek (1985), requiriendo de análisis detallado por parte del estudiante. La **Ilustración 31** presenta una representación del juego y una referencia a su solución.

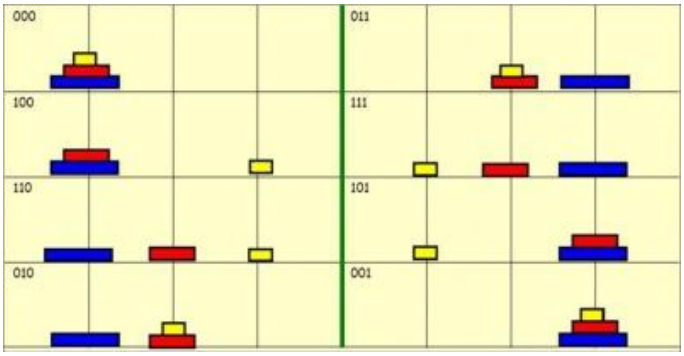


Ilustración 31. Representación básica de las TORRES DE HANOI y su solución. (Fuente: http://innovacioneducativa.upm.es/pensamientomatematico/node/76)

Otros juegos permiten la visualización y representación de algoritmos a través de figuras, colores y sonidos. En ese sentido, la representación visual se orienta más a presentar el comportamiento de un algoritmo y la manera en que este se altera al modificar alguna de sus variables. Shabanah (2009) presenta una amplia recopilación de algunos de estos juegos y el enfoque al que se orientan.

2.1.4. Enseñanza de la Algoritmia a través de Juegos de Programación

Conforme a la descripción presentada por Denning (2005), las ciencias computacionales se enfocan en la obtención de información del entorno y la forma en que esta puede ser procesada con un fin específico. Como ciencia, sus cimientos se encuentran en el uso de las matemáticas y la ingeniería, y en este sentido es el mismo Denning (1989) quien habla de la representación básica de los objetos de interés a través de algoritmos como soporte principal a la construcción de software, haciendo que la equivalencia "*ciencia de la computación es igual a la programación*" se considere especialmente válida en el contexto formativo actual.

Es amplia la discusión derivada de este enfoque, incluyendo en su labor disciplinar aspectos relacionados con diseño y estructuración de infraestructura de hardware y soportes de información. Y aunque no se ahondará en los diversos enunciados que le dan sustento, se tomará entonces como punto de partida la creencia expresada por Denning (1989), según la cual los lenguajes de programación facilitan la expresión de estructuras de solución orientadas a la transformación de información mediante el uso de algoritmos. En este orden de ideas, es claro que la programación es una práctica estándar, que permite alcanzar cierto nivel de competencia siempre que logre concienciarse a los estudiantes de que los lenguajes de programación son solo un vehículo que permiten el acceso a elementos fundamentales de la profesión.

Bajo esta premisa surgen infinidad de iniciativas que buscan que los estudiantes aprendan el concepto de programación a través del aprendizaje directo de las estructuras de codificación, empleando actividades lúdicas que estimulen el logro de objetivos intermedios y acumulativos en complejidad, de manera tal que el aprendizaje de la algoritmia esté supeditado al aprendizaje de las estructuras de control ligadas a un lenguaje de programación en particular.

Existen entornos que buscan enseñar lógica y conceptos generales de programación, sin adentrarse demasiado en la complejidad de las estructuras de codificación inherentes a un lenguaje de programación. En este conjunto puede enumerarse iniciativas como KODU[3], la cual permite la creación de juegos simples a partir de la integración de juegos pre-construidos y el establecimiento de relaciones entre ellos, sin necesidad de requerir conocimientos previos en estructuras algorítmicas.

Dentro de las iniciativas lúdicas, cercanas a los juegos de rol, puede encontrarse iniciativas como Code Combat[4], la cual permite aprender a través de las aventuras en primera persona de un mago, quien debe resolver una serie de retos empleando para ello hechizos construidos en JavaScript. Code Avengers [5]es otro sitio en el cual se plantea el proceso de aprendizaje de lenguajes orientados a la web, escalando siempre en complejidad para lograr metas de aprendizaje. Fight

[3] Disponible en http://www.kodugamelab.com/about (última fecha de consulta: Abril 1 de 2014).
[4] Disponible en http://codecombat.com (última fecha de consulta: Abril 1 de 2014).
[5] Disponible en http://www.codeavengers.com/ (última fecha de consulta: Abril 1 de 2014).

CodeGame[6] y RoboCode[7] son otros sitios que permiten aprender JavaScript, .NET, y Java a través de la programación de robots que deben luchar entre sí, cumplir tareas y sortear obstáculos diversos.

Otras iniciativas[8] más simples buscan desarrollar las habilidades inherentes a la programación desde temprana edad, y aunque puede entenderse que por su simplicidad pueden no ser muy llamativas para estudiantes de secundaria o pregrado universitario, si pueden ser tomadas como referencia para el establecimiento de metodologías que faciliten la interiorización de conceptos algorítmicos. La primera de ellas sería "Daisy the Dinosaur", la cual permite desarrollar habilidades lógicas ya que su objetivo sería que Daisy realice una serie de actividades, orientada a través de instrucciones precisas. En un nivel más básico estaría Kodable, una aplicación que orienta a los estudiantes en la resolución de problemas simples al elegir caminos adecuados para poder avanzar entre los niveles. Finalmente se puede encontrar a Hopscotch, una aplicación que permite a los estudiantes la construcción de animaciones y juegos simples empleando bloques de código para la estructuración de programas, similares en su estructura a los empleados en Scratch.

Obsérvese que en ninguna de estas iniciativas (o en otras más que pueden encontrarse a través de la web) se habla del desarrollo de competencias explícitamente en el área de la algoritmia, sin embargo, bien pueden enfocarse al fortalecimiento de las habilidades en pensamiento lógico o en la aplicación de instrucciones de control ligadas a un lenguaje de programación particular. Así, el entendimiento del funcionamiento del algoritmo y su papel en el procesamiento de la información debe ser complementario al proceso formativo, convirtiéndose en algo fundamental si se desea desarrollar las competencias requeridas para la solución de problemas a través del uso del razonamiento lógico. Es en este punto donde surge la inquietud acerca del mejor modo de enseñar los fundamentos de las ciencias computacionales, y hasta qué punto su aprendizaje a partir del desarrollo directo de código en un lenguaje de programación (o micro-lenguaje, cualquiera sea el caso) es realmente una buena práctica, finalmente se resumiría la inquietud en si lo que se busca es el desarrollo de habilidades algorítmicas o simplemente conocimientos en programación.

2.1.5. Enseñanza de la algoritmia a través del pensamiento computacional

Es posible identificar factores comunes en las metodologías de enseñanza presentadas hasta este momento:

1. Se fundamentan en la necesidad de abstraer las características más relevantes de un problema.
2. Requieren del análisis de dichas características con el objetivo de definir una secuencia de pasos ordenados que conlleven a una solución

[6] Disponible en http://fightcodegame.com/ (última fecha de consulta: Abril 1 de 2014).
[7] Disponible en http://robocode.sourceforge.net/ (última fecha de consulta: Abril 1 de 2014).
[8] Daisy the Dinosaur, Kodable y Hopscotch son aplicaciones diseñadas para plataformas iOS.

3. Involucran de manera activa al estudiante en la construcción de estructuras algorítmicas para el problema, su construcción y verificación.

Existe un enfoque de pensamiento que reúne estas y otras características para ser aplicadas al proceso de enseñanza-aprendizaje: el *Pensamiento Computacional*. Aunque si bien el pensamiento computacional no se enfoca a enseñar a programar – al igual que otras tecnologías igualmente provechosas para tal fin, como fue LOGO – sino en desarrollar la capacidad mental y de abstracción en un enfoque multidisciplinar de entender y enfrentar los problemas, la posibilidad de influenciar de manera positiva otras disciplinas, conforme lo manifiesta Bundy (2007) hace que el sentido de procesamiento y algoritmo tengan una orientación más abstracta, de esta manera es posible pensar que su participación en el fortalecimiento de las ciencias de la computación sea más que evidente.

Experiencias como las presentadas por Yadav (2011) y Eisenberg (2010), ponen de manifiesto la utilidad del pensamiento computacional en el desarrollo de la lógica y la abstracción de problemas, donde el uso del computador es aportante, pero no determinante para lograr los objetivos de aprendizaje; situaciones como el análisis de la recursividad o el procesamiento paralelo en la vida cotidiana hacen parte del análisis computacional que proponen en sus escritos. Por su parte, Serafini (2011) presenta la manera en que la enseñanza de la programación fundamentada en el uso de LOGO ha permitido incrementar los niveles de atención de los estudiantes, disminuyendo sustancialmente la preocupación por el aprendizaje del lenguaje – dada su simplicidad – e incrementando el interés por la composición de secuencias de pasos coherentes y que apunten al resultado esperado.

Otras experiencias como las presentadas por Rosas (2014) y Flores (2011) permiten entender que las habilidades desarrolladas a través del pensamiento computacional no son exclusivas de las ciencias computacionales, ya que otras áreas como la ingeniería electrónica, la ingeniería en diseño del producto, las matemáticas discretas, e inclusive el desarrollo de las habilidades fundamentales de comunicación y aritmética pueden influenciarse positivamente de los principios propuestos por Jeanette Wing.

2.2. Identificar los Principios que cimentan el Pensamiento Computacional

Hasta el momento no ha sido posible establecer un conjunto de principios aceptados como únicos en la teoría del Pensamiento Computacional, de tal suerte que es obligante realizar una labor de identificación, comparación y contraste de las premisas planteadas por algunos de los autores consultados a la luz del planteamiento inicial de Jeannette Wing.

Para lograr un conjunto único de características orientadoras de este trabajo, se realizó una intersección entre los conjuntos en consideración de las capacidades a desarrollar en el estudiante y del nivel de evolución de las habilidades y conocimientos previos requeridos para hacerlo,

tomando como punto de partida la afirmación de Wing (2013) al indicar que el estudiante debe evolucionar paulatinamente en la estructuración y maduración del pensamiento, de manera que pueda construir un conocimiento sólido, como es evidente y aplicado en el desarrollo de la habilidad matemática a lo largo de los diferentes ciclos de formación del estudiante: básica primaria (aritmética), secundaria (álgebra), media y universitaria (cálculos).

A continuación se esquematiza la esencia en los principios de cada enfoque presentado sobre Pensamiento Computacional, la **Tabla 1** aparea (de ser posible) las equivalencias en cada enfoque presentado en el capítulo 3 y los compara con los planteados por Wing. Este apareamiento permite concentrarse en identificar aquellos principios que son compartidos en mayor medida por los enfoques, brindándoles un mayor grado de importancia.

Tabla 1. Sintetización de Componentes del Pensamiento Computacional. (Fuente: El Autor)

WING	BARR	BRENNAN	ISTE, NSF
Abstracción.	Construir abstracciones.	Abstracción y conceptualización del problema.	Realizar la representación de los datos a través de abstracciones.
Identificar intereses transversales y verticales.			
Dividir el problema.		División y modularización del problema.	
Elegir representación.	Modelar datos.		Realizar la representación de los datos a través de abstracciones. Construir modelos y simulaciones.
Trazabilidad.		Logro de hitos intermedios.	
Identificación de variables.	Analizar los datos.		Organizar y analizar los datos.
Identificación de constantes.	Analizar los datos.		Organizar y analizar los datos.
Identificar posibles soluciones.	Identificar posibles escenarios de solución.	Identificar elementos pre-construidos y aplicables.	Identificar, analizar e implementar diversas soluciones.
Evaluar y elegir la solución más adecuada.	Probar los diferentes escenarios de solución.	Identificar elementos pre-construidos y aplicables. Realizar procesos de verificación, validación y depuración.	Construir modelos y simulaciones. Identificar, analizar e implementar diversas soluciones.
	Realizar simulaciones.		Construir modelos y simulaciones.

	Formular soluciones computacionales.		Formular problemas con soluciones computacionales.
	Construir algoritmos.		Automatización de soluciones a través de algoritmos.
	Generalización y aplicación de solución a escenarios amplios.		Generalización de la solución y transferencia a otros entornos.
		Construcción iterativa e incremental de proyecto.	

Como resultado de este ejercicio, fue posible encontrar que algunos de los principios propuestos por Wing en su postulado inicial son compartidos (en mayor o menor medida) por los demás autores, y que en contraposición estos plantean principios complementarios que no hacen parte del postulado inicial de Wing.

Es lógico entonces que el paso a seguir sea la identificación de aquellos principios orientadores compartidos por todos los autores incluidos en el cuadro comparativo, pudiendo entenderse como una puesta en común o unificación de sus enfoques. Este proceso contará no solo con la argumentación de Wing y los demás autores, sino que tomará apartes de los postulados propuestos por Forisek (2012), Selby (2012), Hazzan (2008), Joyanes (2003) y Allan (1997), entre otros autores, con el fin de establecer su aplicabilidad al proceso de aprendizaje algorítmico.

En primera instancia, se identifica que los principios de *Abstracción, Conceptualización y Representación de los elementos más relevantes del Problema*, la *Identificación de los posibles Escenarios de Solución* y la *Evaluación de Dichos Escenarios*, identificando en todos los casos elementos que hayan sido pre-construidos y que puedan ser incorporados en la solución propuesta, son los elementos transversales a todos los enfoques presentados. Su importancia radica en el hecho que todo problema (computacional en este caso) debe ser enfrentado a través de la abstracción de sus elementos más representativos, según lo presentado por Gamer (2002) y Wiener (1945).

Conforme a este planteamiento, y según lo presentan Serna (2011) y Forisek (2012), posterior a la abstracción de las características de un problema se encuentra la *Elección y representación de las características abstraídas a través de modelos*. Este principio es compartido por al menos dos de los autores referenciados en la tabla comparativa, al igual que la Identificación de Variables y Constantes, que según la descripción de Polya (2004) corresponden al conjunto de datos del problema que deben ser tenidos en cuenta en la solución.

Finalmente, Allan (1997), Ala-Mutka (2004) y Joyanes (2001), entre otros autores, hablan de la importancia de realizar una *división del problema* en elementos atómicos y significativos, de

manera que la solución de cada una de estas partes, en conjunto, establezca la solución del problema en sí. Esta división debe ser *clara* y *trazable* con relación a sus intereses y la participación de su solución parcial en la solución global, conforme a lo descrito por Serna (2011) y Wieringa (1995).

Así, los principios del Pensamiento Computacional identificados y cuya aplicabilidad en el campo de la enseñanza algorítmica serán analizados, se enumeran a continuación:

1. Abstracción, Conceptualización y Representación de los aspectos más relevantes del problema.
2. Identificación de los posibles escenarios de solución.
3. Evaluación de los diferentes escenarios de solución identificados.
4. Elección y representación de las características abstraídas a través de modelos.
5. División del problema en elementos atómicos y significativos.
6. Trazabilidad entre los elementos que hacen parte del problema y la solución.

2.3. Aplicabilidad de los principios en el proceso de Enseñanza de Algoritmos

El proceso de construcción algorítmica se fundamenta en el desarrollo de habilidades y competencias orientadas a la solución de problemas, derivadas principalmente de la formación matemática y en ciencias que el estudiante recibe durante sus años de academia, y que según Polya (2004) son expresadas en forma de heurísticas. Estas definiciones aproximan las características del problema hacia un conjunto de situaciones simplificadas o familiares, empleando para ello modelos abstractos que representen sus atributos más representativos, su comportamiento e interacción.

Tómese por ejemplo el análisis de un cuerpo en movimiento, en el cual es posible identificar atributos relevantes a su análisis (p.ej., velocidad, distancia, tiempo) y la manera en que ellos interactúan entre sí, representados de forma abstracta mediante fórmulas. Otro caso más simple puede encontrarse en la necesidad de calcular el volumen de un objeto rectangular, allí sus atributos más relevantes (largo, ancho y altura) permiten analizar mediante su representación abstracta (fórmula) la manera en que afecta alterar uno o varios de ellos. De esta forma pueden identificarse innumerables ejemplos que llevan a inferir que el estudiante tiene cierta familiaridad con el análisis de los enunciados que rodean a una problemática que le sea presentada y su abstracción.

Sin embargo, conforme a este pensamiento surge una inquietud: si el estudiante ha venido desarrollado habilidades para afrontar y resolver problemas a lo largo de su formación académica ¿Por qué se le dificulta solucionar un problema de manera algorítmica? En este sentido Lyster (2004) expresa que dichas insuficiencias se deben a vacíos en el proceso de abstracción del problema y la definición de su dominio al momento de fraccionarlo en partes relevantes e

interrelacionadas, así el estudiante bien puede entender el concepto computacional y operativo del algoritmo (p.ej., iteraciones), y con cierta dificultad accede a su aplicación de forma aislada (p.ej., recorrido en un vector), pero en muchas ocasiones es incapaz de lograr una articulación alrededor de una solución de mayor complejidad (p.ej., manejo de colecciones, recorrido en una matriz o en un cubo).

Otros enfoques presentados por el mismo autor hacen énfasis en la "fragilidad" del conocimiento, manifestado en la incapacidad para superponer los conceptos computacionales básicos (p.ej., condicionales, iteraciones, uso de variables) a situaciones en la vida real, imposibilitando su comprensión y por ende la manera en que pueden ser empleados para resolver un problema o un conjunto de ellos hacia un nivel superior. Es por ello que Futschek (2011) habla acerca de la posibilidad de emplear elementos de la realidad del estudiante, cuya sinestesia permita a los estudiantes identificar sus características fundamentales a través del uso de los sentidos, construyendo metáforas que facilite su apropiación.

Revisando los escritos de Barriga (2002), Brauner (2010), Burkhardt (1997) y Futschek (2011) entre otros autores, es posible inferir que la necesidad de entender el problema, implica necesariamente identificar cada una de sus partes y extraer aquellas que se consideren de mayor relevancia, en conformidad con su contexto particular. Así, el aspecto clave en la abstracción es que permite ignorar aquellos elementos considerados superfluos o inaportantes a la solución, simplificando aquello que debe ser entendido de una manera puntual e inequívoca. Como lo expresa Koppelman (2010), la abstracción permite pensar en términos de conceptos y acciones en lugar de enfocarse en los detalles de la implementación.

El modelamiento surge entonces como una forma de plasmar las características extraídas del problema y articularlas de una manera significativa a la solución del problema. Tómese por ejemplo el hecho de diseñar un algoritmo partiendo de las acciones no atómicas que deban realizarse para su solución (como el caso de una función), lo cual llevaría a tener un algoritmo compuesto enteramente de funciones, de las cuales se conocen sus parámetros de entrada y su respectiva salida. En este caso, la abstracción lleva a comprender el funcionamiento del algoritmo sin necesidad de indagar en el detalle de su operación interna. Una conclusión temprana del análisis realizado anteriormente podría ser que al encontrarse dificultades con la identificación de los atributos del problema, no es posible establecer la manera en que dichos atributos participan del total, derivándose en una entropía del objetivo final de la solución planteada y su correspondencia con las necesidades que dan cimento al problema; de allí la necesidad de reforzar mediante estrategias metodológicas la capacidad del estudiante para enfrentarse al problema, identificar sus componentes y abstraer sus características más relevantes, y conforme lo expresa Hazzan (2008) siempre teniendo presente la manera en que cada una de sus partes participa de la obtención de un resultado final.

Además, si se considera que el análisis del problema debe derivar en la definición de un conjunto de soluciones, se tomará apartes de lo expresado por Libeskind-Hadas (2013) quien indica que las modificaciones curriculares en los cursos de algoritmia han provocado que los estudiantes pierdan cercanía con la derivación de los conceptos algorítmicos y sus aplicaciones, al redirigir su enfoque desde una perspectiva orientada al problema y su comportamiento (p.ej., un proceso de ordenamiento), hacia otra cuya principal preocupación es la aplicación de un paradigma en particular y el diseño robusto de la solución en sí (p.ej., la definición de clases y su representación en diagramas UML) entregando las estructuras pre-construidas al estudiante sin ahondar en su comportamiento o aplicabilidad en situaciones reales. De allí que el ideal de formación buscaría lograr que los estudiantes profundicen en el comportamiento de cualquier estructura algorítmica a través del análisis de sus requisitos y derivando su comportamiento, logrando de esta manera un interés real en abstraer las características de los problemas algorítmicos, comprendiendo su funcionamiento y definiendo estructuras de control que le soporten.

Hasta este punto, la construcción de la solución es un aspecto meramente estructural, pensando en las características del software y la manera en que este se apoyará en patrones de diseño o marcos de trabajo (Frameworks) durante su construcción, pero en pocas ocasiones existe un análisis concienzudo acerca de la idoneidad de una solución algorítmica aplicada a dicha solución.

Podemos extractar una segunda conclusión temprana si se toma lo escrito por Lahtinen (2005) y Vásquez (2012) quienes hablan acerca de la necesidad de definir diversas soluciones algorítmicas a un problema particular, de manera tal que sea posible validar cuál de ellas brinde un mayor nivel de satisfacción a través del análisis de su eficiencia. La eficiencia de un algoritmo se relaciona directamente con el uso de los recursos computacionales y no computacionales que demanda, así que es importante cuestionarse acerca de la manera en que un estudiante puede comparar dos estructuras algorítmicas con el fin de saber su eficiencia si desconoce la manera en que estas fueron construidas o si sus resultados son válidos.

Una vez identificado el abanico de posibilidades que satisface las necesidades de un problema a través de la definición de secuencias algorítmicas, es posible determinar que el punto de partida para su análisis sería la abstracción del problema y su solución. Quizás el modelo de abstracción más empleado en el análisis de un problema corresponde al uso de las analogías y las metáforas, tal y como lo describe Forisek (2012) dada su simplicidad y la cercanía con el lenguaje natural. Aunque de igual manera expresa que en ocasiones la ejemplificación seleccionada no necesariamente encaja en un modelo 1:1 con las características del problema, bien sea por que la metáfora no es aplicable en todos los contextos, posee un gran componente socio-cultural, o por que aplica solo a un conjunto reducido de casos, en cualquier caso evita ampliar las posibles aplicaciones de la solución e incrementa el riesgo que se llegue a conclusiones erróneas, tomándolas como válidas en consideración de la naturaleza del problema.

El proceso de verificación, conforme lo expresa Hartwig (2011) debe ligarse no solo a los resultados que son arrojados del proceso de transformación inherente al algoritmo, sino a la posibilidad de analizar la manera en que su comportamiento puede verse afectado bajo circunstancias específicas, alterando la idoneidad de la solución cuando es utilizada en conjuntos de datos con características particulares. De igual forma Lahtinen (2005) exhibe que las dificultades pueden provenir del desconocimiento en el manejo de los conjuntos de instrucciones reservadas o estructuras relacionadas con la sintaxis y la semántica del lenguaje de programación empleado para implementar dichos algoritmos.

Es así que a la luz del pensamiento computacional, los principios planteados exhortan al desarrollo de competencias fundamentales en la capacidad de observar el comportamiento del ecosistema que rodea al problema, identificando cada una de sus partes y describiendo su funcionalidad, siendo una premisa obligada el hecho que "*todo proceso de solución a problemas algorítmicos debe partir del entendimiento del problema, a través de la disección de sus características más relevantes y de la posibilidad de entender su interrelación mediante la abstracción*" Yadav (2011).

2.4. Aplicación de los principios en el desarrollo de competencias algorítmicas

2.4.1. Consideraciones iniciales

Obsérvese que cada uno de los principios del Pensamiento Computacional abordados con anterioridad hace parte de una dimensión fundamental en la formación del estudiante relacionada a la solución de problemas empleando algoritmos. Las problemáticas descritas podrían entonces ser afrontadas de manera eficiente al aplicar dichos principios en la academia, ya que su enfoque principal radica en el fortalecimiento de la capacidad de abordar sistemática y eficientemente la aplicación de conocimientos básicos adquiridos a la solución de problemas, aunque siendo cautelosos con lo planteado por Bischof (2011), Ala-Mutka (2004) y Lu (2009), quienes establecen que los grandes retos son de índole pedagógico, desprendiéndose de ellos la tarea de analizar hasta qué punto la programación es requerida realmente para desarrollar competencias en abstracción, solución de problemas o inclusive el uso de soluciones genéricas, pudiendo entenderse entonces que el desarrollo de las habilidades propias del pensamiento computacional deben ser independientes del logro en competencias en programación, y que solo hasta que sus aspectos fundamentales sean suficientemente claros, los estudiantes deberían enfrentar problemas relacionados con programación.

En este orden de ideas, si la programación es el punto de llegada en el proceso formativo del estudiante al interior de las ciencias computacionales, ¿entonces cuál sería el punto de partida? Tomando apartes de entre los planteamientos expresados por Lu (2009), Blackwell (1996), Burkhardt (1997) y Brusilovsky (1997) puede inferirse que en ausencia de la programación en su rol introductorio a las ciencias de la computación, sería importante considerar la construcción de

una simbología y un vocabulario que les permitan abstraer, identificar fuentes de información y construir modelos mentales que pueden ser desarrollados a su alrededor.

Así, sería básica la formación orientada al entendimiento de los fundamentos de los procesos computacionales, familiarizando al estudiante con los conceptos algorítmicos y la abstracción de las características del problema y no en la manera en que todo esto es implementado en un lenguaje de programación en particular. Es posible observar en el planteamiento de Lu (2009) algunos ejemplos del proceso de construcción del vocabulario y la terminología computacional como sería el caso de las secuencias de acciones, el manejo de estados, el uso de iteraciones, el análisis de eficiencia de un algoritmo, entre otros. Lo importante es su enfoque hacia la abstracción de las características de un problema, tal y como lo establecen los principios del pensamiento computacional.

2.4.2. Competencias algorítmicas y su desarrollo

Para el caso del desarrollo de competencias algorítmicas, debe partirse del hecho que cualquier solución software es soportada no solo en la construcción de algoritmos que funcionen, sino en que lo hagan de la manera más eficiente posible. Se espera entonces que los estudiantes logren niveles aceptables en el desarrollo de sus habilidades y competencias en esta área específica durante su estadía en la formación universitaria, al igual que el desarrollo de otras competencias consideradas transversales y de crecimiento personal. El desarrollo de dichas habilidades es consecuencia de la definición de objetivos formativos en las asignaturas relacionadas con la lógica y la algoritmia, granularizando los atributos deseables en el proceso de formación.

Una de las principales áreas de conocimiento definidas por la Fuerza de Trabajo Conjunta de la IEEE y la ACM (2013) en las profesiones relacionadas con la informática y la computación corresponde a la algoritmia y el manejo de la complejidad algorítmica. Así, la definición de las competencias obedece a la definición de sus atributos relacionados y de los diversos niveles de profundidad hasta donde se quiere llegar. Se empleará para ello una superposición de los aspectos propuestos por Bloom (ya sea en su propuesta inicial o en su versión revisada) y las características de formación establecidas por la Fuerza de Trabajo Conjunto en Ciencias de la Computación, infiriéndose un marco de referencia general para identificar aquellas competencias que son indispensables en la formación algorítmica, tomando como centro el desarrollo de los conceptos básicos requeridos por el estudiante frente a la teoría de algoritmos. Con este propósito es indispensable preguntarse ¿Qué debe saber el estudiante frente al concepto de algoritmo y su composición? La respuesta hace entrever que elementos como: instrucción, secuencia, ciclo, condicional, variable y operación serían entonces la base de cualquier desarrollo cognitivo aplicable a la solución de problemas.

Podría entonces obtenerse un primer acercamiento del estudiante al conocimiento de dichos conceptos a través de su observación, identificación y recordación, permitiéndole acercarse a ellos

mediante la repetición de sus características, la descripción de su funcionalidad y la relación que existe entre ellos. Así, su descripción inicial sería la presentada a continuación:

1. El estudiante reconoce/describe/identifica el concepto de instrucción y secuencia de instrucciones para la realización de un proceso enmarcado entre un inicio y un final.
2. El estudiante reconoce/describe/identifica el concepto de condicional para la evaluación de posibilidades al interior de una secuencia de instrucciones.
3. El estudiante reconoce/describe/identifica el concepto de ciclo para la realización de secuencias de pasos repetitivas al interior de una secuencia de instrucciones.
4. El estudiante reconoce/describe/identifica el concepto de variable y su papel al interior de una estructura algorítmica compuesta de una secuencia de instrucciones.
5. El estudiante reconoce/describe/identifica el concepto de operación como base de la transformación de datos al interior de una secuencia de instrucciones.

Sin embargo, no es suficiente con tener el conocimiento teórico y recordarlo textualmente para considerar que la competencia está desarrollada, es necesario interpretar la utilidad real de la teoría y la manera en que ésta se relaciona con otros elementos, debiendo preguntarse en este punto ¿De qué manera es evidenciable la comprensión del conocimiento adquirido por el estudiante? Podría entonces obtenerse un segundo acercamiento al entendimiento de dichos conocimientos así:

1. El estudiante interpreta/infiere/clasifica/compara/ejemplifica el comportamiento de un conjunto de instrucciones enmarcadas al interior de un inicio y un final.
2. El estudiante interpreta/infiere/clasifica/compara/ejemplifica el comportamiento de un condicional en consideración de la verificación realizada y las características de su secuencia de instrucciones.
3. El estudiante interpreta/infiere/clasifica/compara/ejemplifica el comportamiento de un ciclo en consideración de su número de repeticiones y las características de su secuencia de instrucciones.
4. El estudiante interpreta/infiere/clasifica/compara/ejemplifica el comportamiento de una variable en consideración de su uso y acceso a lo largo de la secuencia de instrucciones.
5. El estudiante interpreta/infiere/clasifica/compara/ejemplifica el comportamiento de una operación en consideración de las variables que emplea y las características de su secuencia de instrucciones.

En un nivel posterior al conocimiento teórico, su interpretación e interrelación con otros elementos, se encuentra su aplicación en nuevas situaciones, específicas y claramente identificadas, empleando para ello habilidades desarrolladas a partir de la transferencia de conocimientos básicos a situaciones repetitivas. En este punto es necesario preguntarse si el estudiante ¿conoce la manera de hacer uso del conocimiento adquirido? Podría entonces

obtenerse un tercer acercamiento a la definición de una competencia y la aplicación de los conocimientos así:

1. El estudiante implementa/usa/ejecuta un conjunto de instrucciones orientadas a la realización de un proceso que obtiene un resultado y se enmarca entre un inicio y un final.
2. El estudiante implementa/usa/ejecuta condicionales con el fin de evaluar el cumplimiento de condiciones al interior de una secuencia de instrucciones y validar su comportamiento.
3. El estudiante implementa/usa/ejecuta ciclos de instrucciones con el fin de estructurar secuencias repetitivas de instrucciones que le permiten alcanzar logros intermedios la interior de una secuencia de instrucciones.
4. El estudiante implementa/usa/ejecuta secuencias de instrucciones que hacen uso de variables con el fin de almacenar conjuntos de datos.
5. El estudiante implementa/usa/ejecuta operaciones sobre las variables que contienen conjuntos de datos con el fin de modificar su contenido a lo largo dela secuencia de instrucciones.

Hasta este punto, el estudiante ha podido aplicar sus conocimientos en situaciones específicas. Sin embargo, se espera que él evolucione hacia un nivel cognoscitivo superior en el que pueda descomponer el problema e interpretar sus características a partir del análisis del dominio del problema. Debe partirse entonces del hecho que un problema algorítmico (o parte de él) puede tener características similares a las de otros problemas en los cuales la aplicación de estructuras algorítmicas se llevó a cabo exitosamente. Es necesario preguntarse si ¿Es posible llevar las características del problema a un plano general que permita aplicar su solución a situaciones equiparables? es allí donde el análisis de las características del problema algorítmico deben fundamentarse en la generación de patrones, permitiendo obtener un cuarto acercamiento al análisis de la relación problema/solución y la aplicación del conocimiento adquirido así:

1. El estudiante compara/categoriza/estructura conjuntos de instrucciones empleadas en la solución de un problema algorítmico previo con otros similares, a partir de sus características más relevantes.
2. El estudiante compara/categoriza/estructura secuencias de instrucciones, condicionales y ciclos a la solución de problemas algorítmicos partiendo de su uso en soluciones previas a problemas de características similares.
3. El estudiante compara/categoriza/estructura variables que le permiten almacenar conjuntos de datos a partir de las características y soluciones planteadas a problemas algorítmicos similares.
4. El estudiante compara/categoriza/estructura conjuntos de operaciones que le permitan modificar el contenido de las variables definidas, en consideración de su comportamiento esperado en problemas de características similares.

Ahora, si se piensa que a partir del análisis del problema y la derivación de sus características más relevantes es posible obtener diversas soluciones factibles, quiere decir que se ha logrado un nivel de abstracción superior del problema, a partir del cual la estructuración de la solución algorítmica puede satisfacer no solo las necesidades puntuales del problema planteado, sino que podría eventualmente, hacerlo con problemáticas similares partiendo de una generalización de sus atributos. Es en este punto que el desarrollo de la competencia se enfoca en la posibilidad de innovar en la construcción de una solución con relación a la obtención de un resultado derivado de ésta, siendo las preguntas obligadas si ¿es posible definir más de una solución al problema? Y ¿de qué manera puede seleccionarse de entre las opciones la más adecuada? Aquí se obtiene un quinto acercamiento a la definición de competencias a través del análisis de impacto de sus soluciones así:

1. El estudiante adapta/prueba/desarrolla la secuencia de instrucciones definida en una estructura algorítmica, valorando su impacto y resultado.
2. El estudiante adapta/prueba/desarrolla el comportamiento y los resultados obtenidos a partir de un condicional, midiendo su impacto.
3. El estudiante adapta/prueba/desarrolla el comportamiento y los resultados obtenidos al interior de un ciclo, midiendo su impacto.
4. El estudiante adapta/prueba/desarrolla la definición de variables y su comportamiento a lo largo de la ejecución de la estructura algorítmica, valorando su impacto.
5. El estudiante adapta/prueba/desarrolla el comportamiento de las operaciones definidas, valorando su impacto.

Finalmente, una vez que el estudiante ha logrado construir soluciones a partir de la abstracción de las características del problema y generalizado sus atributos con el fin de establecer patrones de comportamiento, logra el nivel más alto de aplicación del conocimiento al valorar las calidades sistémicas de dicha solución, permitiéndole contrastarlas con un conjunto de valores esperados y determinar o no la necesidad de optimizarla, reemplazarla o dejarla como está. En este punto de desarrollo de la competencia se enfoca en la posibilidad de evaluar una solución y compararla con puntos de referencia establecidos, siendo sus preguntas obligadas: ¿es óptima la solución planteada? ¿De qué manera puede evaluarse su desempeño y resultados? Aquí se obtiene un sexto acercamiento a la definición de las competencias y la argumentación del resultado obtenido así:

1. El estudiante evalúa/califica/estima la secuencia de instrucciones definida en una estructura algorítmica, comparando sus resultados con puntos de referencia establecidos.
2. El estudiante evalúa/califica/estima el comportamiento y los resultados obtenidos a partir de un condicional, comparando sus resultados con puntos de referencia establecidos.
3. El estudiante evalúa/califica/estima el comportamiento y los resultados obtenidos al interior de un ciclo, comparando sus resultados con puntos de referencia establecidos.

4. El estudiante evalúa/califica/estima la definición de variables y su comportamiento a lo largo de la ejecución de la estructura algorítmica, comparando sus resultados con puntos de referencia establecidos.
5. El estudiante evalúa/califica/estima el comportamiento de las operaciones definidas, comparando sus resultados con puntos de referencia establecidos.

Es claro entonces que el desarrollo de las competencias algorítmicas en el estudiante debe lograrse a través de diferentes hitos de evolución, cada uno de los cuales debe brindar al estudiante la capacidad de aplicar de manera gradual el conocimiento, hasta apropiarlo de forma satisfactoria, empleando una estructuración taxonómica que facilite la clasificación y establecimiento de elementos interrelacionados. Sin embargo, a este nivel se plantean nuevos interrogantes relacionados con ¿Cuál sería la mejor manera de lograr el desarrollo de las competencias en los estudiantes? ¿Qué actividades deben ser realizadas para desarrollar dichas competencias? ¿Cómo preparar al estudiante para afrontar el desarrollo evolutivo de sus capacidades, evitando caer en las mismas prácticas que hasta ahora han sido aplicadas en los planes de estudio? Para intentar dar solución a cada una de ellas, se hace necesario establecer las premisas de aplicación de los principios del pensamiento computacional que faciliten el desarrollo de habilidades de razonamiento, lógica y abstracción.

Supóngase entonces necesidad de resolver un problema básico de programación: *"Generar la serie de números: 5, 10, 15, 20…hasta un número N dado por el usuario". E*n primera instancia el estudiante observa la manera en que el docente le orienta en la construcción de un algoritmo que satisfaga las necesidades del problema. De esta manera estaría en capacidad de **reconocer/describir/identificar** no solo las características del problema, sino el conjunto de instrucciones y la manera en que se emplearon en la solución. Posteriormente, el estudiante comenzará a aplicar dichas instrucciones en solucionar nuevos enfoques del mismo problema: *"Generar la serie 2, 4, 6, 8, 10… hasta un número N dado por el usuario"* la construcción de este nuevo algoritmo, de características similares al primero, es consecuencia que el estudiante haya logrado **interpretar/clasificar/comparar** las características del problema y el conjunto de instrucciones empleado, de forma que la variación en pequeñas porciones del problema facilita su adecuación. Así, al transformar la definición del problema en *"Generar la serie 1, 5, 3, 7, 5, 7, 9, 7… ,23"* es obligante a que el estudiante no solo **implemente/use/ejecute** las instrucciones requeridas, sino que deba **comparar/categorizar/estructurar** las características de la solución, de manera que la variabilidad en su enunciado corresponda a una variabilidad en su solución. Así, al enfrentar problemáticas en las que sus características básicas se transforman, como *"Generar y hallar la sumatoria de la serie 2!+4!+8!"* hace que el estudiante deba **adaptar/probar** los conjuntos de instrucciones aplicados en definiciones anteriores, pero con un nuevo enfoque de aplicación. Finalmente, el estudiante estaría en capacidad de **evaluar/calificar/estimar** la validez y consistencia del resultado obtenido a través del algoritmo, pudiendo **adaptar/probar** nuevos enfoques de ser necesario.

2.5. Conclusiones del capítulo

En este capítulo se realizó una revisión del estado actual en la enseñanza de algoritmos, de manera que fuese posible identificar i) los criterios considerados en las diferentes tendencias, ii) la forma en que buscan simplificar la interiorización de los conceptos algorítmicos por parte del estudiante, ii) La manera en que se transforman las prácticas de enseñanza-aprendizaje.

El conjunto de prácticas analizadas parte (en su mayoría) de la simplificación de estructuras de control y su superposición a elementos que los estudiantes reconozcan en su cotidianidad, permitiendo ligarlos de esta manera a su propia experiencia. De esta forma se logra que actividades como la interpretación de necesidades y el planteamiento de soluciones se conviertan en una derivación de sus vivencias. Del mismo modo, la posibilidad de llevar estas simplificaciones a ambientes de trabajo minimalistas o de gran impacto visual ha permitido que los estudiantes se instruyan en el uso de interfaces y entornos de desarrollo, restando impacto al momento de enfrentarse a ambientes por demás complejos, ligados normalmente al uso de lenguajes de programación de alto nivel.

El uso de ejercicios recreativos orientados al desarrollo de habilidades (lúdica) al valerse no solo de ejemplificaciones, sino de acciones y movimientos corporales (kinestésica) han sido factores determinantes al identificar no solo prácticas, sino enfoques de representación que permitan agrupar dichas prácticas conforme a la orientación impartida a los estudiantes para representar la solución a los problemas. De esta manera, dichos enfoques convergen en factores comunes que permiten sacar partido de metodologías de desarrollo del pensamiento, que si bien no fueron estructuradas para la enseñanza de algoritmos, pueden ser igualmente aprovechadas ya que se orientan al desarrollo de las competencias que son requeridas de manera transversal en la construcción de estructuras algorítmicas, como es el caso del pensamiento computacional.

La formación del estudiante en la solución de problemas depende en gran medida de la manera en que se le oriente en el desarrollo de sus competencias. En este capítulo pudo observarse que, contrario a lo que puede esperarse, el perfeccionamiento de las habilidades en programación no necesariamente apoya la formación del enfoque algorítmico, ya que las habilidades lógicas y de estructuración de secuencias de pasos deben ser independientes de la implementación en un lenguaje de programación particular.

El primer paso para introducir al estudiante en la lógica y la algoritmia debe ser la puesta en común y la interiorización de las diferentes estructuras de control, la manera y situaciones en que deben usarse y los resultados esperados de ellas, con el fin que pueda entender la forma en que son aplicadas en la solución de problemas, como base a un posterior proceso de programación.

Pero la querencia en la aplicación de conceptos algorítmicos requiere de una evolución de la madurez mental del estudiante, partiendo de la observación y la recordación de características generales que puedan ser dicientes de la estructura del problema, esto se facilita a través de la

repetición, y tiene como resultado el reconocimiento de las estructuras algorítmicas, las características del problema y la relación que se establece entre ellas.

El siguiente paso sería entonces la posibilidad de interpretar la relación causada, pudiendo extrapolarla a situaciones similares, bien sea por parte del problema o de comportamiento de las estructuras algorítmicas, pudiendo aplicarlas a nuevos enfoques problémicos que, si bien no son iguales a los planteados originalmente, guardan relación en su estructura general y permiten emplear el conocimiento adquirido. Una vez que el estudiante ha logrado la aplicación exitosa de las estructuras algorítmicas a la solución de problemas, son los cuestionamientos relacionados con cualidades de dicha solución lo que le preocupan, como bien pudo observarse en la clasificación de la evolución en el desarrollo de competencias algorítmicas.

Un factor clave en el desarrollo del trabajo consiste en la identificación de los principios del pensamiento computacional que son concurridos por los planteamientos de cada autor referenciado, seleccionándoles a través de la coincidencia entre los diversos enfoques y la importancia que les es dada. Además, se ha procurado facilitar la adhesión de otros que si bien, no fueron expuestos por Wing, sería igualmente importante considerarlos como parte del enfoque orientador del proyecto: **el desarrollo de competencias algorítmicas y de solución de problemas en los estudiantes**.

Así, el proceso de identificación y selección de los principios, tomó en cuenta los aportes de reconocidos autores, que hacen mención a prácticas y actividades concretas, aplicables al desarrollo de habilidades algorítmicas.

De esta manera, la abstracción de las características del problema sin lugar a dudas constituye el pilar del proceso analítico y de solución de problemáticas a través del desarrollo de estructuras algorítmicas, de forma tal que su entendimiento se soporta en la identificación de aspectos relevantes y la simbiosis existente entre ellos, partiendo del hecho que se haya logrado la capacidad de fragmentar las características del problema y se haya planteado un conjunto de escenarios de solución que, al ser evaluados, permitan seleccionar aquellos que se acoplen mejor a las necesidades planteadas en el problema, manteniendo siempre claridad acerca de la manera en que se relaciona e interactúan.

Puede verse entonces que los principios seleccionados en el desarrollo del capítulo guardaron semejanza con las habilidades esperadas por los estudiantes de diferentes niveles en las carreras relacionadas con las ciencias computacionales, tal y como pudo evidenciarse en el sustento dado por los diferentes autores citados, de tal forma que se esperaría que su aplicación en ambientes académicos a través del pensamiento computacional presente poco impacto.

3. Desarrollo del Framework propuesto

3.1. Framework de aplicación a los principios del Pensamiento Computacional

Hasta el momento ha sido posible observar planteamientos como los presentados por Voyiatzaki (2004), Serafini (2011) y Milne (2002), entre otros autores, con relación al proceso formativo y el desarrollo de competencias algorítmicas en el estudiante. Dichos planeamientos expresan que las competencias no se limitan a la simple construcción de código en un lenguaje de programación, sino que por el contrario, hacen entrever una mayor relación con la habilidad de estructurar conjuntos de instrucciones que brinden solución a un problema específico.

Es así que, partiendo del análisis preliminar de los principios del pensamiento computacional y su aplicabilidad al proceso de aprendizaje de algoritmos, se pretende estructurar un Framework que considere su articulación y enfoque hacia el logro de competencias algorítmicas. Dicho Framework dispondrá de manera simple un conjunto de actividades a realizar y articulará su interrelación, definiendo el impacto esperado y la manera en que apoyarán al desarrollo de las competencias algorítmicas.

En la actualidad, conforme puede inferirse de lo presentado por Allan en sus escritos de (1997) y (2010), las ciencias computacionales son percibidas como un ecosistema donde la programación es “demasiado complicada” o “poco accesible”. Bajo esta perspectiva, no es incoherente compendiar de lo planteado por Everis (2012) y Garner (2002) que muchos estudiantes encuentran en la algoritmia un punto de inflexión en el desarrollo de sus competencias, ya que les es esquiva la facilidad de aplicar los conocimientos necesarios para enfrentar problemas específicos mediante la lógica.

Adicionalmente, autores como Aktunc (2013), Ala-Mutka (2004), Allan (1997) y Bond (2007) esbozan diversos acercamientos a dicha problemática, especialmente en relación a los procesos de idealización y construcción algorítmica que han sido mencionados precedentemente. De esta forma, la estructura del Framework, al igual que su objetivo principal, estarían sustentados en brindar puntos de referencia basados en los principios del pensamiento computacional que permitan superar dichas dificultades y lograr las competencias algorítmicas en el estudiante, independientemente de su nivel de formación. Se tomará para su definición partes de lo definido por Lyster (2004) y se mezclará con el análisis de principios del pensamiento computacional que se realizó en el capítulo 6, orientándolos al logro de las competencias descritas en el capítulo 7.

3.2. Base de la propuesta

El Framework propuesto consta de 8 partes, cada una de las cuales es correspondiente en relación uno a uno (1:1) a los principios del pensamiento computacional que fueron extractados en el capítulo 5, y que dan respuesta a la primera pregunta de investigación planteada en el capítulo 2:

1. Abstracción, Conceptualización y Representación de los aspectos más relevantes del problema.
2. Identificación de los posibles escenarios de solución.
3. Evaluación de los diferentes escenarios de solución identificados.
4. Elección y representación de las características abstraídas a través de modelos.
5. División del problema en elementos atómicos y significativos.
6. Trazabilidad entre los elementos que hacen parte del problema y la solución.

Así, su estructura estará soportada en conceptos teóricos, siendo su principal ventaja la facilidad para abstraer las principales características del contexto en que será implementado, facilitando de esta manera su adecuación a las necesidades particulares que esboce cada individuo o institución con relación al aprendizaje algorítmico. Esta adecuación se logra a partir de la articulación de sus partes, abarcando el proceso de sustentación de las competencias algorítmicas requeridas en el estudiante, permitiendo hacer uso repetitivo de las estrategias y plataformas actuales de formación, acompañadas de nuevas herramientas que apoyen el proceso formativo.

Dado este contexto, es necesario considerar que la identificación de los principios del pensamiento computacional requiere de igual forma de un modelo de aplicación al proceso educativo, presentando una solución a la segunda pregunta de investigación planteada en el capítulo 2. En el caso de dicha propuesta, serán definidas un conjunto de actividades interrelacionadas que, apoyándose en herramientas computacionales (p.ej., SCRATCH, ALICE o LOGO) acerquen al estudiante al logro de sus metas de formación algorítmica.

3.2.1. Componentes del Framework

Los componentes del Framework se estructuraron de manera ascendente, considerando inicialmente aquellas tareas fundamentales para el entendimiento del problema, y avanzando de manera paulatina hasta lograr su traducción a conceptos algorítmicos. La articulación de sus componentes pretende guardar semejanza con la estructura ordinal de las actividades y fases fundamentales para el desarrollo de software planteadas en los libros de Ingeniería de Software de Sommerville (2005) y Pressman (2005), en las cuales se instituyen conjuntos de tareas requeridas para la construcción de software de calidad, además de lo postulado por Lyster (2004), quien inicia con una serie de cuestionamientos acerca del problema a resolver, y avanza con sus características, la representación correspondiente a la entidad problema-solución, finalizando con su construcción y verificación. Además, en concordancia con el desarrollo de competencias algorítmicas, planteado en el capítulo anterior, obsérvese el hecho de que parte del análisis del

problema, relacionado su solución con aquellos conceptos algorítmicos que le permitirían brindar una solución adecuada.

Su visión general se compone de cuatro grandes conjuntos, cada uno de los cuales permite sentar las bases de un proceso robusto de aprendizaje:

1. Actividades orientadas a la contextualización del problema y la identificación de sus partes (caracterización).
2. Actividades orientadas a la fragmentación del problema en segmentos relevantes e interrelacionadas (fragmentación).
3. Actividades orientadas a la aplicación de conceptos algorítmicos en la construcción de posibles soluciones (conceptualización).
4. Actividades orientadas a la verificación de las estructuras algorítmicas estipuladas como solución, permitiendo seleccionar aquellas que cumplan con características particulares (verificación).

La **Ilustración 32** presenta la estructura general del Framework, sus componentes y la manera en que se articulan:

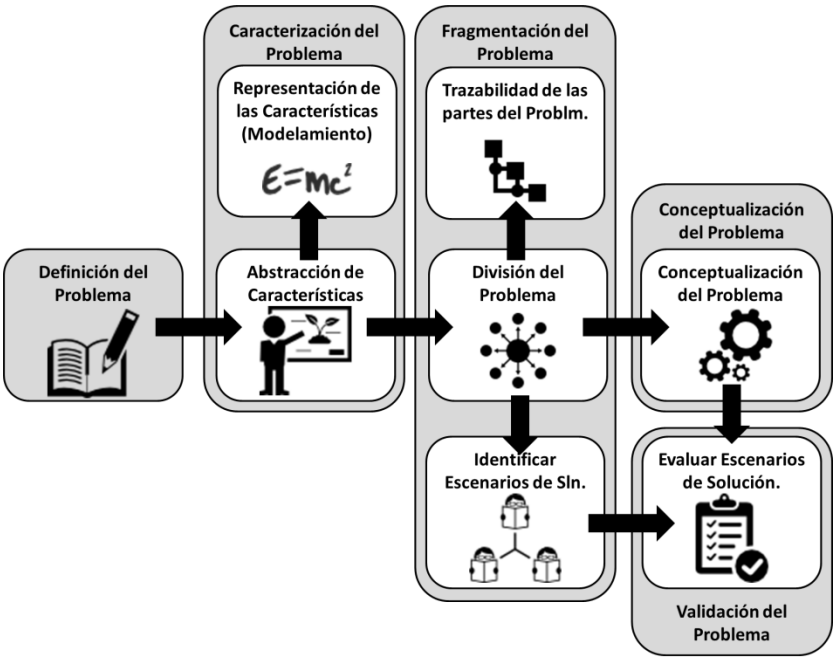


Ilustración 32. Estructura de Fases y Actividades del Framework propuesto. (Fuente: El Autor)

El Framework es entonces una estructura dinámica, donde el éxito de sus procesos guarda estrecha relación con una correcta **Definición del Problema**. Según lo expresado por Pressman (2005) es posible entender una "definición correcta" como aquella que contiene tanto las generalidades descriptivas del problema como el conjunto de datos necesario para darles soporte.

Sin embargo, es necesario aclarar que en conformidad a lo presentado por Burkhardt (1997) es posible que dicha definición se encuentre descrita en un texto proposicional soportado en lenguaje natural o en un modelo situacional o mental. Para el primer caso, el texto puede ser isomorfo y dependiente de la neutralidad de sus expresiones para poder ser apropiado correctamente por el lector, mientras que para el segundo caso, la descripción debe fundamentarse en representaciones independientes del contexto lingüístico que bien pueden ser del tipo narrativo o a través de representaciones visuales, con respecto a la situación descrita por el texto, pero en cualquier caso requieren de un esfuerzo mayor en la inferencia de las características del problema no presentadas de manera explícita.

Luego que el problema ha sido definido, delimitado e interiorizado, se encuentra la segunda fase: la **Caracterización del Problema**. Ésta se compone principalmente de dos actividades: *abstraer las características* principales del problema y *representarles a través de modelos* que faciliten su interpretación o visualización. Allí se tomará como cimento las exposiciones realizadas por Mow (2008), Selby (2012) y Brusilovsky (1997), quienes hablan de tomar el problema e identificar aquellos puntos fundamentalmente importantes en términos de: especificidad de su alcance, acciones necesarias para el logro de los objetivos y conjuntos de datos que permitan alimentar dichas acciones en pos de obtener la solución requerida. Para lograrlo de manera satisfactoria serán considerados los planteamientos de Forisek (2012) acerca del uso de metáforas para hacer visualmente más simples las características del problema, logrando así una aproximación sistemática en la medida que las características del problema se hacen más complejas, siempre observando la minimización de aquellas dificultades inherentes al ecosistema social, los aspectos culturales, y la imposibilidad de encontrar coincidencias exactas entre el ejemplo y el aspecto a ser explicado. De esta manera, la representación parte de las características abstraídas para construir modelos que reflejen las entidades más relevantes, conforme a lo dicho por Burkhardt (1997) al establecer que una de las mejores formas de enfrentar el problema es a través de la construcción de representaciones mentales que permitan modelar las características del problema y sus relaciones a través de los modelos de dominio y de programa, siendo coherente con lo planteado en el informe presentado por Duro (1992). En cualquiera de los casos sería importante tener presente los planteamientos de Hazzan (2008) al establecer que en ocasiones los estudiantes no logran en muchos casos un nivel de abstracción adecuado debido a que no poseen estructuras mentales o de conocimiento que les permitan partir de elementos símiles.

La fase de **Fragmentación del Problema** se compone principalmente de tres actividades: *División del problema*, *Identificar los escenarios de solución* y *establecer la trazabilidad entre las partes del problema*. El punto inicial sería entonces la fragmentación del problema en segmentos atómicas y representativos (2001) que posean significado en sí mismas como instancias más pequeñas del mismo tipo de problema, y cuyas soluciones combinadas permitan componer una gran solución al problema original. Según Brushan (2009), dicha combinación parte del hecho que en todo momento es clara la participación de cada fragmento, su aporte e importancia ante la construcción de la solución, pudiendo traducirse en una verificación temprana de la completitud

del modelo derivado de la definición del problema. Para la identificación de los diversos escenarios de solución se inferirá en lo expuesto por Eiter (2009), quien parte del hecho que, al fraccionar el problema macro en segmentos representativos de menor tamaño, es deseable la identificación de diversas estrategias de solución derivadas para cada uno de ellos, permitiendo tomar decisiones en cuento a su aplicabilidad de conformidad a las características del problema macro, es así que siguiendo continuando con lo proferido por Sedgewick (2011) al momento de ser combinadas nuevamente las soluciones, sus diversas combinaciones permiten la construcción de nuevas soluciones con enfoques diferentes y complementarios entre sí. Finalmente, es en este punto que la trazabilidad entra a jugar un papel de vital importancia, y conforme lo descrito por Kannenberg (2009) debe permitir no solo establecer la relación de proveniencia y consecuencia de los requisitos, sino que facilite además la identificación de su importancia, la manera en que ha sido entendido, implementado y finalmente, probado. Esto es especialmente complejo si se medita que el estudiante aún se encuentra en etapas tempranas de la solución al problema, y la utilización de herramientas computacionales que brinden soporte a la trazabilidad es nula, de tal suerte que deberá contemplarse la realización manual de dicho seguimiento, involucrando para ello grandes cantidades de esfuerzo y tiempo.

La fase de **Conceptualización del Problema** se compone de una sola actividad macro (del mismo nombre), de gran importancia durante el proceso, ya que es en este punto donde se recibe como entrada todo lo relacionado con el problema, los segmentos en los que fue fraccionado y sus respectivos enfoques de solución, con el fin de aplicar a cada uno de ellos los conceptos algorítmicos fundamentales que mejor le representen y construyendo una estructura lógica coherente que permita desarrollar y obtener una solución computacional. A este nivel se podría decir que es realizado un proceso de "traducción" de las especificidades del problema hacia modelos funcionales basados en la lógica, preludio a su implementación en un lenguaje de programación específico. Según Sedgewick (2011), el desarrollo algorítmico comprende un nivel de esfuerzo superior en la definición y el entendimiento del problema a ser resuelto, siendo gran parte de este esfuerzo, el requerido en la elección de las estructuras algorítmicas aplicables a la solución de un problema, ya que posterior a la descomposición, es posible que algunas de las estructuras sean fácilmente solucionables a partir de algoritmos simples, mientras que la elección de otros algoritmos requiera de mayor cuidado, ya que tienden a consumir gran cantidad de recursos (p.ej., búsquedas, ordenamientos y manejo de cadenas).

Finalmente, la estructura del Framework tiene un punto de cierre al llegar a la fase de **Verificación del Problema**. Ésta se compone también de una sola actividad macro: *Evaluar los escenarios de la solución*. En ella se toman cada uno de los escenarios de solución obtenidos previamente (representados en la subdivisión del problema) y el conjunto de representaciones algorítmicas que fueron derivadas de su estructura, analizando de manera concreta los resultados obtenidos durante su ejecución y los recursos que fueron empleados para lograr su objetivo.

3.2.2. Interacción de los elementos del Framework

Como pudo observarse, la idea básica en la estructuración del Framework no solo es la identificación de un conjunto de fases que permitieran la división de las tareas propias del pensamiento computacional, sino que cada una de estas fases (y actividades) se encargue de una pequeña parte del proceso general. Así, la separación de sus componentes permite que cada fase pueda ser abordada de forma independiente sin desconocer la importancia de su interacción. La actual vista del modelo hace evidente que la interacción de sus componentes se sustentaría en la entrega y recepción de información de una fase o actividad hacia otra, dándose comúnmente a partir de artefactos de software, aunque es posible que la construcción del algoritmo y su verificación pueda partir simplemente de una especificación del problema.

Entonces, la interacción no es en ningún momento una limitante para la reestructuración del modelo, y tampoco genera una dependencia que constituya una camisa de fuerza en su aplicación, debido a que es posible interactuar con el modelo a cualquier nivel, teniendo en cuenta el simple mantenimiento de la integridad requerida en cada componente o actividad. En este sentido, no es desacertado pensar en cada fase como en un componente o una función, donde se expone una interfaz al medio, empleándole para entregar y recibir la información requerida para el desarrollo de su actividad a modo de llamados y/o parámetros.

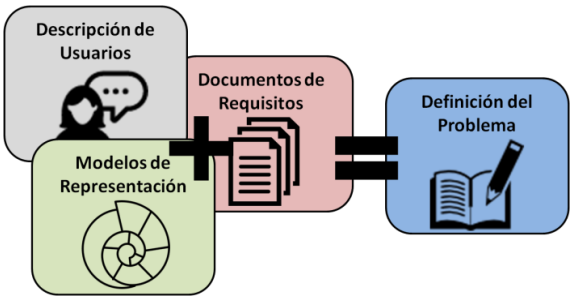


Ilustración 33. Componentes que dan origen a la Definición del Problema. (Fuente: El Autor)

La **Ilustración 33** muestra el punto de partida del modelo. Su entrada es toda documentación, descripción o modelo esquemático u procedimental que contenga la **Definición del Problema**, radicando su importancia en que esta brindará soporte a cualquier análisis o implementación en las etapas posteriores, y de conformidad con el planteamiento de Joyanes (2003), debe brindar claridad al contemplar aquello que deberá hacer el programa y los resultados que esperan obtenerse de él, permitiendo al estudiante interiorizar sus características en aras de desarrollar una solución adecuada. Aunque según Pressman (2005) debe tenerse especial cuidado con el hecho que esta etapa sea una de las menos estables en términos de acuerdos con las necesidades y la satisfacción del problema. De esta manera, el **Documento de Definición del Problema** deberá contemplar al menos los siguientes elementos como entrada de la etapa de acuerdo a Joyanes (2001):

1. ¿Cuál es la problemática general? ¿es entendible el propósito de brindarle solución?

2. ¿Qué espera obtenerse al final? ¿Hay forma de establecer los resultados de manera previa? ¿Existe más de una forma evidente de lograr la respuesta requerida?
3. ¿Qué información (datos o entradas) son necesarias para lograr una solución?
4. ¿Qué métodos (acciones, funciones, fórmulas, cálculos, procesos) permiten lograr una solución al problema empleando los valores especificados mediante transformaciones?

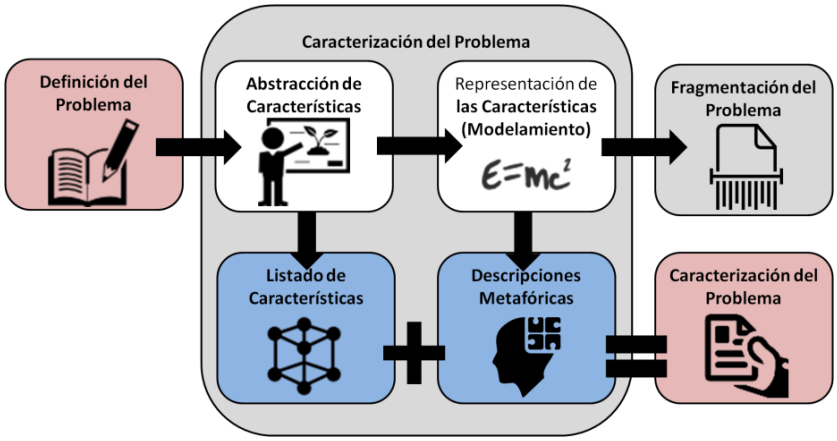


Ilustración 34. Entradas, Salidas y Actividades que conforman la Caracterización del Problema. (Fuente: El Autor)

Una vez sea clara la definición del problema, se espera que su entendimiento resida en la capacidad de resaltar o extraer de su definición aquellas características consideradas sobresalientes por medio de la abstracción de sus elementos y la posibilidad de describirlos de forma simple y comprensible. La **Ilustración 34** hace entrever que estas son actividades propias de la primera fase del Framework: la **Caracterización del Problema**. En conformidad y a lo expresado por Koppelman (2010) dichas actividades se orientan a eliminar detalles que si bien hacen parte del objeto de análisis, no serían considerados esenciales al momento de interpretar su esencia. Para lograrlo es necesario emplazarse en el planteamiento de Kramer (2007) con relación a una construcción documental que permita evidenciar las características del problema, dando respuesta a las siguientes preguntas, ligadas a la remoción de detalles y la generalización de sus características:

1. ¿Cuáles pueden ser retiradas sin que se pierda la esencia o sea posible entender el contexto del problema?
2. ¿Es posible tomar algunas de estas características y formularlas de manera general?
3. ¿dicha formulación permite abarcar otras características aplicables a diversidad de problemas o aún continúan reflejando las características de ese problema específico?
4. ¿existe la posibilidad de realizar una trasposición de dichas características a otro tipo de problemas?
5. ¿Existen símiles, metáforas o alegorías que permitan expresar de manera más simple el concepto que se desea analizar?

Así, la **Abstracción de Características** entregaría como resultado un documento que relacione el **Listado de Características** que enmarquen las generalidades del problema y la **Representación de Características** un conjunto de símiles o **Descripciones Metafóricas** que permitan entender su comportamiento o estructura a través de elementos más simples.

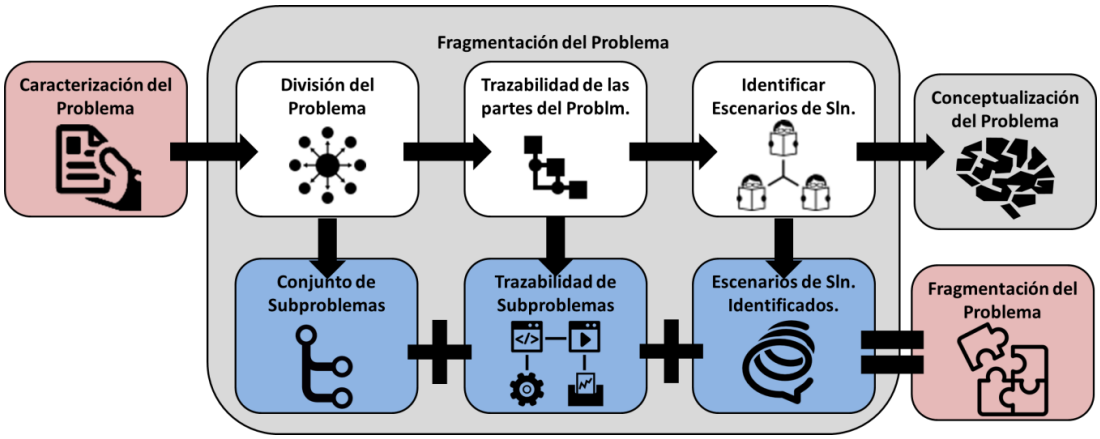


Ilustración 35. Entradas, Salidas y Actividades que conforman la Fragmentación del Problema. (Fuente: El Autor)

En este punto podría decirse que el proceso de simplificación resultante de la abstracción permite establecer el **QUE** compone el problema y **QUE** se espera obtener a través de su solución, así que partiendo de lo descrito por Joyanes (2001) el paso a seguir consistiría en entender la manera en que se llevará a cabo dicha solución, es decir el **COMO** se espera lograrlo a través de la simplificación en la complejidad del problema. Se espera entonces que el resultado de la **Caracterización del Problema** sea la base para realizar la **Fragmentación del Problema**. La **Ilustración 35** muestra que la fragmentación obedece entonces a una reducción sistémica del problema, tomando como punto originario sus características, y obteniendo un conjunto de sub-elementos que lleguen a ser, a suficiencia simples para ser resueltos mediante estructuras algorítmicas básicas. Así, se entiende que es posible acondicionar una o varias posibles soluciones empleando algoritmos a cada sub problema concebido.

Entonces, partiendo de lo definido por Cormen (2009) es necesario realizar una **División del Problema** de manera recursiva, de forma tal que se obtenga un conjunto finito de problemas de menor complejidad pero igualmente relacionados con el problema macro a solucionar. En este caso la recursividad hace referencia al hecho de subdividir el problema, obteniendo una colección heterogénea de enfoques problémicos complementarios, y no necesariamente al hecho que los subproblemas serán iguales en esencia al problema macro, simplemente se busca hacerlos más reducidos en su complejidad, buscando responder a las siguientes preguntas:

1. ¿Es el problema lo suficientemente pequeño para ser resuelto de manera directa?
2. ¿Qué tan simple es definir una solución al problema? ¿Es necesario considerar más de una variable al interior del problema para darle solución?

3. ¿Es posible dividir el problema? ¿Cuáles son las consideraciones que permiten orientar la división?
4. ¿Existe más de una forma de darle solución al problema?

La división del problema en enfoques heterogéneos hace necesario saber **QUE** partes lo componen, **CUAL** es su enfoque y por supuesto **COMO** se integran para dar solución al problema macro, dicha información se obtiene a través de realizar una labor de etiquetación de cada subproblema, logrando la **Trazabilidad de las partes del Problema**. En este respecto es posible inferir de los planteamientos realizados por Stepanian (2006), Gotel (1994) y Wieringa (1995) que la identificación de las partes es un proceso jerárquico, que permite identificar no solo los diferentes niveles de refinamiento de la problemática a partir de su origen, sino la manera en que aportan (individual o conjuntamente) a la solución del problema macro. Así, la trazabilidad obedece a la solución de las siguientes preguntas:

1. ¿Es posible tomar cualquier subproblema e identificar a que otros complementa?
2. ¿Es posible identificar a que segmento, parte o enfoque del problema macro pertenece?
3. ¿Es posible ligarlo a su justificación y a su explicación?
4. ¿Es posible establecer sus dependencias y a su vez quienes dependen de él?

Una vez se hayan logrado segmentos o subproblemas con complejidad lo suficientemente simple para ser resueltos de manera directa, es necesario **Identificar los Escenarios de Solución** para cada uno de estos segmentos. Cuando se habla de escenarios (en plural) quiere decir que para cada subproblema no existe necesariamente una única solución, sino que es posible encontrar diferentes conjuntos de pasos que permiten lograr un resultado satisfactorio.

Esta es una situación que requiere bastante cuidado por parte del estudiante, ya que si bien la construcción de diversas estructuras algorítmicas permiten validar los resultados obtenidos entre ellas, y según Pressman (2005) se realizaría de manera similar a como se llevaría a cabo una verificación por pares, es necesario entonces considerar algo de lo presentado por Bischof (2004) con relación a la sensibilidad en aspectos como el rendimiento y la precisión de los resultados obtenidos, medible a partir de los resultados obtenidos por las entradas y la secuencia de acciones empleadas en cada solución planteada. Entonces, la identificación y construcción de las diversas soluciones a un subproblema se hace posible al analizar de manera concienzuda las siguientes preguntas:

1. ¿Existe un conjunto de pasos que permita lograr la solución al problema planteado?
2. ¿Es posible plantear el logro de la misma solución a través de un conjunto de pasos diferente?
3. ¿Es posible validar las implicaciones de cada conjunto de pasos en términos de rendimiento?

Entonces, la **División del Problema** entregaría como resultado un documento que enumere el **Conjunto de Subproblemas** en los que se espera haber fraccionado la problemática general. Cada una de estas partes debe estar interconectada con las demás, y es la **Trazabilidad de las Partes del Problema** quien arroja un documento o esquema de **Trazabilidad de Subproblemas** que permita entender la manera en que se aporta a la solución del problema macro. Finalmente, **Identificar los Escenarios de Solución** permite acumular y clasificar soluciones aplicables, partiendo de entender su comportamiento y su ajuste a las necesidades propias de su integración con otras soluciones enumeradas en el documento de **Escenarios de Solución Identificados**.

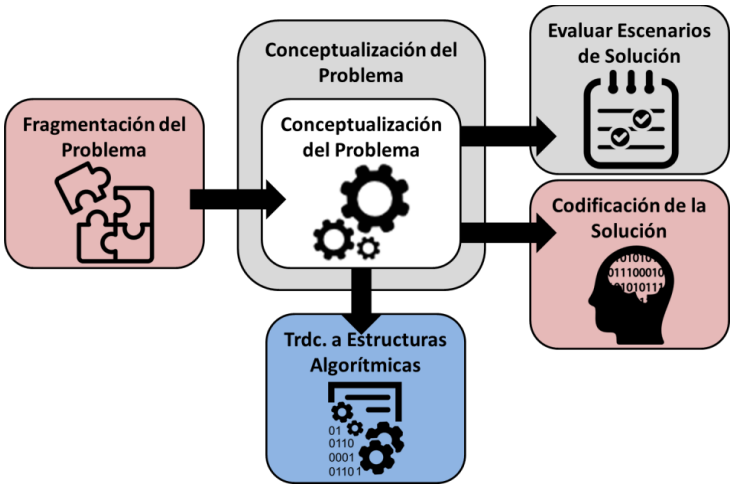


Ilustración 36. Entradas, Salidas y Actividades que conforman la Conceptualización del Problema. (Fuente: El Autor)

Posteriormente se procede a realizar la traducción de cada uno de los escenarios encontrados y su estructura narrativa a un conjunto de pasos estructurados algorítmicamente, valiéndose para ello de un lenguaje simple de representación (p.ej., pseudocódigo) o bien de un lenguaje con un nivel alto de especificidad (p.ej., instrucciones en C++).

La **Conceptualización del Problema** sería la tercera fase del Framework, y se fundamenta en apartes de lo descrito por Sedgewick (2011) quien describe al proceso de construcción del algoritmo como la definición de un conjunto de pasos que son susceptibles de implementarse en un lenguaje de programación particular, donde dicha implementación es solo una de las posibles formas de expresar el algoritmo. Así, la conceptualización parte el hecho que la división arrojada en la fase anterior debe permitir que la implementación de cada solución sea trivial, de tal manera que pueda lograrse en algunos casos una comparación uno a uno con las estructuras del lenguaje. Es allí donde se habla de realizar la **Traducción a Estructuras Algorítmicas** de cada solución encontrada. Según los lineamientos de Joyanes (2003) la traducción de cada estructura de solución encontrada se realiza a través del proceso de codificación, consistente en llevar cada una de las acciones planteadas a su equivalente en instrucciones (o palabras reservadas) propias de un lenguaje de programación específico. Para lograr una traducción es necesario analizar las siguientes acciones a realizar:

1. Identificar y declarar cada una de las funcionalidades que derivan de los subproblemas provenientes de la fase anterior
2. Identificar, declarar e inicializar para cada funcionalidad descrita las variables necesarias para su ejecución (p.ej., contadores, acumuladores, auxiliares, etc.)
3. Traducir las líneas de código simples, es decir, que no implique estructuras como condicionales, ciclos, selecciones, etc.
4. Identificar los segmentos comprendidos al interior de un condicional, delimitando su espacio (p.ej., con llaves), traducir la verificación de las condiciones y delimitando puntos de inicio y final de la condición simple (SI), anidada (SI… SI) y opuesta (SINO).
5. Identificar los segmentos comprendidos al interior de una selección, traduciendo la verificación de sus condiciones y delimitando cada una de las acciones al interior de sus opciones y su valor por defecto.
6. Identificar los segmentos comprendidos al interior de un ciclo, estableciendo claramente si se trabajará en términos de MIENTRAS (p.ej., while en C++), traduciendo la verificación de sus condiciones, estableciendo si se tiene cumplimiento de las condiciones previo a su ingreso al ciclo y que al interior del ciclo se realiza modificación de las condiciones, evitando ciclos infinitos.
7. Identificar los segmentos comprendidos al interior de un ciclo, estableciendo claramente si se trabajará en términos de HAGA MIENTRAS (p.ej., do while en C++), traduciendo la validación de sus condiciones, verificando que se debe ejecutar al menos una vez antes de ser validado, se tiene cumplimiento de las condiciones posteriores a su ingreso al ciclo y que al interior del ciclo se realiza modificación de las condiciones, evitando ciclos infinitos.
8. Identificar los segmentos comprendidos al interior de un ciclo, estableciendo claramente si se trabajará en términos de PARA (p.ej., for en C++), traduciendo la validación de sus condiciones, verificando el número de ejecuciones y el criterio de validación para romper el ciclo, evitando ciclos infinitos.

Así, el proceso de **Traducción a Estructuras Algorítmicas** entregará un conjunto de segmentos codificados que podrán ser interconectados para obtener una solución final adecuada a las necesidades del problema. Este proceso de interconexión y ensamblaje parte del análisis de características deseables y se entiende como una **Codificación de la Solución**.

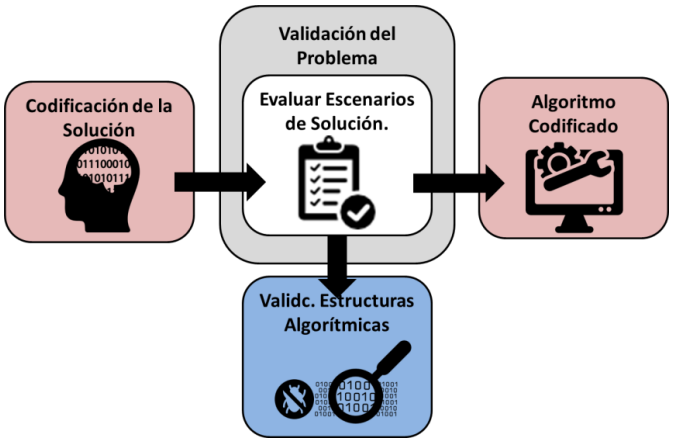


Ilustración 37. Entradas, Salidas y Actividades que conforman la Validación del Problema. (Fuente: El Autor)

Finalmente, todo segmento de código construido debe ser validado en su integridad, funcionamiento y resultados obtenidos. La **Ilustración 37** presenta la última fase del Framework: la **Verificación del Problema**. Según Joyanes (2001) la verificación de los algoritmos se realiza a partir de su ejecución sucesiva, sea de manera manual o automática a través de lo que Zhu (1997) entiende como pruebas de unidad o Unit Testing empleando para ello conjuntos de datos que permiten observar no solo los resultados obtenidos por el programa, sino las características que rodean el hallazgo de dicha solución (p.ej., tiempo de respuesta o consumo de recursos). En su interior, la **Verificación de Estructuras Algorítmicas** busca identificar y depurar errores que puedan derivarse de:

1. *La compilación*, como es el caso de aquellos errores que se originan en el uso de las palabras reservadas de un lenguaje de programación o en su estructuración
2. *La ejecución*, como es el caso del ingreso de datos no permitidos o fuera de los límites de un intervalo
3. *La lógica*, como pueden ser los recorridos de matriz por fuera de los límites, desbordamiento de memoria o uso dispar de tipos de dato en operaciones.

Cuando se realiza verificación del algoritmo construido para soportar el diseño funcional del software se habla de un tipo especial de pruebas conocido como “caja blanca” o “caja blanca estática” y que según Patton (2001) son responsabilidad primaria del desarrollador antes de dar el visto bueno de una aplicación. Esto obliga a la definición de afirmaciones que permiten validar no solo el comportamiento normal o esperado del algoritmo (es decir, que haga lo que tiene que hacer) sino el comportamiento no esperado (que le algoritmo no haga lo que no tiene que hacer), la manera en que el ambiente o entorno de trabajo debe estar preparado para su ejecución (a través de condiciones previas o “precondiciones”) y la manera en que este continuaría luego de esta (a través de estados finales, resultados o “postcondiciones”). Así, el proceso de verificación requiere de un seguimiento detallado al comportamiento de las estructuras algorítmicas al interior

de las pruebas de caja blanca o "pruebas de escritorio", y sus resultados permiten dar solución a los siguientes interrogantes:

1. ¿Se ha presentado errores de compilación? De ser así ¿es posible solucionarlos sin alterar la estructura lógica del algoritmo?
2. ¿Se han identificado claramente las precondiciones de ejecución del algoritmo? ¿es posible ejecutarlo una vez se cumplan a cabalidad? ¿es posible ejecutarlo sin su cabal cumplimiento?
3. ¿Se han identificado claramente las postcondiciones de ejecución del algoritmo?
4. ¿Se han identificado diversas maneras de lograr las postcondiciones? ¿Es posible identificar para cada una de estas opciones las características complementarias al resultado, como es el caso del tiempo de ejecución y el consumo de recursos? ¿Existen claramente definidos criterios que permitan seleccionar la respuesta más acorde a las necesidades del problema?
5. ¿Se obtiene una ejecución limpia (sin errores) toda vez que se ejecute el algoritmo con un conjunto de valores válidos? ¿se obtienen los resultados esperados en las postcondiciones?

El seguimiento, Como resultado de este proceso se obtiene una estructura algorítmica validada en su estructura, comportamiento y resultados, de manera que su aplicación a la solución del problema es ya un hecho.

Por otra parte, si se habla del desarrollo de las pruebas unitarias, si bien se cumplirían exactamente los mismos interrogantes planteados con anterioridad, su enfoque sería la necesidad de construir segmentos de código que permitan probar segmentos de código del programa, simplificando la labor de construcción y permitiendo asegurar que cada segmento de código funcional desarrollado se encuentra en condiciones óptimas.

3.2.3. Desarrollo de las Competencias Algorítmicas a través del Framework

Como pudo observarse, el Framework propuesto se compone principalmente de actividades que permiten orientar el trabajo de construcción de algoritmos a la vez que propende por el desarrollo de competencias y el reconocimiento de las estructuras algorítmicas básicas por parte del estudiante (instrucción, secuencia, ciclo, condicional, variable y operación).

Así, la estructura del Framework puede ser empleada de manera regresiva o descendente cuando se desea desarrollar las competencias más básicas (identificación, interpretación e implementación del concepto) mientras que se haría de forma progresiva o ascendente cuando lo que se desea es desarrollar las competencias más altas (comparación, adaptación y evaluación).

3.2.3.1. Uso regresivo del Framework

Emplear el Framework de manera regresiva implica acompañar al estudiante en la realización de las actividades que lo componen, de forma que le sea posible observar la manera en que se

resuelve un problema particular, tomando atenta nota del desarrollo de cada uno de los pasos a seguir. Esto no implica en ningún momento que las actividades se realicen en un orden diferente al establecido originalmente o que exista un análisis a través de ingeniería inversa, sino que la solución de problemas se realiza en acompañamiento al estudiante.

Tómese como punto de partida la siguiente definición del problema: *"Sea N un número natural cualquiera y C un número natural divisible exactamente por 5, generar una serie 5, 10, 15, 20, 25… C de tal manera que la serie contenga N números"*.

Al retomar la estructura definida para las competencias algorítmicas, sería necesario en primera instancia orientar al estudiante en identificar la manera de abordar dicho problema, presentándole finalmente la solución. La **Tabla 2** ilustraría la manera resumida de aplicación del Framework.

Tabla 2. Aplicación del Framework de forma regresiva en la solución de un problema. (Fuente: El Autor)

Fase	Actividad	Resultado
Caracterización del Problema	Abstracción de Características	• Número natural N >=0 representando número de términos • Número natural C >=0 representando número múltiplo de 5
	Representación de Características	• El número N es un contador que incrementa de uno en uno y permite establecer la cantidad de términos de una serie en particular • El número C es un contador que incrementa de cinco en cinco tantas veces como lo indique N
División del Problema	División del Problema	• Es necesario imprimir cada elemento que conforma la serie • Es necesario contar cada elemento que será impreso en la serie, hasta el total solicitado • Es necesario generar cada elemento que hace parte de la serie
	Trazabilidad de las Partes	• Es necesario generar el elemento que conforma la serie en primera instancia • Posterior a la generación, imprime el elemento generado • Finalmente, se cuenta el elemento y se compara con el total requerido
	Identificar Escenarios de Solución	• Para contar el elemento, se acumula el valor en una variable determinada en una estructura: contador = contador + 1 • Para generar el elemento, se acumula el valor en una variable determinada en una estructura: acumulador = acumulador + 5 • Para imprimir el elemento, se toma el resultado

		de la variable acumuladora y se presenta en una estructura: imprima(acumulador)
	Conceptualización del Problema (Pseudocódigo)	inicio escriba (indique el número de términos) lea (número de términos) contador ← 0 acumulador ← 0 mientras (contador < número de términos) haga acumulador = acumulador + 5 escriba (acumulador) escriba (espacio en blanco) contador = contador + 1 fin mientras fin

	Conceptualización del Problema (C++)	#include<iostream.h>
		int main (){
		int términos=0, contador=0, acumulador=0;
		cout<<("indique el número de términos");
		cin>>terminos;
		while(contador<terminos){
		acumulador+=5;
		cout<<acumulador<<” ”;
		contador++;
		}
		}

Verificación del Problema	Evaluar Escenarios de Solución (Pseudocódigo y C++)	inicio Número de términos: 5 contador 0 acumulador 0 contador < 5 ? (si) acumulador 5 → 5 contador 1 contador < 5 ? (si) acumulador 10 → 5 10 contador 2 contador < 5 ? (si) acumulador 15 → 5 10 15 contador 3 contador < 5 ? (si) acumulador 20 → 5 10 15 20 contador 4 contador < 5 ? (si) acumulador 25 → 5 10 15 20 25 contador 5 contador < 5 ? (no) fin

Al final, se espera que el estudiante pueda construir un algoritmo (en pseudocódigo o codificado en un lenguaje de alto nivel) similar estructuralmente al presentado en el segmento de conceptualización del problema. Obsérvese que en este ejemplo se presenta al estudiante la solución completa, de tal suerte que el estudiante solo debe **identificar** dos conceptos fundamentales: ***El proceso de generación de la serie depende del uso de un ciclo, el cual es empleado de igual manera para imprimir los elementos de la serie y contabilizarlos***.

Una vez que el estudiante haya interiorizado esta información, podrá evidenciar que cualquier serie se fundamenta en el mismo principio: el uso de ciclos. Así, un enunciado como el siguiente tendría la misma fundamentación: *"Generar la serie 1, 5, 3, 7, 5, 9, 7,… , 15"*. Sin embargo, no se hace evidente la necesidad de definir un punto de partida y controlar el incremento de la serie desde el enunciado como tal, sino que se convierte en una consecuencia de la interpretación de sus características, partiendo de su interpretación, comparación y adaptación del conocimiento adquirido. La **Tabla 3** presenta la manera de aplicación del Framework para este último enunciado problémico.

Tabla 3. Aplicación del Framework en un nivel de Interpretación del Concepto. (Fuente: El Autor)

Fase	Actividad	Resultado
Caracterización del Problema	Abstracción de Características	• Número natural S >0 representando los números de la serie • Número natural C >0 representando el contador de la serie
	Representación de Características	• El número S es un acumulador que incrementa en cuatro o en dos, dependiendo de si se trata de un incremento en ciclo par o impar y hasta un valor de 15 • El número C es un contador que incrementa de uno en uno tantas veces como se requiera y hasta que S alcance el valor de 15
División del Problema	División del Problema	• Es necesario imprimir cada elemento que conforma la serie • Es necesario contar las veces que se calculan los elementos hasta el total solicitado • Es necesario generar cada elemento que hace parte de la serie • Se finaliza cuando la serie alcanza el número 15
	Trazabilidad de las Partes	• Es necesario generar el elemento que conforma la serie en primera instancia • Posterior a la generación, imprime el elemento generado • Finalmente, se cuenta el elemento y se compara

<table>
<tr><td></td><td></td><td>con el total requerido</td></tr>
<tr><td></td><td>Identificar Escenarios de Solución</td><td>• Se parte de un primer elemento 1
• En los ciclos impares (1, 3, 5, 7…) se suma 4 al acumulador en una estructura: acumulador = acumulador + 4
• En los ciclos pares (2, 4, 6, 8…) se resta 2 al acumulador en una estructura: acumulador = acumulador - 2
• Los ciclos terminan cuando se alcance el valor 15 en el acumulador
• Para imprimir el elemento, se toma el resultado de la variable acumuladora y se presenta en una estructura: imprima(acumulador)
• Se valida el hecho que un ciclo sea par o impar tomando el contador, en una estructura par = contador / 2, si par = 0 entonces par, sino entonces impar</td></tr>
<tr><td></td><td>Conceptualización del Problema (Pseudocódigo)</td><td>inicio
contador ← 1
acumulador ← 1
mientras (acumulador < 15) haga
escriba (acumulador)
si (contador es par)
acumulador = acumulador - 2
sino
acumulador = acumulador + 4
contador = contador + 1
fin mientras
escriba (acumulador)
fin</td></tr>
<tr><td></td><td>Conceptualización del Problema (C++)</td><td>#include<iostream.h>
int main (){
int contador=1, acumulador=1;
while(contador<15){
cout<<acumulador<<” ”;
if(contador%2==0)
acumulador-=2;
else
acumulador+=4;
contador++;
}
cout<<acumulador<<” ”;
}</td></tr>
<tr><td>Verificación del Problema</td><td>Evaluar Escenarios de Solución (Pseudocódigo, Unit Test y C++)</td><td>inicio
contador 1
acumulador 1
acumulador < 15 ? (si)
→ 1
contador%2 == 0 ? (no)
acumulador 5</td></tr>
</table>

		contador 2 acumulador < 15 ? (si) → 1 5 contador%2 == 0 ? (si) acumulador 3 contador 3 acumulador < 15 ? (si) → 1 5 3 contador%2 == 0 ? (no) acumulador 7 contador 4 acumulador < 15 ? (si) → 1 5 3 7 contador%2 == 0 ? (si) acumulador 5 contador 5 acumulador < 15 ? (si) → 1 5 3 7 5 contador%2 == 0 ? (no) acumulador 9 contador 6 acumulador < 15 ? (si) → 1 5 3 7 5 9 contador%2 == 0 ? (si) acumulador 7 contador 7 acumulador < 15 ? (si) → 1 5 3 7 5 9 7 contador%2 == 0 ? (no) acumulador 11 contador 8 acumulador < 15 ? (si) → 1 5 3 7 5 9 7 11 contador%2 == 0 ? (si) acumulador 9 contador 9 acumulador < 15 ? (si) → 1 5 3 7 5 9 7 11 9 contador%2 == 0 ? (no) acumulador 13 contador 10 acumulador < 15 ? (si) → 1 5 3 7 5 9 7 11 9 13 contador%2 == 0 ? (si) acumulador 11 contador 11 acumulador < 15 ? (si) → 1 5 3 7 5 9 7 11 9 13 11 contador%2 == 0 ? (no) acumulador 15 contador 12 acumulador < 15 ? (no) → 1 5 3 7 5 9 7 11 9 13 11 15

		`fin`

Al igual que en el enunciado desarrollado en la **Tabla 2**, este último ejercicio lleva a que el estudiante reflexione acerca de la necesidad de definir **secuencias periódicas** que dependen de variables o condiciones al interior de los ciclos. Así, el estudiante debe interpretar que no solo se trata de un ***ciclo ordenado y periódico uniforme***, sino que ***la periodicidad y uniformidad condiciona su comportamiento***, bien sea ***en su incremento*** o ***su decremento*** en la generación de los elementos que lo componen, a la par que ***los imprime y contabiliza***. Al final, toda **implementación** que el estudiante realice alrededor de la generación de series se traduce en una apropiación del concepto y en la posibilidad de desarrollar competencias de nivel superior al permitirle compararlos otros más complejos.

3.2.3.2. Uso progresivo del Framework

Por otra parte, emplear el Framework de manera progresiva implica que el estudiante es quien realiza su propio análisis y desarrollo de la solución, y por tanto, ha sido acompañado de manera previa en la solución a problemas que le han orientado en el uso de las estructuras algorítmicas requeridas. En este punto el estudiante no solo está en capacidad de aplicar el concepto visto de manera directa, sino que puede comparar la estructura del problema y encontrar semejanzas con otros que haya resuelto anteriormente, adaptando su conocimiento a las nuevas características presentadas.

Analícese la siguiente definición de problema: *"Sea N un número natural par, hallar el resultado de la serie: 2! + 4! + 6! +8! + 10! + ... + N!"*.

Si observamos las características de los problemas resueltos anteriormente, el estudiante ya ha trabajado el concepto de ciclo y esto le permitiría entender que el cálculo del factorial de un número H (representado como **H!**) se realiza a través de la definición de un ciclo similar al de los dos ejemplos anteriores, al igual que identifica e interpreta el hecho que la sumatoria de los factoriales también corresponde a una estructura de tipo cíclico. Así, el estudiante parte de la comparación de las características del problema y estructura un nuevo enfoque adaptativo. La **Tabla 4** presenta la manera en que se abordaría la problemática por parte del estudiante a partir de la aplicación del Framework.

Tabla 4. Aplicación del Framework de forma progresiva en la solución de un problema. (Fuente: El Autor)

Fase	Actividad	Resultado
Caracterización del Problema	Abstracción de Características	• Número natural N >=0 representando términos a los que se calculará el factorial • Número natural N múltiplo de 2 • Número natural F >=0 representando el cálculo del factorial • Número natural S >=0 representando resultado de

		la sumatoria de los factoriales • Limitaciones en la implementación de números factoriales de gran tamaño (p.ej., capacidad máxima en arquitecturas de 8, 16, 32, y 64 bits)
	Representación de Características	• El número N es entregado por el usuario y debe ser par • El número F es un factorial que se fundamenta en el uso de un contador C para su cálculo • El número C es un contador que permite calcular el resultado de un factorial a través de un ciclo • El número S es un acumulador que recibe los resultados de los factoriales
División del Problema	División del Problema	• Es necesario calcular el resultado de cada factorial para un número par de la serie • Es necesario acumular el resultado de los factoriales calculados en cada iteración • Es necesario imprimir el resultado de la sumatoria una vez se haya completado el ciclo
	Trazabilidad de las Partes	• Es necesario generar el factorial del número F que conforma la serie en primera instancia • Posterior al factorial, se acumula en S su valor • Al finalizar se presenta en pantalla el valor final acumulado en S
	Identificar Escenarios de Solución	• Para calcular el factorial de un número, se acumula el valor en una variable determinada en una estructura basada en un contador, de la forma: contador = contador + 2 y factorial (N) = N * factorial (N- 1) • Para calcular la sumatoria de los factoriales, se acumula el valor en una variable en una estructura: sumatoria = sumatoria + factorial (n) • Para imprimir el elemento, se toma el resultado de la variable acumuladora y se presenta en una estructura: imprima(sumatoria)

	Conceptualización del Problema (Pseudocódigo)	`inicio`
		`escriba (indique el valor de cálculo)`
		`lea (valor de cálculo)`
		`si (valor de cálculo es par) haga`
		`contador sumatoria ← 0`
		`contador factorial ← 0`
		`sumatoria ← 0`
		`factorial ← 0`
		`mientras (contador sumatoria < valor de cálculo)`
		`mientras (contador factorial < valor par)`

		fin mientras fin mientras fin si sino escriba (valor ingresado no es par) fin sino contador ← 0 acumulador ← 0 mientras (contador < número de términos) haga acumulador = acumulador + 5 escriba (acumulador) escriba (espacio en blanco) contador = contador + 1 fin mientras fin

	Conceptualización del Problema (C++)	#include<iostream.h>
		int main (){
		int términos=0, contador=0, acumulador=0;
		cout<<("indique el número de términos");
		cin>>terminos;
		while(contador<terminos){
		acumulador+=5;
		cout<<acumulador<<” ”;
		contador++;
		}
		}

Verificación del Problema	Evaluar Escenarios de Solución (Pseudocódigo y C++)	inicio Número de términos: 5 contador 0 acumulador 0 contador < 5 ? (si) acumulador 5 → 5 contador 1 contador < 5 ? (si) acumulador 10 → 5 10 contador 2 contador < 5 ? (si) acumulador 15 → 5 10 15 contador 3 contador < 5 ? (si) acumulador 20 → 5 10 15 20 contador 4 contador < 5 ? (si) acumulador 25 → 5 10 15 20 25 contador 5 contador < 5 ? (no)

		`fin`

Así, una vez completado el desarrollo del ejercicio, el estudiante deberá estar en capacidad de construir un algoritmo similar estructuralmente al presentado en la tabla anterior. En la construcción de esta solución deberá no solo **aplicar** los conocimientos adquiridos en momentos anteriores (**identificación** conceptual, **comparación** con soluciones anteriores, **implementación** y **uso** de estructuras algorítmicas conocidas) sino enfocarse en adaptar dicho conocimiento a nuevos enfoques problémicos (**categorización** de las características del problema con otros similares, **adaptación** a las características del nuevo problema y **calificación** del comportamiento de las estructuras algorítmicas empleadas). Nuevamente, esta **implementación** es realizada por el estudiante alrededor de la generación de series, con la diferencia que pueden anidarse al interior de otras series, traduciéndose en una apropiación por más compleja del concepto.

3.3. Definición del entorno de aplicación del Framework

Debido a la estructuración de sus fases y actividades, el Framework propuesto no tiene dependencia directa con un nivel de formación específico, pudiendo ser aplicado tanto en los niveles formativos pertenecientes a la media vocacional, media técnica (décimo y once) o en los primeros semestres de formación universitaria (bien sea a nivel técnico, tecnológico o profesional). De esta forma es posible entrelazarlo con el currículo ordinario u orientarlo como una actividad complementaria a los cursos de formación básica en algoritmia. p.ej., fundamentos de programación, fundamentos de algoritmia o desarrollo del pensamiento analítico y sistémico.

En cualquier caso, es necesario observar que el desarrollo de competencias al que apunta el Framework debe ser correspondiente al de la actividad formativa (p.ej., curso, semillero) que hará uso de él. En general debe propenderse por que el estudiante logre un desarrollo básico de las competencias en resolución de problemas a través de la construcción de procedimientos algorítmicos que hacen uso de la lógica, realizando de igual manera una verificación de las construcciones realizadas.

Para emplear el Framework es necesario tener a mano ejercicios acompañados de su respectivo análisis y solución, estructurados conforme a las fases y actividades del Framework (como puede observarse en las tablas 2 a la 5). Para ello es importante tener en cuenta que si bien a lo largo del desarrollo del trabajo se relaciona un lenguaje de programación particular (C++) es posible emplear otros lenguajes de programación como JAVA o C#, inclusive pseudocódigo o herramientas de aprendizaje como LOGO, KAREL, ALICE, SCRATCH u otras similares, que empleen un lenguaje propio de estructuración de secuencias que, luego de un análisis estructural adecuado, pueda transfigurarse a un lenguaje de alto nivel que sea acorde a las necesidades de solución del problema.

Sin embargo, es importante considerar que los lenguajes o entornos de desarrollo empleados no tengan una dependencia muy alta de esquemas visuales de desarrollo, ya que tienden a desligar al estudiante de la necesidad de explorar y conocer la lógica del código a generar, desdibujándose la lógica que se pretende inculcar en la creación de estructuras algorítmicas simples o complejas.

Como puede observarse en la definición del Framework, el análisis de cada una de las fases está acompañado de una serie de interrogantes que deberán ser analizados de manera concienzuda, haciendo de su solución la manera más eficaz de interiorizar los objetivos que persigue cada una de sus fases.

En ese orden de ideas, aunque el Framework y su uso propenden a la formación de competencias, es importante considerar que debe existir en el estudiante unos saberes previos que permitan su aplicación, acompañada de una conciencia en las posibles limitaciones en su uso.

3.3.1. Requerimientos y restricciones para la aplicación

Si bien se espera que el estudiante desarrolle competencias algorítmicas desde su nivel más básico, es importante que cuente con algunos saberes previos que le permitan sacar mejor provecho de él.

Aspectos transversales como la comprensión de lectura, la capacidad de extractar ideas principales, de estructurar ordenadamente instrucciones, la capacidad de analizar las características del problema y el conocimiento básico de formulaciones o generación de expresiones simples son aspectos clave para lograr el entendimiento del problema y estructurar su definición en términos computacionales. De igual manera la capacidad de análisis y síntesis, acompañado de un dominio del lenguaje y la capacidad narrativa hacen posible que el estudiante pueda describir los aspectos generales más relevantes del problema, permitiéndole caracterizarlo de manera correcta. Finalmente, el análisis y la extracción de las características fundamentales del problema, la capacidad de abstracción y la facilidad de entender tanto el dominio de problema como el de la solución, hacen que para el estudiante sea viable desglosar y metaforizar el comportamiento del problema y de su respectiva solución, permitiéndole separar el problema y su respetiva solución.

Además, si bien podría decirse que el dominio de un lenguaje de programación es básico, en realidad podría considerarse irrelevante para el uso del Framework, ya que el desarrollo de las competencias algorítmicas no depende del dominio de un lenguaje de programación, y por el contrario se espera que las construcciones sean independientes de su uso.

Finalmente, es importante pensar en lo que no se espera lograr con el Framework en su forma actual, este sería el caso del desarrollo de competencias en estructuras de datos dinámicas (p.ej., uso de nodos en listas, pilas y colas), manejo de archivos, árboles, métodos de búsqueda, análisis asintótico de algoritmos y optimización de código serían algunos ejemplos. Podría pensarse en un trabajo futuro que permita ampliar los horizontes del Framework o simplemente definir tareas

independientes que permitan lograr estos y otros objetivos relacionados al proceso de construcción algorítmica.

3.3.2. Instrumentos a emplear

Se ha mencionado en diversas ocasiones que el Framework no se ha diseñado para coexistir con un lenguaje de programación particular, y que por el contrario, el uso de otro tipo de estrategias de aprendizaje le permitiría al estudiante desarrollar de manera menos restringida sus competencias algorítmicas.

Sin embargo, como pudo observarse en la exploración literaria, en la actualidad se presenta un creciente interés en el uso de herramientas educativas que, a través de la lúdica, permitan entender las premisas del comportamiento lógico de las cosas a través de la estructuración de algoritmos representados mediante segmentos de código simples, los cuales, en un entorno controlado, guardan similitud con un lenguaje de programación.

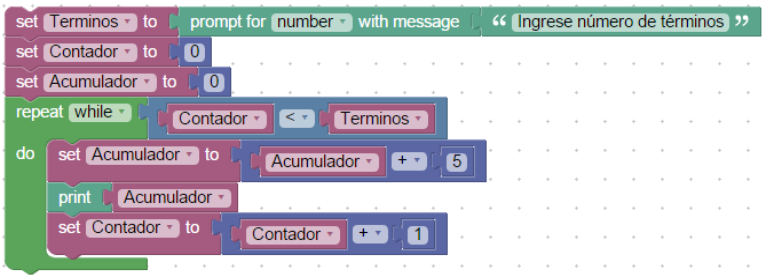
Luego de observar diversos entornos de programación que brinden estas facilidades de aprendizaje, se ha tomado como referentes para el desarrollo del Framework a dos de ellos: SCRATCH (diseñado por el Media Lab del MIT como una aplicación de escritorio) y BLOCKLY (diseñado por Google inc. como un editor web). Ambos emplean bloques visuales como representación de las instrucciones, estructuras, y composiciones funcionales, siendo su principal ventaja la posibilidad de brindar libertad al estudiante para estructurar prototipos cuyo comportamiento puede ser validado de manera visual e interactiva. Para el caso particular de este trabajo se emplearía BLOCKLY como entorno de aprendizaje, ya que no es necesario instalarlo en una máquina para disponer de él (como es el caso de SCRATCH), además de permitir la exportación del algoritmo diseñado a estructuras de control pertenecientes a lenguajes de programación como JavaScript o PHP.

El uso de BLOCKLY es intuitivo, permitiendo fragmentar el proceso de codificación en sus elementos atómicos: asignación de valores a variables, manejo de secuencias repetitivas, impresión de resultados, entre otros, están diferenciados por figuras similares a las de un rompecabezas, orientando al estudiante acerca de la manera en que estarán dispuestas al interior del algoritmo y cómo interactúan en su intercambio de información. Tómese por ejemplo la definición del problema descrito y analizado en la **Tabla 2**, allí se conceptualizó la solución al problema a través de una estructura de pseudocódigo cuya representación es comparada con su equivalente en BLOCKLY en la **Tabla 5**.

Tabla 5. Comparativa de la conceptualización de un problema en Pseudocódigo y en BLOCKLY. (Fuente: El Autor)

Definición del Problema Planteada.
"Sea N un número natural cualquiera y C un número natural divisible exactamente por 5, generar una serie 5, 10, 15, 20, 25… C de tal manera que la serie contenga N números"

Conceptualización en Pseudocódigo	Representación en BLOCKLY

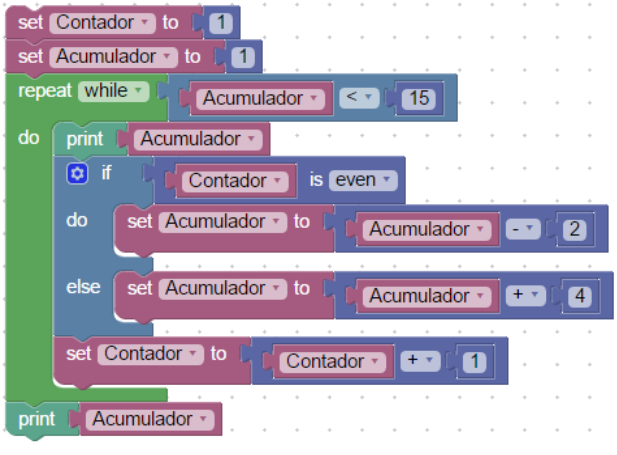
inicio
  escriba (indique el número de términos)
  lea (número de términos)
  contador ← 0
  acumulador ← 0
  mientras (contador < número de términos) haga
  acumulador = acumulador + 5
  escriba (acumulador)
  escriba (espacio en blanco)
  contador = contador + 1
  fin mientras
fin
	

Obsérvese que las acciones de solicitud de información al usuario (prompt) son claramente diferenciables de aquellas encargadas de asignar valores a una variable, y estas de igual manera lo son frente a aquellas estructuradas para contener el ciclo. Se ha coloreado la estructura de pseudocódigo de manera similar a la de los bloques, de forma que sea más fácil identificar las relaciones.

Realizando un análisis similar al anterior en la definición de problema planteado en la **Tabla 3**, es posible obtener una estructura algo más compleja en su construcción, pero igualmente simple en la manera de estructurarla y analizarla, como puede evidenciarse en la **Tabla 6**.

Tabla 6. Otra comparativa de la conceptualización de un problema en Pseudocódigo y en BLOCKLY. (Fuente: El Autor)

Definición del Problema Planteada.	
“Sea N un número natural cualquiera y C un número natural divisible exactamente por 5, generar una serie 5, 10, 15, 20, 25… C de tal manera que la serie contenga N números”	
Conceptualización en Pseudocódigo	**Representación en BLOCKLY**

inicio
  contador ← 1
  acumulador ← 1
  mientras (acumulador < 15) haga
  escriba (acumulador)
  si (contador es par)
  acumulador = acumulador - 2
  sino
  acumulador = acumulador + 4
  contador = contador + 1
  fin mientras
  escriba (acumulador)
fin
	

Es evidente en las comparaciones realizadas que la estructura de composición del algoritmo es equivalente, pudiendo traducirse en ambos sentidos sin mayor esfuerzo por parte del estudiante, facilitando enormemente la transferencia de conocimiento hacia el uso de un lenguaje de programación específico.

3.3.3. Resultados esperados

El uso de BLOCKLY como complemento al uso del Framework definido permitirá al estudiante acercarse de manera más intuitiva al proceso algorítmico de resolución de problemas, consintiendo de igual manera que los orientadores tengan puntos de análisis que les faciliten la transmisión del conocimiento requerido en la formación de las competencias algorítmicas, partiendo de la separación y diferenciación de cada una de las instrucciones en el logro de la solución del problema.

3.4. Aplicación del Framework al interior del proceso de articulación

La Alianza Futuro Digital Medellín ha realizado una inversión considerable en articular la educación media con la educación superior, y conforme a lo planteado al interior de la Alianza Futuro Digital Medellín (2012) dicha articulación se evidencia como la base de un proceso de profesionalización, brindando una alternativa de alto impacto en la construcción del proyecto de vida de los estudiantes, apoyándoles en el desarrollo de competencias tempranas en las áreas relacionadas con el desarrollo de software.

La articulación comienza con la identificación de módulos de los programas técnicos profesionales o tecnológicos que pueden ser cursados por los estudiantes en formación al interior de la educación media, y a su vez reconocidos al interior de los programas de formación en las instituciones de educación superior. El diseño de los currículos para los módulos inmersos en la articulación se fundamentó en la formación de competencias, las cuales corresponden a conocimientos que los estudiantes aplican no solo a la solución de problemas, sino a cada aspecto de su cotidianidad. La aplicación del Framework planteado en este trabajo permitiría dar un soporte extra al proceso de formación de los estudiantes, aplicándose bien sea de manera directa o a través de su adecuación.

Actualmente, el proceso de articulación cuenta con el acompañamiento del Politécnico Colombiano Jaime Isaza Cadavid, quien al interior del Área de Programas Informáticos y de Telecomunicaciones (APIT), adscrito a la Facultad de Ingenierías, ofrece el programa de **Técnica Profesional en Programación de Sistemas de Información**, cobijando algunas de las Instituciones Educativas adscritas a la Secretaría de Educación de Medellín.

3.4.1. Módulos involucrados en el proceso de articulación

A continuación se listan los módulos involucrados al interior del proceso de articulación en los grados décimo y once de las instituciones de educación media:

1. Para grado décimo:
    a. Identificación del Ciclo de Vida del Software.
    b. Desarrollo del Pensamiento Analítico y Sistémico 1.
    c. Construcción de elementos de Software 1.
    d. Construcción de Informes empleando Herramientas Ofimáticas.
2. Para grado once:
    a. Construcción de bases de datos 1.
    b. Interpretación de Requisitos.
    c. Construcción de elementos de Software web.

Cada uno de ellos realiza un aporte significativo al desarrollo de las competencias propias del perfil. Sin embargo, al revisar los formatos de programa para cada uno de ellos, pudo evidenciarse que el módulo que presenta mayor cercanía con la formación de competencias propias en la construcción de algoritmos es el de: **Desarrollo del Pensamiento Analítico y Sistémico 1** para ambos niveles.

Este módulo, conforme a su descripción programática[9] al interior del Politécnico Colombiano Jaime Isaza Cadavid (2011) pretende desarrollar en el estudiante *"las competencias que permiten adquirir y aplicar el razonamiento lógico, analítico, sistémico y algorítmico en la comprensión de problemas, identificación de sus diversas variables y definición de alternativas de solución que sean susceptibles y factibles de ser implementadas"* de tal suerte que guarda perfecta correspondencia con los planteamientos del Framework y su entorno de aplicación.

De igual forma, en el formato de contenido de la asignatura se presenta una clasificación que hace evidente el aporte de este módulo a las normas de competencia (COMP), tomando como punto de partida sus diferentes elementos de competencia (ELM) y los resultados de aprendizaje esperados:

1. (COMP) Analizar los requisitos del cliente para la construcción de una solución.
    a. (ELM) Construir un bosquejo del modelo funcional, acorde a la información recolectada y la metodología seleccionada.
        i. Identificar las variables o datos del caso de estudio o problema a resolver.
        ii. Reconocer las pautas, modelos y tendencias en problemas o situaciones similares o afines.
        iii. Descompone un problema o situación compleja en pequeñas partes o subproblemas, con el fin que sean más manejables.

[9] El contenido del módulo o asignatura se plasma en el formato FDP70 al interior del Politécnico Colombiano Jaime Isaza Cadavid.

iv. Analiza relaciones entre las partes de un problema o situación y establece relaciones causa-efecto sencillas.

2. (COMP) Diseña la solución de acuerdo a los requisitos del cliente.
    a. (ELM) Detalla la estructura técnica de acuerdo con un análisis de los requisitos de la solución a construir.
        i. Identifica y define los diferentes sistemas que hacen parte del sistema mayor.
        ii. Establece relaciones entre los diferentes subsistemas.
        iii. Define los elementos fundamentales que identifican o caracterizan un subsistema.
        iv. Construye un modelo que ilustra el sistema y sus correspondientes subsistemas.
3. (COMP) Desarrollar el sistema que cumpla con los requisitos de la solución informática.
    a. (ELM) Construir el software para el sistema de acuerdo a la metodología seleccionada.
        i. Identifica la secuencia de pasos lógicos o acciones a realizar.
        ii. Divide el problema en pequeños subproblemas.
        iii. Aplica adecuadamente las estructuras de control en la solución de problemas algorítmicos.
        iv. Construye soluciones a problemas algorítmicos empleando estructuras de datos estáticas.

Los resultados de aprendizaje esperados para dicho módulo comprenden:

1. La identificación de variables que conforman un problema con el propósito de generar una solución informática.
2. La aplicación de técnicas de desarrollo de algoritmos, orientadas a la solución de problemas reales o simulados, empleando para ello diversos paradigmas.
3. La selección de estructuras de control y decisión que permitan satisfacer los requisitos del problema propuesto.
4. El uso de estructuras de datos estáticas para el almacenamiento temporal de información.
5. La división de programas en módulos o subprogramas.
6. La definición y aplicación de conceptos fundamentados en el paradigma orientado a objetos.

Así, el desarrollo de los resultados 4, 5 y 6 debe ir de la mano con el uso de un lenguaje de programación específico, mientras el desarrollo de los resultados 1, 2 y 3 corresponde a una necesidad de base para alcanzar los demás, y que a pesar de todo, aún pueden ser independientes del uso del lenguaje.

Los demás módulos de grado 10 encaminan sus esfuerzos en el logro de otro tipo de competencias, más ligadas al desarrollo de soluciones a través del uso de un lenguaje de programación particular (p.ej., **Construcción de elementos de Software 1**), al entendimiento del proceso de desarrollo de software y sus partes (p.ej., **Identificación del Ciclo de Vida**) o al uso de herramientas de productividad para la elaboración de informes y presentaciones (p.ej., **Construcción de Informes empleando Herramientas Ofimáticas**).

Por su parte, todos los módulos de grado 11 se orientan a la construcción de soluciones software bajo un paradigma distinto: la web. Esto hace que en lugar de fortalecer el desarrollo de competencias algorítmicas, hagan uso de ellas para alcanzar sus resultados de aprendizaje esperados.

3.4.2. Aplicación del Framework en el contexto de la articulación

Es claro entones que existe la posibilidad de establecer una relación de uso directa entre el Framework y el módulo de **Desarrollo del Pensamiento Analítico y Sistémico 1**, ya que los resultados de aprendizaje de este último darían cuenta en el logro de competencias algorítmicas.

Sin embargo, para aplicar realmente el Framework sería necesario que dicho módulo se desligara de la utilización de un lenguaje de programación de alto nivel (p.ej., JAVA) en su proceso de enseñanza. Si se observa el programa de la asignatura al interior del Politécnico Colombiano Jaime Isaza Cadavid (2011) podría identificarse que en el nivel de desarrollo de las competencias provenientes de las normas adoptadas por el SENA para dicho módulo, se habla de implementar soluciones que no solo manipulen información, sino que se orienten al almacenamiento temporal en estructuras de datos estáticas. Esto se traduce en necesidades propias del manejo de lenguaje orientadas a:

1. Conocer el entorno de trabajo del lenguaje seleccionado.
2. Manejar los fundamentos del lenguaje de programación.
3. Codificar las estructuras de programación en el lenguaje.
4. Emplear estructuras modulares de código, haciéndolo reusable.
5. Aplicar estructuras de almacenamiento desde el lenguaje de programación.

Como resultado, gran parte del esfuerzo de la enseñanza por parte del docente y de aprendizaje por parte del estudiante se enfocan al lenguaje de programación, y este a su vez se convierte en el instrumento para la enseñanza de la algoritmia. Esto, conforme se sustenta en los planteamientos de Kölling (1999) y Ala-Mutka (2004) representa una limitante más que una ventaja, ya que la comprensión algorítmica debe desligarse del aprendizaje de una estructura o lenguaje de programación y orientarse más a la solución de problemáticas partiendo de la definición de secuencias de pasos.

Al desligar el proceso de aprendizaje del lenguaje y concentrarse solo en el aprendizaje y concienciación de las estructuras algorítmicas y su uso, sería posible que el estudiante aprendiera

primero a enfrentar y solucionar problemáticas mediante la definición de secuencias de pasos, disminuyendo la brecha que según Ala-Mutka (2005) se genera al no poder separar el conocimiento algorítmico del conocimiento del lenguaje, y que en algunos casos, dificulta la aplicación del conocimiento algorítmico en el aprendizaje de cualquier lenguaje de programación.

3.5. Conclusiones del capítulo

Como resultado del proceso de exploración y análisis de la información recopilada, se presentó un conjunto de actividades encaminadas a orientar la aplicación del pensamiento computacional. Estas actividades se encuentran ordenadas y agrupadas en fases, permitiendo de esta manera su aplicación paulatina y ascendente en complejidad, semejando la estructura ordinal de las actividades planteadas en diversos modelos de desarrollo de software, buscando que su adopción por parte de los estudiantes no entre en contravía a los saberes que pueden adquirir a lo largo de su proceso de formación, evidenciando que en realidad el desarrollo de software no corresponde a conjuntos de actividades aisladas (como podría entenderse, partiendo del enfoque modular de las asignaturas de un currículo) sino a actividades interrelacionadas, más como un sistema productivo.

El Framework se dividió en fases, cada una de las cuales guarda correspondencia con los principios del pensamiento computacional seleccionados como orientadores del proyecto. Sus fases poseen actividades internas que permiten su aplicación y encaminan al estudiante en el logro de las competencias algorítmicas planteadas en el capítulo 8, y su principal fortaleza corresponde al hecho que pueden ser adecuadas a las necesidades propias de cualquier institución y articuladas con sus procesos de enseñanza-aprendizaje.

La articulación sobre la cual se construye el Framework permite que este sea aplicado de dos maneras: progresiva y regresiva. La manera progresiva se orienta a desarrollar en el estudiante sus competencias más complejas o altas, mientras que su uso regresivo buscaría solo el desarrollo de las competencias más fundamentales o básicas. Esto quiere decir que cualquier problema algorítmico sería susceptible de analizarse a través del Framework, requiriendo de la orientación docente para aplicarse de una u otra manera, de manera independiente al nivel formativo o del currículo, brindando facilidades para su aplicación de manera directa en el currículo o como complemento a los cursos de algoritmia.

El Framework en si es solo una guía, una orientación de la manera de entender el proceso formativo como una tarea incremental, que puede ligarse o no a un entorno de programación específico, aunque se recomendaría emplear herramientas de aprendizaje simples, intuitivas y visuales como SCRATCH o BLOCKLY, en lugar de incursionar en lenguajes más complejos y abstractos como C++ o JAVA.

Como prerrequisito para la aplicación del Framework es necesario que el estudiante haya desarrollado previamente habilidades transversales, como es el caso de la comprensión lectora y la síntesis de ideas, ya que son competencias de formación básicas que no pueden ser cobijadas debido a su complejidad.

Para el caso particular de este proyecto, la implementación del Framework se soporta en el uso de BLOCKLY, ya que brinda facilidades al estudiante para comprender los conceptos básicos de la algoritmia y la manera en que se articulan al interior de una solución, empleando para ello algunas definiciones de problemas básicos planteados en capítulos anteriores.

Así, el proceso de formación fundamental en algoritmia al interior de la articulación solo se realiza en grado 10 con en el módulo **Desarrollo del Pensamiento Analítico y Sistémico 1**, ya que módulos como **Construcción de Elementos de Software 1** encaminan sus esfuerzos a la codificación de aplicaciones a través del uso de mecanismos de programación, e **Identificación del Ciclo de Vida** hacia el conocimiento del proceso de desarrollo. Mientas en grado 11 **Construcción de elementos de Software Web** encamina el aprendizaje hacia la construcción de sitios que interactúen con bases de datos. Esto implica que para grado 11 los estudiantes no profundizan en procesos algorítmicos y se limitan a la aplicación de aquellas estructuras algorítmicas que les fue enseñada en grado 10.

La aplicación del Framework en el proceso de articulación implicaría adherirlo al módulo **Desarrollo del Pensamiento Analítico y Sistémico 1** como parte integral de su estructura, sin embargo se insistiría en que no se trabajara con un lenguaje de programación, sino con un entorno de aprendizaje como BLOCKLY o SCRATCH, ya que de igual manera permitiría lograr los resultados de aprendizaje 1, 2, 3, y 4, considerados como básicos en el contexto algorítmico y de solución de problemas, mientras los resultados de aprendizaje 5 y 6 pueden ser trasladados al módulo de **Construcción de Elementos de Software 1**, lugar donde se apreciaría de manera especial su desarrollo al enfocarse directamente sobre un lenguaje de programación específico.

De esta manera es posible separar la formación algorítmica de la formación propia en un lenguaje de programación, atendiendo las inquietudes y sugerencias de diversos autores que hacen ahínco en la confusión que genera trabajar ambos conceptos de manera paralela, ya que el estudiante no logra diferenciarlas.

Esto no quiere decir que el Framework pierda validez ante los demás módulos, ya que puede ser aplicado de manera regresiva en módulos como **Construcción de Elementos de Software 1** y **Construcción de elementos de Software Web**, ya que el estudiante podría aprender mediante la identificación, interpretación e implementación de los conceptos de desarrollo sobre un lenguaje de programación particular, aunque requeriría de un esfuerzo mayor por parte de los docentes y estudiantes en el proceso de aprendizaje.

El presente capítulo permitió evidenciar que la aplicación del Framework podría realizarse de manera directa y con pocas intervenciones sobre algunos de los módulos de la articulación (como es el caso de **Desarrollo del Pensamiento Analítico y Sistémico 1**), mientras que para otros requeriría de un mayor esfuerzo en su adecuación total o parcial.

Una de las grandes dificultades en su implementación sería el hecho que solo un módulo se orienta al desarrollo de las competencias algorítmicas básicas, mientras que los demás hacen uso de las habilidades que allí se desarrollan. Esto plantea un interrogante acerca de si el conocimiento y el desarrollo de las habilidades algorítmicas se encuentra lo suficientemente maduro para adentrarse en el desarrollo de soluciones en módulos como **Construcción de Elementos de Software 1**, que se imparten al mismo nivel de **Desarrollo del Pensamiento Analítico y Sistémico 1**.

4. Consideraciones Finales y Conclusiones

4.1. Limitaciones del Framework

La propuesta de Framework presentada como resultado de este trabajo se orienta a establecer puntos de referencia que permitan desarrollar actividades formativas en estudiantes de ciencias computacionales, independientemente de su nivel formativo.

Sin embargo, en ningún caso consiste en una fórmula unívoca para el aprendizaje de los conceptos y el desarrollo de competencias algorítmicas, ya que dicho proceso se encentra rodeado de infinidad de elementos transversales, necesarios para que el estudiante pueda absorber y aplicar su conocimiento.

4.2. Trabajo Futuro

Con relación al proceso de aprendizaje de los conceptos y fundamentos algorítmicos, es necesario ampliar el campo de acción del framework propuesto a través de la transformación en los enfoques pedagógicos, permitiendo acercar el conocimiento a los estudiantes, no solo desde una perspectiva cercana a la matemática, sino como lo expresa Morales (2006) desde un enfoque más filosófico, permitiendo al estudiante relacionar los conceptos de secuencialidad, decisión y repetición con su aplicación en situaciones cotidianas reales.

Esto no quiere decir que se abandone el pensamiento matemático ligado a la resolución de problemas. Por el contrario, es necesario fortalecer el desarrollo de competencias matemáticas a través de la inclusión de actividades derivadas del desarrollo de competencias STEM, que de acuerdo a lo descrito por De Guzmán (2003), Gaulin (2001), González (2012) cobra especial importancia cuando se relaciona al hecho que los lenguajes, ambientes y estructuras de desarrollo evolucionan, haciéndose más ligeros en su estructura y abarcando cada vez más tareas, conforme lo expresa Wasserman (1982).

Con relación a los procesos de verificación y validación de las estructuras algorítmicas, es importante considerar la inclusión de pruebas unitarias y procesos de automatización de pruebas, que conforme a lo presentado por Rodríguez (2006) y Garbajosa (2009) pueden verse como el estandarte del proceso de verificación temprana del algoritmo desde su origen.

Otro de los enfoques de trabajo que puede considerarse a futuro sería el desarrollo de un ambiente de desarrollo especializado (como es el caso de SCRATCH o BLOCKLY para el pensamiento computacional) pero enfocado al uso particular del Framework propuesto, basado en estructuras algorítmicas similares a las empleadas en lenguajes de programación como C++ o java, permitiendo afianzar el desarrollo de las competencias. Esto permitiría la inclusión de

temáticas más densas como el manejo de algoritmos recursivos o adentrarse en el análisis de la complejidad y el orden de los algoritmos, temas de vital importancia al entender el comportamiento de los algoritmos y las soluciones planteadas.

Es importante considerar además la ampliación en el desarrollo de las competencias orientadas a estructuras de datos dinámicas, como es el caso de listas, pilas, colas y árboles, e incluir la creación de archivos o el manejo de datos de precisión, permitiendo abarcar módulos de formación que, si bien no hacen parte del proceso de articulación tratado en el proyecto, si fortalecerían el trabajo realizado en módulos como **Desarrollo del Pensamiento Analítico y Sistémico 2, Desarrollo del Pensamiento Analítico y Sistémico 3** e inclusive aquellos relacionados con **Desarrollo del Pensamiento Lógico y Matemático**.

Finalmente, incluir aspectos de programación en múltiples capas, tomando como punto de referencia la definición de estructuras algorítmicas que sustenten los niveles de persistencia y negocio, sin necesidad de adentrarse demasiado en el tema de diseño de interfaces gráficas, es un módulo complementario importante para la consolidación del Framework, fortaleciendo el trabajo en módulos como **Construcción de Elementos de Software 1**, **Construcción de Elementos de Software 2**, **Construcción de Elementos de Software 3** y **Construcción de Elementos de Software Web**.

4.3. Conclusiones

Una vez completado el desarrollo de este trabajo, es posible plantear las siguientes conclusiones:

- Tomando como punto de partida la exploración realizada a las diferentes metodologías de enseñanza y aprendizaje de algoritmos, no fue posible hallar una metodología única que permitiera garantizar altos niveles de aprendizaje en la totalidad de los estudiantes. Por el contrario, la riqueza en el ecosistema de los procesos formativos y los conocimientos previos de los estudiantes hicieron que debiera contemplarse un sinnúmero de opciones al momento de presentar los conceptos algorítmicos y su aplicación en la solución de problemas. Es dicha diversidad en las metodologías de enseñanza-aprendizaje lo que conlleva a la definición del presente Framework como factor unificador de sus principales fortalezas.
- Una de las grandes dificultades derivadas del proceso formativo de los estudiantes en la actualidad es la imposibilidad de relacionar conocimientos algorítmicos adquiridos con situaciones problémicas reales. Esto hace necesario que los ejemplos y problemas ligados a la ejercitación de los conceptos sea lo más cercana posible a la cotidianidad de los estudiantes, de tal manera que el proceso de transferencia de conocimiento sea mucho más simple.

- Otro de los aspectos fundamentales en el proceso de enseñanza-aprendizaje corresponde a la necesidad de emplear modelos y metáforas en la enseñanza de los conceptos algorítmicos, esto facilita que el estudiante pueda acceder a su base de conocimiento previo, asimilando el nuevo conocimiento de manera más rápida y efectiva.
- Los principios del Pensamiento Computacional identificados en este trabajo tienen relación estrecha con mejores prácticas de la industria ligadas al desarrollo de software, esto permitiría al estudiante facilidad al momento de continuar con su proceso formativo y se enfrente a conceptos más complejos como es el caso de la ingeniería de software y los modelos y ciclos de vida que sustentan el desarrollo de software, permitiéndole verlo como una actividad transversal y conjunta, en lugar de hacerlo como un conjunto separado y desarticulado.
- Las premisas del Pensamiento Computacional se fundamentan en el entendimiento de secuencias lógicas que no necesariamente fueron definidas para aprender a codificar, sin embargo su uso permite al estudiante tomar los conceptos algorítmicos y aplicarlos en su cotidianidad, facilitando de esta manera su percepción e interiorización en la solución de problemas.
- Es necesario realizar un trabajo intensivo en el fortalecimiento de la capacidad de análisis y solución de problemas de los estudiantes en su cotidianidad, ya que la aplicabilidad de los componentes del Framework depende directamente del logro evolutivo de la capacidad cognitiva del estudiante desde su entorno.
- Si bien las etapas del Framework se estructuraron de forma ordenada, considerando su uso de manera incremental en el proceso formativo de las competencias algorítmicas, es posible emplearlo de manera fragmentada, parcial o inversa. Esto brinda mayor flexibilidad y posibilidad de aplicarlo en contextos de características divergentes sin necesidad de realizar demasiados cambios, lo cual brinda facilidades en la conservación de su esencia.
- Es necesario ampliar la estructura del Framework a otros aspectos que hacen parte de la formación básica en algoritmia, como es el caso del manejo de

Bibliografía

ACM/IEEE-CS Joint Task Force. (2013). *Computer Science Curricula 2013.* Washington: ACM.

Aktunc, O. (2013). A teaching methodology for introductory programming courses using Alice. *International Journal of Modern Engineering Research*, 350-353.

Ala-Mutka, K. (2004). Problems in Learning and Teaching Programming. *Codewitz Needs Analysis*, 1-13.

Ala-Mutka, K. (2005). A study of the difficulties of novice programmers. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, 14-18.

Alianza Futuro Digital Medellín. (2012). *AFDM: Apuesta productiva y educativa en el sector software de Medellín.* Medellín: Créame.

Alvarado, T. (2000). La evolución del pensamiento de Hilary Putnam. *Revista del Instituto de Filosofía de la Pontificia Universidad Católica de Valparaíso*, 197-227.

Allan, V. (1997). Teaching Computer Science: A problem solving approach that works. *SIGCUE Outlook*, 2-10.

Allan, V. (2010). Computational Thinking in high school courses. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education*, 390-391.

Anderson, L. (2000). *A Taxonomy for Learning, Teaching and Assessing: A revision of Bloom's Taxonomy of Educational Objectives.* Washington: Pearson.

Baeza, R. (1997). Searching: An Algoritmic Tour. *Encyclopedia of Computer Science and Technology*, 10.

Bárcena, E. (2003). Los sistemas de enseñanza del inglés para fines específicos basados en el aprendiente. *Revista Iberoamericana de Educación a Distancia*, 41-54.

Barr, V. (2011). Bringing Computational Thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Transactions on Computational Logic*, 111-122.

Barrera, F. (2012). *Calidad de la Educación Básica y Media en Colombia: Diagnóstico y Propuestas.* Santafé de Bogotá: Universidad del Rosario.

Barriga, F. (2002). *Estrategias Docentes para un Aprendizaje Significativo, una estrategia constructivista.* Ciudad de México: McGraw-Hill.

Barriga, F. (2003). Cognición situada y estrategias para el aprendizaje significativo. *Revista Electrónica de Investigación Educativa*, 1-13.

Bell, T. (2010). *computer Science Unplugged, an enrichment and extension programme for primary-aged children.* Pittsburg: Carneige Mellon.

Bergin, J. (1996). *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming.* Phoenix: John Wiley and Sons.

Bergin, J. (2013). *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java.* Phoenix: Dreamsongs Press.

Bergin, J. (2013). *Monty Karel: A Gentle Introduction to the Art of Object-Oriented Programming in Python.* Phoenix: Dreamsongs Press.

Bischof, C. (2004). Algorithmic differentiation of different algorithms for the same problem: a case study. *Proceedings 18th European Simulation Multiconference*.

Bischof, E. (2011). Computer science in primary schools. Not possible, but necesary? *Informatics in Schools. Contributing to 21st Century Education*, 94-105.

Blackwell, A. (1996). Metacognitive theories of visual programming: what do we think we are doing? *IEEE Simposium on Visual Languages*, 240-246.

Bond, M. (2007). Tracking bad apples, Reporting the origin of null and undefined value errors. *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, 405-422.

Brandt, K. (2002). Using backtracking to solve theScramble squares puzzle. *Consortium for Computing Sciences in Colleges*, 21-27.

Brauner, P. (2010). The Effect of Tangible Artifacts, Gender and Subjective Technical Competence on Teaching Programming to Seventh Graders. *Teaching Fundamentals Concepts of Informatics*, 61-71.

Brennan, K. (1 de Septiembre de 2012). *Nuevas propuestas para estudiar y evaluar el desarrollo del pensamiento computacional.* Recuperado el 30 de 10 de 2013, de Eduteka: http://www.eduteka.org/modulos/9/284/2120/1

Brushan, B. (2009). *Software Engineering, Second Edition.* New Delhi: Firewal Media.

Brusilovsky, P. (1997). Mini-Languages: a way to learn programming principles. *Education and Information Technologies*, 65-83.

Bundy, A. (2007). Computationl Thinking is Pervasive. *Journal of Scientific and Practical Computing*, 67-69.

Burgess, C. (1994). Should Software Quality be a major issue when teaching first year programming to Software Engineers? *Transactions on Information and Communications Technologies*, 75-81.

Burkhardt, J.-M. (1997). Mental representations constructed by experts and novices in Object-Oriented program comprehension. *Human-Computer Interaction: Interact*, 339-346.

Caroll, L. (2003). *Alicia en el Pais de las Maravillas.* Buenos Aires: Ediciones del Sur.

Cernuda, A. (2004). Enseñanza por descubrimiento de los algoritmos de ordenación. *IX Jornadas de Enseñanza Universitaria de la Informática*, 7.

Consorcio de Habilidades Indispensables para el Siglo XXl. (1 de septiembre de 2007). *Eduteka.org.* Recuperado el 1 de agosto de 2013, de Logros indispensables para los estudiantes del siglo XXI: http://eduteka.org/SeisElementos.php

Consorcio de Habilidades Indispensables para el Siglo XXl. (1 de septiembre de 2007). *Eduteka.org.* Recuperado el 1 de agosto de 2013, de 21st Century Student Outcomes: http://eduteka.org/SeisElementos.php

Cormen, T. (2009). *Introduction to Algrithms, Third Edition.* Massachusetts: Massachusetts Institute of Technology.

Chernova, S. (2008). Teaching collaborative multi-robot tasks through demonstration. *8th IEEE-RAS International Conference on Humanoid Robots*, 385-390.

Chiarani, M. (2013). Las herramientas de la web 2.0 en el aula de programación. *Revista de tecnología de información y comunicación en educación*, 37-47.

Churches, A. (1 de Octubre de 2009). *Educational Origami.* Recuperado el 01 de 09 de 2013, de Educational Origami Blog: http://edorigami.wikispaces.com

Dagiené, V. (2011). Informatics education for new millenium learners. *Informatics in Schools. Contributing to 21st Century Education*, 9-20.

de Guzmán, M. (2003). *Cátedra Universidad Complutense de Madrid*. Recuperado el 1 de Junio de 2015, de Cátedra UCM Miguel de Guzmán:

http://www.mat.ucm.es/catedramdeguzman/drupal/migueldeguzman/legado/educacion/tendenciasInnovadoras

Denning, P. (1989). Computing as a Discipline. *Communications of the ACM*, 9-23.

Denning, P. (2005). Is Computer Science a Science? *Communications of the ACM*, 27-31.

Duro, A. (1992). *La coherencia textual en los modelos para la comprensión de textos, Informe basado en la tesis de Kintsch.* Madrid: ADM Consultores.

Eisenberg, M. (2010). Bead Games, or, Getting Started in Computational Thinking Without a Computer. *International Journal of Computers for Mathematical Learning*, 161-166.

Eiter, T. (2009). Finding Similar/Diverse Solutions in Answer Set Programming. *International Conference on Logic Programming, 2009*, 1-58.

Ertmer, P. (1993). Conductismo, Cognitivismo y Constructivismo: Una comparación de los aspectos críticos desde la perspectiva del Diseño de Instrucción. *Performance Improvement Quarterly*, 50-72.

Everis. (1 de Octubre de 2012). *En 2019 habrá en España un 40% menos de nuevos titulados en carreras y ciclos de grado superior TIC*. Recuperado el 1 de Diciembre de 2013, de sitio web de Everis Consulting: http://www.everis.com/spain/WCLibraryRepository/La%20falta%20de%20ingenieros.pdf

Fernández, R. (2006). SQUEAK, la herramienta que hará la revolución educativa. *Linux User*, 76-80.

Flores, A. (2011). Desarrollo del Pensamiento Computacional en la formación Matemática Discreta. *Lampsakos*, 29-33.

Forisek, M. (2012). Metaphors and analogies for teaching algorithms. *Proceedings of the 43rd ACM technical symposium on Computer Science Education* , 15-20.

Fuller, U. (2007). Developing a computer science-specific learning taxonomy. *ITiCSE-WGR '07 Working group reports on ITiCSE on Innovation and technology in computer science education*, 152-170.

Futschek, G. (2011). Learning Algorithmic Thinking with Tangible Objects Eases Transition to Computer Programming. *Informatics in Schools. Contributing to 21st Century Education*, 155-164.

Futschek, G. (2011). Learning algoritmic thinking with tangible objects eases transition. *Informatics in schools. Contributing to 21st century education*, 155-164.

Galdeano, M. O. (2002). *La Enseñanza de la Programación.* México D.F.: UPIICSA.

Garbajosa, J. (2009). Comparativa práctica de las pruebas en entornos tradicionales y ágiles. *Revista Española de Innovación, Calidad e Ingeniería del Software*, 19-32.

Garner, S. (2002). Reducing the Cognitive Load on Novice Programmers. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 578-583.

Gaulin, C. (2001). Tendencias actuales en la resolución de problemas. *SIGMA*, 51-63.

Gayo, D. (19 de Julio de 2013). *Daniel Gayo Website.* Recuperado el 1 de Agosto de 2013, de Daniel Gayo Website: http://di002.edv.uniovi.es/~dani/

Gonzalez, H. (2012). Science, Technology and Mathematics (STEM) Education. *Congressional Research Service*, 1-34.

Gotel, O. (1994). An analysis of the requirements traceability problem. *Proceedings of the First International Conference on Requirements Engineering*, 94-101.

Haden, P. (2003). The trouble with teaching programming. *Proceedings of the 16th annual NACCQ*, 63-70.

Hamada, M. (2010). Lego NXT as a Learning Tool . *ITiCSE '10: Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 321.

Harel, D. (2004). *Algorithmics, the Spirit of Computing.* Edinburgh: Addison Wesley.

Harel, D. (2004). *Algorithmics, the Spirit of Computing.* Edinburg: Addison-Wesley.

Hartwig, M. (2011). On the relationship between proof writing and programming: Some conclusions for teaching future software developers. *Communications in Computer and Information Science Volume 181*, 15-24.

Hazzan, O. (2008). Reflections on Teaching Abstraction and Other Soft Ideas. *ACM SIGCSE Bulletin*, 40-43.

Hromkovik, J. (2011). Why teaching informatics in schools is as important as teaching mathematics and natural sciences. *Informatics in Schools. Contributing to 21st Century Education*, 21-30.

Hvorecky, J. (1992). Karel, the Robot for PC. *Proceedings of the East–West Conference on Emerging Computer Technologies in Education*, 157-160.

ISTE, NSF & CSTA. (1 de Abril de 2012). *Definición operativa del Pensamiento Computacional para la Educación Escolar.* Recuperado el 30 de Octubre de 2013, de Eduteka: http://www.eduteka.org/modulos/9/272/2082/1

James, J. (16 de 12 de 2012). *10 skills for developers to focus on in 2013.* Recuperado el 1 de Agosto de 2013, de TechRepublic.com: http://www.techrepublic.com/blog/10things/10-skills-for-developers-to-focus-on-in-2013/3525

Joyanes, L. (2001). *Programación en C: Metodología, algoritmos y estructuras de datos.* Madrid: McGraw-Hill.

Joyanes, L. (2003). *Fundamentos de Programación, Algoritmos, Estructuras de Datos y Objetos.* Madrid: McGraw-Hill.

Kannenberg, A. (2009). Why Software Requirements Traceability Remains a Challenge. *Crosstalk, the journal of Defense Software Engineering*, 14-19.

Kölling, M. (1999). The problem of teaching object-oriented programming. Part 1: Languages. *Journal of Object-Oriented Programming*, 8-15.

Konopasek, M. (1985). The towers of Hanoi from a different viewpoint. *SIGPLAN Notices*, 39-46.

Koppelman, H. (2010). Teaching abstraction in introductory courses. *ITiCSE '10 Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, 174-178.

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM*, 36-42.

Lahtinen, E. (2005). A study of the difficulties of novice programmers. *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* , 14-18.

Lee, B. (6 de Noviembre de 2006). *Stanford News*. Recuperado el 30 de Agosto de 2015, de Stanford News: http://news.stanford.edu/news/2006/november8/vardi-110806.html

Libeskind-Hadas, R. (2013). A derivation-first approach to teaching algorithms. *SIGCSE '13 Proceeding of the 44th ACM technical symposium on Computer science education*, 573-578.

Loyarte, H. (2006). Desarrollo e implementación de un intérprete de pseudocódigo para la enseñanza de algoritmica computacional. *Primer congreso de tecnología en educación y educación en tecnología*, 63-70.

Lu, J. (2009). Thinking about Computational Thinking. *Proceeding SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education*, 260-264.

Lui, a. (2010). Facilitating independent learning with Lego Mindstorms robots. *ACM Inroads*, 49-53.

Lyster, R. (2004). A multi-national study of reading and tracking skills in novice programmers. *ITiCSE-WGR '04 Working group reports from ITiCSE on Innovation and technology in computer science education* , 119-150.

Mahnic, V. (1997). Some experience in teaching an introductory programming course using Oberon. *Transactions on Information and Communications Technologies*, 29-36.

Mahnic, V. (2003). Some experiences in teaching introductory programming at the faculty level. *World Transactions on Engineering and Technology Education*, 441-444.

Maniccam, S. (2012). Towers of Hanoi related problems. *Journal of Computing Sciences in Colleges*, 205-213.

Marron, A. (2012). A decentralized approach for programming interactive applications with JavaScript and blockly. *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions*, 59-70.

McManus, M. (1996). Teaching Collaborative Skills with a Group Leader Computer Tutor. *Education and Information Technologies*, 75-96.

McWhorter, W. (2009). Do LEGO Mindstorms Motivate Students in CS1. *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, 438-442.

Meinke, K. (2004). Automated black-box testing of functional correctness using function approximation. *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, 143-153.

Mendes, A. (2012). Increasing student commitment in introductory programming learning. *42nd ASEE/IEEE Frontiers in Education Conference*, 82-87.

Milne, I. (2002). Difficulties in Learning and Teaching Programming - Views of Students and Tutors. *Education and Information Technologies*, 55-66.

Minsker, S. (2008). Another brief recursion excursion to Hanoi. *SIGCSE Bulletin*, 35-37.

Misión de Educación Técnica, Tecnológica y de formación Profesional. (1999). *Hacia un sistema de oportunidades de formación para el trabajo.* Santafé de Bogotá D.C.: Cargraphics s.a.

Morales, Á. (2006). Una experiencia pedagógica para programas de ingeniería: La enseñanza de algoritmos mediada por la lógica cognitiva y la elaboración de juegos de lógica. *Educación en Ingeniería*, 26-32.

Moschovakis, Y. N. (2001). What is an Algorithm. *Springer* , 919-936.

Mow, C. (2008). Issues and Difficulties in Teaching Novice Computer Programming. *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, 199-204.

Navarridas, F. (2002). La evaluación del aprendizaje y su influencia en el comportamiento estratégico del estudiante universitario. *Contextos educativos: Revista de educación*, 141-156.

Observatorio de la Universidad Colombiana. (4 de enero de 2013). *Observatorio de la Universidad Colombiana.* Recuperado el 1 de agosto de 2013, de La formación del licenciado en educación, en cualquier área del saber, tiene como objetivo la formación de maestros: http://universidad.edu.co/index.php?option=com_content&view=article&id=1247:la-formacion-del-licenciado-en-educacion-en-cualquier-area-del-saber-tiene-como-objetivo-la-formacion-de-maestros&catid=36:ensayos-acadcos&Itemid=81

Ohland, M. (2014). Understanding Migration Patterns of Engineering Undergraduates: The Impact of Course Grades on Student Major Choice. *Frontiers in Education Conference Proceedings*, 65-70.

Papert, S. (1980). *Mindstorms,Children, Computers and Powerful Ideas.* New York: Basic Books.

Pattis, R. (1995). *Karel The Robot: A Gentle Introduction to the Art of Programming.* Phoenix: John Wiley & Sons.

Patton, R. (2001). *Software Testing.* Indianápolis: SAMS Publishing.

Pérez, I. (2008). Estudio sobre la problemática en los enunciados de los problemas de programación. *Articulación de Saberes, Memorias Arbitradas de Congresos*, 1-6.

Pérez, S. (2013). Abandono y egresos en las carreras de ingeniería de la universidad nacional de La Matanza. *Tercera Conferencia Latinoamericana sobre el Abandono en la Educación Superior*, 1-11.

Perrenoud, P. (2000). Aprender en la escuela a través de proyectos, ¿por que? ¿cómo? *Revista de tecnología educativa*, 311-321.

POLIJIC, P. C. (1 de Julio de 2011). Programa de la asignatura. *Programa de la Asignatura, Desarrollo del pensamiento analítico y sistémico 1*. Medellín, Antioquia, Colombia: Alianza Futuro Digital Medellín.

Polya, G. (2004). *How to Solve It: A New Aspect of Mathematical Method.* New Jersey: Princeton University Press.

Popescu, A. (30 de Abril de 2013). *mashable.com.* Recuperado el 1 de Agosto de 2013, de Coding Is the Must-Have Job Skill of the Future: http://mashable.com/2013/04/30/job-skill-future-coding/

Prensky, M. (2001). *Digital Game-Based Learning.* New York: McGraw-Hill.

Pressman, R. (2005). *Ingeniería de Software, Un Enfoque Práctico.* New York: McGraw Hill.

Real Academia de la Lengua Española. (1 de Marzo de 2013). *Diccionario de la Lengua Española, Vigésima segunda edición*. Recuperado el 01 de 07 de 2013, de Diccionario de la Lengua Española, Vigésima segunda edición: www.rae.es

Resnick, M. (18 de Febrero de 2011). *Grandes ideas subyacentes en Scratch.* Recuperado el 30 de 10 de 2013, de Eduteka: http://www.eduteka.org/modulos/9/284/1206/1

Roberts, E. (1 de Septiembre de 2005). Karel the robot learns JAVA. Stanford, California, Estados Unidos.

Rodríguez, A. (01 de 08 de 2008). *Sitio Web Docente, Ingeniero Alejandro Rodríguez Aguayo.* Recuperado el 01 de 08 de 2013, de Sitio Web Docente, Ingeniero Alejandro Rodríguez Aguayo: http://dcb.fi-c.unam.mx/users/alejandromra/

Rodriguez, J. (2006). Pruebas unitarias. *continuum.cl*, 1-9.

Rosas, M. (2014). Estrategia metodológica B-Learning para la enseñanza de la programación a los alumnos de primer año de Ingeniería Electrónica. *Noveno congreso de tecnología en educación y educación en tecnología*, 30-37.

Sedgewick, R. (2011). *Algorithms.* Boston: Addison - Wesley.

Selby, C. (2012). Promoting Computational Thinking with Programming. *WiPSCE '12 Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, 74-77.

Serafini, G. (2011). Teaching Programming at Primary Schools: Visions, Experiences, and Long-Term Research Prospects. *Informatics in Schools. Contributing to 21st Century Education*, 143-154.

Serna, E. (2011). La abstracción como componente crítico de la formación en Ciencias Computacionales. *Avances en Informática*, 79-83.

Shabanah, S. (2009). Simplifying Algorithm Learning Using Serious Games. *Proceedings of the 14th Western Canadian Conference on Computing Education* , 34-41.

Shen, A. (2010). *Algorithms and Programming, Problems and Solutions.* New york: Springer.

Simari, G. (2013). Los fundamentos computacionales como parte de las ciencias básicas en las terminales de la disciplina informática. *Memorias del VIII Congreso de Tecnología en Educación y Educación en Tecnología*.

Sommerville, I. (2005). *Ingeniería del Software.* Madrid: Pearson.

Stadel, M. (1984). Another nonrecursive algorithm for the towers of Hanoi. *SIGPLAN Notices*, 34-36.

Stepanian, L. (2006). *Solving the requirements traceability problem: A comparison of two pre-requirements specification traceability enablers.* Toronto: technical report CSC2106.

Stucki, D. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. *SIGCSE '00 Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, 75-79.

Swan, K. (1983). The U.S. M-16 rifle versus the Russian AK-47 rifle. A comparison of terminal ballistics. *U.S. National Library of Medicine*, 472-476.

Swan, K. (1991). Principles of ballistics applicable to the treatment of gunshot wounds. *U.S. National Library of Medicine*, 221-239.

Tamayo, M. (2004). *El proceso de la investigación científica, cuarta edición.* México: Limusa.

Toledo, F. (2002). *Fundamentos de Informática y Programación .* Valencia: Universidad de Valencia.

Vásquez, B. (2012). *Análisis y Diseño de Algoritmos.* México D.F.: Red Tercer Milenio S.C.

Voyiatzaki, E. (2004). Teaching Algorithms in Secondary Education: A Collaborative Approach. *World Conference on Educational Multimedia, Hypermedia and Telecommunications*, 2781-2789.

Wasserman, A. (1982). The future of programming. *Communications of the ACM*, 196-206.

Weisstein, E. W. (1 de Enero de 2013). *Wolfram MathWorld*. Recuperado el 1 de Julio de 2013, de Wolfram MathWorld: http://mathworld.wolfram.com/Algorithm.html

Wiener, A. R. (1945). The Roles of Models in Science. *Philosophy of Science*, 6.

Wieringa, R. (1995). *An introduction to requirements traceability.*

Wieringa, R. (1995). *An Introduction to Requirements Traceability.* Amsterdam: Free University, Faculty of Mathematics and Computer Science.

Wing, J. (2006). Computational Thinking. *Communications of the ACM*, 33-35.

Wing, J. (30 de 10 de 2013). *Jannette M Wing.* Recuperado el 30 de 10 de 2013, de Sitio web de Jannette Wing: http://www.cs.cmu.edu/~wing/

Yadav, A. (2011). Introducing computational thinking in edeucation courses. *Proceedings of the 42nd ACM technical symposium on Computer science education* , 465-470.

Zhu, H. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR)*, 366-427.

Zsakó, L. (2012). ICT Competences: Algorithmic Thinking. *Acta Didactica Napocensia*, 49-58.