Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Componente gráfico para la depuración, completamiento y edición de código para el framework Kivy

Autor: Luis Angel Rodríguez Martínez

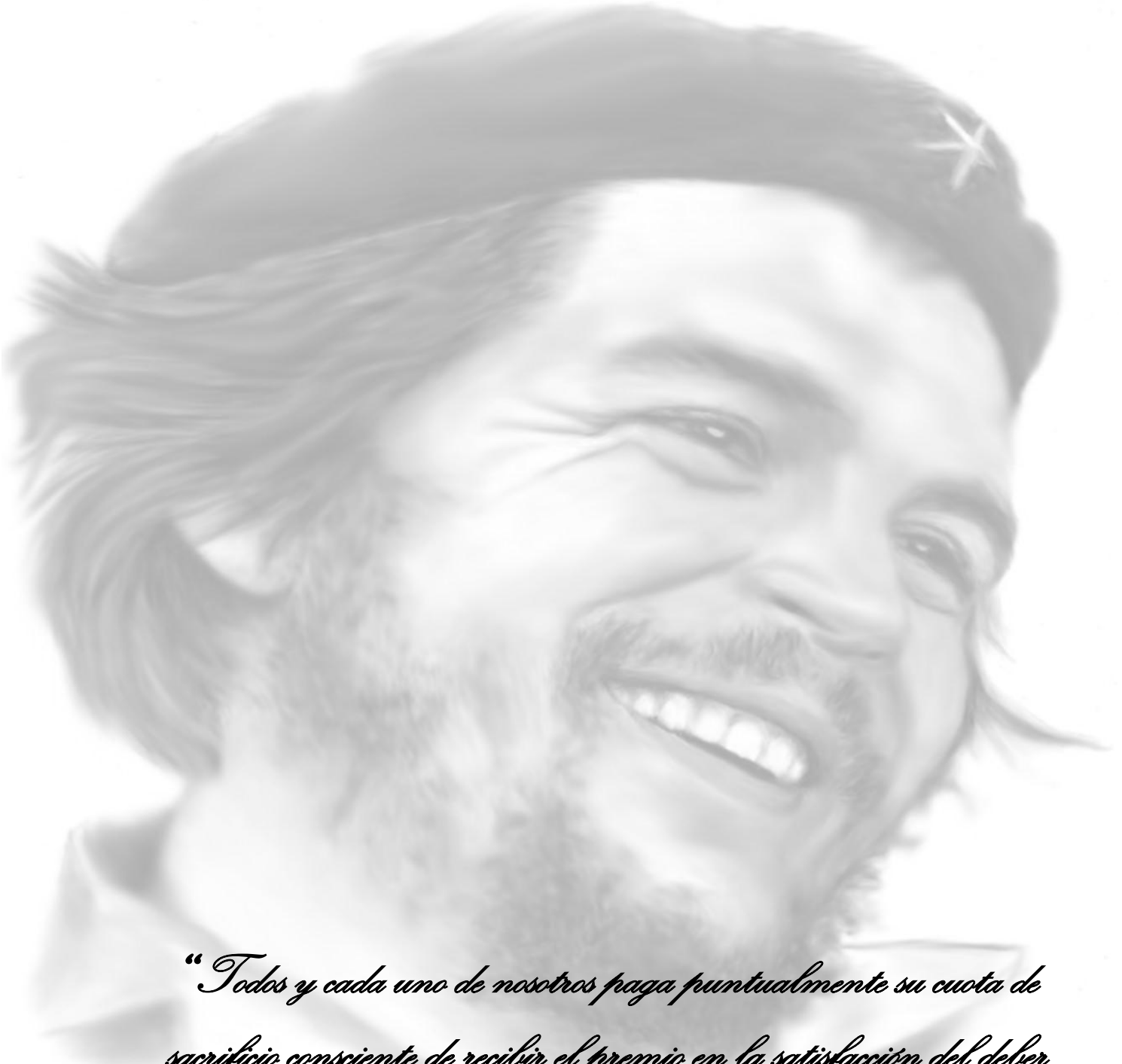
Tutor: Ing. Ernesto Carrasco De la Torre

Cotutores: Ing. Adrián Hernández Aguilera

Ing. Karel Piorno Charchabal

"Año 56 de la Revolución"

La Habana, Junio de 2014



"Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte."

Ernesto "Che" Guevara

Declaración de Autoría

Declaro que soy el único autor de esta investigación y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ___ días del mes de ____ del año ___.

_____	_____
Autor: Luis Angel Rodríguez Martínez	Tutor: Ing. Ernesto Carrasco De la Torre
_____	_____
Cotutor: Ing. Adrián Hernández Aguilera	Cotutor: Ing. Karel Piorno Charchabal

Datos de contacto

Autor: Luis Angel Rodríguez Martínez

E-mail: lamartinez@estudiantes.uci.cu

Tutor: Ernesto Carrasco De la Torre

Graduado de: Ingeniero en Ciencias Informáticas

Años de experiencia: 5

E-mail: ecarrasco@uci.cu

Cotutor: Adrián Hernández Aguilera

Graduado de: Ingeniero en Ciencias Informáticas

Años de experiencia: 2

E-mail: ahaguilera@uci.cu

Cotutor: Karel Piorno Charchabal

Graduado de: Ingeniero en Ciencias Informáticas

Años de experiencia: 2

E-mail: kpiorno@uci.cu

A mis padres.

A mi hermana, a mi familia.

A mi novia Yaneisi por tanto apoyo y cariño.

A mi madre, por creer siempre en mí.

A mi padre por ser un ejemplo para mí, gracias viejo por todo tu apoyo en la pelota y por los tantos consejos que me has dado.

A mi hermanita por ayudarme en los momentos difíciles, de todo corazón gracias.

A mi novia Yaneisi por todo su amor y comprensión en cada momento, gracias mi amor TE QUIERO.

A mi familia por su apoyo incondicional en todo momento.

A la familia de mi novia por acogerme como uno más de la familia, especialmente a Shirlita.

A mis tutores (Adrián, Karel y Ernesto) por toda la ayuda, gracias de todo corazón.

A Teijón y a Lorena por su gran amistad.

A los profesores de la facultad por su ayuda (Rubén, Gustavo, Hasan, Ernesto, Guille, Alina, Yadira y Ramón Carrasco).

A todos mis compañeros del equipo de pelota.

A mis amigos (Burgo, José Leandro, Robelkis, Yunier (mellizo), Yusnel (mellizo), Valero, Yenier y al chino).

A mis compañeros de apartamento por su amistad en estos cinco años.

A todas mis amistades.

A la vida.

Y a mí por tanto esfuerzo y sacrificio en estos cinco años.

Resumen

Los dispositivos móviles desde sus inicios se han caracterizado por un progreso vertiginoso en la tecnología que los sustenta. Estos han transitado por varias generaciones, todas caracterizadas por el incremento de la velocidad de transmisión y el aumento en las prestaciones y servicios.

Actualmente estos dispositivos cuentan con la limitante de no poder desarrollar y probar aplicaciones. A raíz de un estudio realizado se identificó al *framework* Kivy como una posible alternativa a la limitante planteada. El desarrollo de aplicaciones en dicho *framework* depende en gran medida de la experiencia de los programadores, debido a que no permite el completamiento de código, siendo además, el proceso de depuración y detección de errores engorroso. El objetivo de este trabajo es desarrollar un componente gráfico para el *framework* Kivy que permita la depuración, completamiento y edición de código en dispositivos móviles. Para dar cumplimiento al objetivo propuesto se utilizó como lenguaje de programación Python, como *framework* de desarrollo Kivy y como Entorno de Desarrollo Integrado PyCharm. El proceso de construcción del sistema fue guiado por la metodología Programación Extrema, generándose cada uno de los artefactos propuestos por la misma. Al culminar el presente trabajo se logra la implementación satisfactoria de un componente gráfico que permite crear y probar aplicaciones en dispositivos móviles, cumpliendo con los estándares de codificación definidos y las tareas planteadas. Además, se recomiendan algunos aspectos para el mejoramiento futuro de la aplicación.

Palabras clave: completamiento de código, depuración de código, dispositivos móviles, edición de código, Kivy.

Índice

Índice de figuras

Índice de Tablas

Introducción

El creciente avance de las Tecnologías de la Información y las Comunicaciones (TIC) ha propiciado un notable desarrollo en diversas esferas de la sociedad, destacándose en estas el campo de la comunicación. En el que el hombre, debido a la necesidad de comunicarse en cualquier momento y lugar, sin importar la distancia, el tiempo y las condiciones, ha desarrollado una serie de herramientas en su beneficio, que a lo largo de los años se han ido perfeccionando y desarrollando de manera significativa. Un ejemplo de ello son los dispositivos móviles, los que se definen como aparatos electrónicos de pequeño tamaño, con limitadas capacidades de procesamiento, con conexión permanente o intermitente a una red, lo suficientemente ligeros para ser transportados de manera sencilla por una persona (1).

Los dispositivos móviles desde sus inicios se han caracterizado por un progreso vertiginoso en la tecnología que los sustenta. Estos han transitado por varias generaciones, todas caracterizadas por el incremento de la velocidad de transmisión de datos, de las posibilidades de negocios y el aumento en las prestaciones y servicios. Los mismos se han convertido en una herramienta muy importante en la vida de las personas. Su gran desarrollo ha permitido la incorporación de nuevas tecnologías, destacándose en este sentido las pantallas *multitouch*, también conocidas como pantallas capacitivas, las que permiten detectar más de un punto de contacto al mismo tiempo (gestos múltiples), permitiendo a varios usuarios interactuar sobre la misma pantalla simultáneamente, al igual que ampliar, disminuir o girar una imagen con dos o más dedos al mismo tiempo.

En la actualidad estos dispositivos están provistos con sistemas operativos (OS, por sus siglas en inglés) que permiten manejar, controlar y actualizar sus datos y aplicaciones. Dichos sistemas son más simples que los empleados en las computadoras, ya que están orientados a la conectividad inalámbrica y a las diferentes maneras de introducir información en ellos. Existe una variada gama de aplicaciones que pueden ser ejecutadas en los sistemas operativos móviles, lo que ha provocado que grandes comunidades de desarrollo se encuentren inmersas en la creación de aplicaciones para dichos dispositivos.

Conjuntamente con la aparición de los sistemas operativos móviles emergen un grupo de herramientas de desarrollo de *software*, encargadas de la realización de aplicaciones para dichos terminales. En este sentido, surgen los llamados Kit de Desarrollo de

Software (SDK[1]), estos están compuestos a grandes rasgos por entornos de desarrollo, simuladores y/o emuladores de arquitecturas en los que es posible desplegar aplicaciones que puedan ser ejecutadas en dichos OS.

Actualmente estos dispositivos cuentan con la limitante de no poder desarrollar y probar aplicaciones. Esta afirmación la valida el hecho de que para desarrollar una aplicación que pueda ser ejecutada en un dispositivo móvil se necesita hacer uso de emuladores y/o simuladores de arquitecturas, los que en ocasiones resultan lentos para el desarrollo. Además de que en diversas ocasiones se hace necesario probar elementos propios de dichos terminales, dígase el acelerómetro, GPS y cámara, lo que se dificulta en dichas herramientas, ya que no cuentan con los elementos mencionados, siendo esta, otra de las limitantes presentes en el proceso de desarrollo de aplicaciones móviles.

En este escenario, surge el *framework* Kivy como solución a las limitantes descritas. Kivy es un *framework* de código abierto y multiplataforma, que permite el desarrollo de aplicaciones *multitouch*. El mismo propone una plataforma de desarrollo colaborativo (Garden) propiciada por sus patrocinadores, en la que son almacenadas las implementaciones de los módulos desarrollados por terceros[2]. El desarrollo de aplicaciones en dicho *framework* depende en gran medida de la experiencia de los programadores, debido a que no permite el completamiento de código, siendo además, el proceso de depuración y detección de errores engorroso. Un ejemplo que valida lo planteado anteriormente es, que para poder detectar la ocurrencia de errores en el desarrollo de una aplicación, se debe recurrir a un fichero que almacena las trazas, recomendaciones y errores. Es decir, si ocurre un problema determinado se debe recurrir a la dirección en la que se almacena el fichero.

Teniendo en cuenta lo antes expuesto se define como **problema a resolver**: ¿Cómo garantizar la depuración, completamiento y edición de código en dispositivos móviles utilizando el *framework* Kivy?

El objeto de estudio estará enfocado en el proceso de depuración, completamiento y edición de código.

Por lo que se define como **objetivo general** de esta investigación: Desarrollar un componente gráfico para el *framework* Kivy que permita la depuración, completamiento y edición de código en dispositivos móviles.

El **campo de acción** estará centrado en el proceso de depuración, completamiento y edición de código en dispositivos móviles.

[1] SDK. Acrónimo de: *Software Development Kit.*
[2] Personas ajenas a la plataforma.

Para dar cumplimiento al objetivo trazado se determinó que las **tareas de la investigación** estarían encaminadas a:

- ✓ Confección del marco teórico de la investigación para garantizar el basamento y fundamento científico de la misma.
- ✓ Análisis de las herramientas de edición de código en dispositivos móviles.
- ✓ Selección de la metodología, herramientas y tecnologías para el desarrollo de la propuesta solución.
- ✓ Análisis y diseño de la propuesta de solución para guiar la implementación del sistema.
- ✓ Implementación de la propuesta de solución para darle cumplimiento al objetivo de la investigación.
- ✓ Realización de pruebas al *software, que* permitan validar el correcto funcionamiento del sistema.

Por lo que se considera como **posibles resultados** del presente trabajo:

1. Los principales documentos y artefactos asociados al proceso de desarrollo, definidos por la metodología seleccionada.
2. Un componente gráfico para el *framework* Kivy que permita la depuración, completamiento y edición de código en dispositivos móviles.

Para el desarrollo de las tareas antes mencionadas se determinó utilizar los siguientes **métodos científicos**: teóricos y empíricos.

Los métodos teóricos:

Analítico-Sintético: utilizado para analizar y comprender la teoría y documentación relacionada con:

- ✓ Los Entornos de Desarrollo Integrado (IDE, por sus siglas inglés) existentes en la actualidad y la manera en que los mismos realizan la depuración, completamiento y edición de código.
- ✓ Los sistemas operativos móviles, así como las formas de realización de aplicaciones para dichos sistemas.
- ✓ El *framework* Kivy y Python como lenguaje interpretado a utilizar.
- ✓ Además, este método se emplea para definir los principales conceptos y definiciones que sustentan el basamento teórico de la presente investigación, el mismo permite hacer una selección de las herramientas necesarias para el desarrollo de la aplicación.

Histórico-Lógico: permite realizar un análisis histórico sobre los sistemas operativos móviles, Kivy y Python, posibilitando el análisis de la trayectoria de estos, lo que permite una mejor comprensión de los mismos.

Los métodos empíricos:

Observación: utilizado para detectar las necesidades existentes así como las posibles mejoras que se le pueden añadir al sistema en desarrollo, con el propósito de obtener una aplicación mejor estructurada. Además, permite realizar un registro visual de lo que ocurre en el entorno del problema.

La consulta de información de todo tipo de fuentes: utilizado para la elaboración del marco teórico de la investigación, así como para investigar sobre las herramientas, lenguajes y metodología a utilizar en la realización del sistema.

EL presente trabajo de diploma se encuentra estructurado en **3 capítulos**.

Capítulo 1. "Fundamentación teórica": Contiene todo el basamento teórico que sustenta la presente investigación. En este capítulo se describen las herramientas, lenguajes y metodología a utilizar en el desarrollo del sistema, justificando la selección de las mismas.

Capítulo 2. "Propuesta de solución": En este capítulo se describe la propuesta de solución. Contiene elementos significativos como las principales características que debe cumplir el sistema a desarrollar. Se abordan las fases de Exploración, Planificación e Iteraciones. Además, se especifican la arquitectura, los patrones de diseño y los artefactos que propone la metodología seleccionada.

Capítulo 3. "Implementación y prueba": El capítulo se centra en los procesos de implementación (perteneciente a la fase Iteraciones) y prueba de las funcionalidades del componente gráfico (perteneciente a la fase de Producción). Se identifican y describen las tareas de ingeniería. Además, se detallan las pruebas realizadas al *software* con el objetivo de asegurar la calidad de la solución.

Capítulo1: Fundamentación teórica

1.1 Introducción

Este capítulo tiene como principal objetivo explicar de manera más profunda los conceptos y definiciones necesarias para dar cumplimiento al objetivo general planteado, logrando así una mayor comprensión de este trabajo. Además, se analizarán las posibles herramientas, lenguajes y metodología a utilizar para la elaboración de la aplicación, justificando la selección de las mismas.

1.2 Conceptos asociados al dominio del problema

A continuación se muestran un grupo de conceptos y definiciones que harán posible un mejor entendimiento de la presente investigación.

1.2.1 Dispositivos móviles

Los dispositivos móviles son equipos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red. Tienen memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales (1).

Un dispositivo móvil es un micro-ordenador lo suficientemente ligero como para ser transportado por una persona. Estos disponen de una capacidad de batería suficiente como para poder funcionar de forma autónoma, normalmente son versiones limitadas en prestaciones y por tanto en funcionalidades (2).

A partir del estudio de varios conceptos se define como dispositivo móvil: un objeto (generalmente pequeño) que se caracteriza por su gran portabilidad, a través del cual se puede tener acceso permanente o intermitente a una red. Estos dispositivos están provistos de aplicaciones que brindan una gran cantidad de funcionalidades, lo que permite un mejor uso de los mismos. Las aplicaciones que conforman estos dispositivos son conocidas como aplicaciones móviles.

1.2.2 Aplicaciones móviles

Las aplicaciones móviles son extensiones informáticas para dispositivos portátiles, diseñadas para ser ejecutadas en teléfonos inteligentes, tabletas y asistentes digitales personales. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS y Windows Phone (3). Una aplicación móvil es un programa informático creado para llevar a cabo o facilitar una tarea en un dispositivo móvil.

1.2.3 Framework

Un *framework* es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, que puede servir de base para la organización y desarrollo de *software*. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado[3], entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Son diseñados con la intención de facilitar el desarrollo de *software*, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de *software* (4).

Teniendo en cuenta lo antes expuesto se puede afirmar que un *framework* representa una arquitectura de *software* que modela las relaciones generales de las entidades del dominio, provee una estructura y una metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio. Desde el punto de vista del desarrollo de *software*, un *framework* es una estructura de soporte definida, en la cual otro proyecto de *software* puede ser organizado y desarrollado.

1.2.4 Depuración de código

La depuración de código es el proceso de identificar y corregir errores de programación, consiste en revisar y analizar si la sintaxis de un programa creado es correcta y/o genera errores al ejecutarlo. En inglés se denomina a este proceso *debugging*, esto es, eliminar *bugs*[4] (5). Es el proceso de encontrar y reducir el número de errores, o defectos en un programa. Mediante la depuración lo que se hace es ejecutar el programa paso a paso y si alguna instrucción no es correcta o no la comprende el ordenador, entonces se genera un informe de error para esa instrucción en particular, permitiendo así que el programador la corrija posteriormente.

1.2.5 Completamiento de código

El completamiento de código es el proceso de predicción de una palabra o frase que el usuario desea escribir. De manera tal que cuando el usuario escribe la primera letra o letras de una palabra, el programa predice una o varias palabras como posibles opciones (6).

1.2.6 Editor de código

Un editor de código es un editor de texto diseñado específicamente para editar el código fuente de programas informáticos. Puede ser una aplicación individual o estar incluido en un entorno de desarrollo integrado. Los editores de código fuente tienen

[3] Es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete.
[4] Error de *software*.

características diseñadas exclusivamente para simplificar y acelerar la escritura de código fuente, como: resaltado de sintaxis y autocompletado de código (7).

1.3 Depuración, completamiento y edición de código en los Entornos de Desarrollo Integrado

La depuración, completamiento y edición de código son características que están presentes en los entornos de desarrollo integrado. A continuación se profundizará en los mismos con el objetivo de determinar cómo estos manejan dichas características y proporcionar una guía para definir las mejores alternativas, lo que permitirá el cumplimiento del objetivo general definido en el presente trabajo.

Un IDE es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos. Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario *(GUI*[5]*)* y opcionalmente, un sistema de control de versiones (8).

Otra bibliografía plantea:

(...) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes (…) (9)*.*

Teniendo en cuenta las bibliografías consultadas se puede decir que un IDE es un programa informático compuesto por varias herramientas de programación, las que facilitan el trabajo del desarrollador, haciéndolo más dinámico. Los mismos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación mediante el uso de una interfaz gráfica de usuario, lo que facilita el diseño e implementación de las aplicaciones.

Existen disímiles entornos de desarrollo que ejemplifican el concepto expuesto con anterioridad como son: QTCreator, Visual Studio, PyCharm, Eclipse, NetBeans, entre otros. De acuerdo con el objetivo propuesto en el presente trabajo se describirán algunos de estos, con la finalidad de crear la base teórica necesaria para el posterior desarrollo de la propuesta de solución. Dichos IDEs son:

- ✓ Eclipse

[5] Acrónimo de *Graphical User Interface* (Interfaz Gráfica de Usuario).

- ✓ Visual Studio
- ✓ PyCharm

Estos IDEs normalmente están compuestos por:

- ✓ Un editor de código fuente.
- ✓ Un compilador y / o intérprete.
- ✓ Un depurador**.**
- ✓ Automatización para la generación de herramientas.
- ✓ Un sistema de ayuda.

1.3.1 Composición de los IDE

Muchos entornos de desarrollo modernos poseen, además de las partes antes mencionadas, un navegador de clases, un inspector de objetos y un diseñador de diagramas de jerarquía de clases, normalmente, todo ello para su uso con el desarrollo de *software* orientado a objetos. También están integrados por un sistema de control de versiones y varias herramientas para simplificar la construcción de una GUI. El límite en cuanto al conjunto de herramientas que puede o no poseer un entorno de desarrollo no está definido, por lo que, a modo de acotar el contenido a tratar en este documento, a continuación se explican las fundamentales para el desarrollo del presente trabajo (editor de código fuente, compilador y/o intérprete y depurador).

Editor de código fuente

Un editor de código fuente es un editor de texto del programa, diseñado específicamente para la edición de código fuente de programas informáticos. Puede ser una aplicación independiente o normalmente suele estar integrado (10).

Estos están diseñados con características específicas que permiten simplificar y acelerar la entrada del código fuente, como son: el resaltado de sintaxis, que ocurre mientras se programa en tiempo real, avisando de inmediato de los problemas de sintaxis y el autocompletado de código. Los editores brindan también un conjunto de funcionalidades de acceso, que permiten ejecutar un compilador, intérprete, depurador u otra herramienta que sea útil en el proceso de obtención del *software* (10).

En el estudio realizado se identificó que los IDEs en los que se profundizó disponen de un editor de código con resaltado de sintaxis y autocompletado de código que brinda las siguientes opciones:

- ✓ Abrir múltiples archivos a la vez mediante fichas o pestañas, de un modo similar al formato de exploración por fichas que ofrecen muchos exploradores.
- ✓ Un panel de proyecto que proporciona una vista de la estructura de ficheros de un directorio dado.

- ✓ Funciones que permiten la búsqueda y reemplazo de una palabra o una cadena de caracteres.
- ✓ Copiar, cortar y pegar una región marcada.
- ✓ Deshacer y rehacer una operación realizada.
- ✓ Importar un fichero en el archivo que se está editando.

De las opciones antes mencionadas se empleará para el desarrollo de la aplicación el editor de código con resaltado de sintaxis y autocompletado de código, así como las funciones de copiar, pegar e importar un fichero en el archivo que se está editando.

Compilador y/o Intérprete

Un compilador es un programa de computadora que transforma código fuente legible, comprensible por los desarrolladores, en código máquina de forma tal que pueda ser ejecutado por el CPU[6]. A este proceso se le denomina compilación, también puede ser entendido como el proceso de transformación de código de un lenguaje de programación de origen (código fuente) a un lenguaje de destino, este último que a menudo es codificado en binario, es conocido como lenguaje objeto. La razón más común por la que se realiza la compilación es para crear un ejecutable del programa escrito. Un intérprete es un programa informático capaz de analizar y ejecutar otros programas (11). Los intérpretes difieren de los compiladores en que, estos últimos, traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, generalmente producen un binario ejecutable, que puede utilizarse cuando sea requerido. Por otro lado, los intérpretes realizan el proceso de compilación o sea: análisis léxico, sintáctico, semántico y generación del código intermedio y luego ejecutan el resultado directamente. La interpretación se realiza a medida que sea necesaria, típicamente, instrucción por instrucción y normalmente no guardan el resultado del proceso (12).

De acuerdo con la definición expuesta anteriormente se decide el uso de un intérprete para la ejecución del código previamente editado en el componente a desarrollar.

Depurador

Un depurador (*debugger*) es un programa usado para probar y depurar, o sea, detectar los errores de otros programas, denominados “programas objetivo”. El código al ser examinado puede alternativamente ser ejecutado. Una técnica que permite gran potencia es su capacidad de detenerse cuando son encontradas condiciones específicas, pero será típicamente algo más lento que ejecutando el código directamente.

[6] CPU: Acrónimo de *Central Processing Unit* (Unidad Central de Procesamiento).

Algunos depuradores ofrecen dos modos de depuración: la simulación parcial y la completa; para limitar este impacto. Típicamente, los depuradores también ofrecen funciones más sofisticadas tales como: ejecutar un programa paso a paso, detener la ejecución de un programa para examinar el estado actual en cierto evento o instrucción especificada por medio de un *breakpoint*[7], y el seguimiento de valores de algunas variables. Algunos depuradores tienen la capacidad de modificar el estado del programa mientras se está ejecutando, en vez de simplemente observarlo. A continuación se muestran elementos obtenidos a partir de un estudio realizado al proceso de depuración propuesto en los IDEs seleccionados.

Depurador de Eclipse

El depurador de Eclipse no es un proyecto único, está compuesto por dos subproyectos:

- ✓ Plataforma de depuración
- ✓ JDT depuración[8]

El componente de depuración de la plataforma define interfaces para un modelo de depuración independiente del lenguaje, que resumen las características de depuración comunes de muchos lenguajes. El componente de depuración de la plataforma no proporciona una implementación de un depurador, es deber de los otros *plugins*[9] proporcionar implementaciones específicas de un depurador dependiendo del lenguaje que se desea utilizar.

Depurador de Microsoft Visual Studio

Visual Studio incluye un depurador que funciona como depurador a nivel de fuente y como depurador a nivel de máquina. Se puede utilizar para depurar aplicaciones escritas en cualquier lenguaje compatible con Visual Studio. Además, también se puede unir a los procesos en ejecución y supervisar y depurar esos procesos. El depurador de Visual Studio también puede crear volcados de memoria así como cargar más tarde para la depuración. También se admiten programas de subprocesos múltiples. El depurador permite establecer puntos de interrupción (la ejecución se detiene temporalmente en una posición determinada) y relojes (los cuales controlan los valores de las variables y cómo la ejecución progresa). Cuando se realiza la depuración, si el puntero del ratón se cierne sobre cualquier variable, su valor actual se muestra en la ventana de herramientas, donde también se puede modificar si se desea (13).

Depurador de PyCharm

[7] Punto de interrupción.
[8] Permite la compatibilidad con la depuración de Java.
[9] Extensiones.

PyCharm puede ejecutar y depurar varios tipos de aplicaciones sin salir del IDE. Además, usando el depurador de PyCharm, puede encontrar el origen de los errores y las excepciones en tiempo de ejecución. El depurador permite ejecutar su aplicación paso a paso, examinar la información del programa en relación con las variables, relojes, o hilos, y cambiar su programa sin salir del IDE. Antes de lanzar la sesión del depurador, tiene que establecer puntos de interrupción que causan que el depurador pueda suspender la aplicación (o tomar otras acciones) cuando se alcanza tal punto. La salida generada y la información de la consola durante las sesiones de carrera o de depuración se muestran en la ventana de herramientas (14).

Una vez estudiado el proceso de depuración en los IDEs abordados, se decide emplear el modo de depuración completa mediante el cual el depurador realiza el análisis del código previamente editado y se detiene solo en caso de existir algún tipo de error (informando el lugar donde ocurre el mismo) o después de analizar la última línea de código. Para la elaboración del depurador del presente trabajo se hará uso de funcionalidades y bibliotecas propias del lenguaje de programación a emplear, dicho depurador debe brindar la posibilidad de realizar el seguimiento de código línea a línea.

1.4 Herramientas para la edición de código en dispositivos móviles

A continuación se realiza un estudio de las principales herramientas utilizadas en el proceso de edición de código en dispositivos móviles a nivel mundial.

1.4.1 DroidEdit

DroidEdit es un editor especial de texto y código para Android. ¿Por qué es especial?, porque sus funciones van más allá de solo escribir texto. Puedes escribir código prácticamente de cualquier lenguaje de programación, por ejemplo: Python, C# y C++. Permite escribir un *script*[10] en Python y guardar el texto en su extensión .py, automáticamente este resalta las variables y las órdenes como si se estuviera trabajando sobre el editor de Python. Es lento cuando se tienen muchas líneas de código, haciendo que la edición del código se dificulte, siendo esta una de las limitantes presentes en el mismo (15). La principal desventaja del mismo es que no brinda opciones de autocompletado de código, además de la ausencia de un depurador que permita conocer los errores presentes en el código editado.

1.4.2 920 Text Editor

920 Text Editor es un editor de código para Android con licencia GPL[11]. En un principio era únicamente un editor de texto, pero ha ido incorporando poco a poco numerosas

[10] Es un guion o conjunto de instrucciones.
[11] Licencia Pública General orientada a proteger la libre distribución, modificación y uso de *software*.

novedades. Hoy en día reconoce y resalta el código específico de más de 20 lenguajes diferentes. Dispone también de: numeración de líneas y pestañas, caracteres especiales de programación y diferentes funciones muy prácticas (duplicar texto, convertir a mayúsculas/minúsculas, ir a líneas en concreto e insertar tiempos) (16). En definitiva, un buen editor de código para Android completamente gratuito. Además, se puede utilizar como lector de libros electrónicos y para abrir archivos txt (17). Este no permite conocer la presencia de errores en el código editado, lo que dificulta el proceso de edición.

1.4.3 Touchqode

Touchqode es un editor de código que posee herramientas de desarrollo de *software* para dispositivos móviles. Permite ver y editar código fuente en dispositivos con OS Android, la versión para iOS está en proceso de desarrollo (18). Permite redactar y editar código fuente de una manera simple y con una interfaz bastante amigable con resaltado de sintaxis (19). Su principal desventaja está dada por la ausencia de un depurador que permita conocer la ocurrencia de errores en el código que se está editando.

1.4.4 Scripting Layer for Android

Scripting Layer for Android (SL4A), originalmente llamado ASE (*Android Script Enviroment*), permite que desarrolladores editen, ejecuten *scripts* e interactúen con intérpretes directamente en el dispositivo, así como el resaltado de sintaxis. Estos *scripts* tienen acceso a muchas de las APIs del sistema operativo, pero con una interfaz muy simplificada que hace que sea fácil:

- ✓ Realizar llamadas telefónicas.
- ✓ Enviar mensajes de texto.
- ✓ Escanear códigos de barras.

SL4A es desarrollado y es distribuido bajo la licencia Apache 2.0, también soporta otros lenguajes como JRuby, Lua, Perl y Rhino (20). El desarrollo de aplicaciones en esta herramienta suele ser engorroso ya que no permite el completamiento de código. En la siguiente imagen se muestran las partes que componen la arquitectura de una aplicación realizada en SL4A.

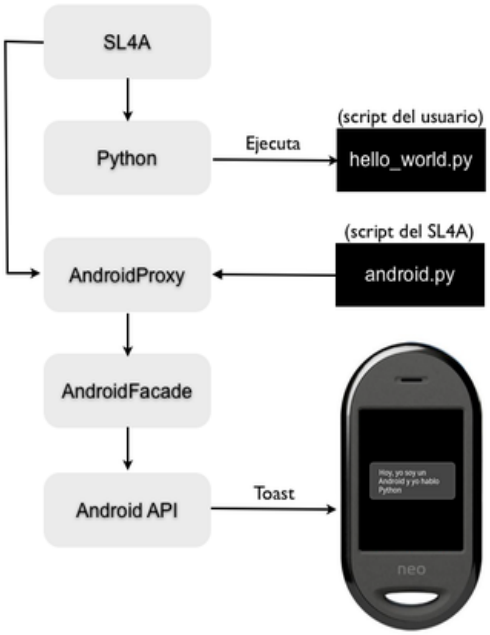


Figura 1: Arquitectura de una aplicación en SL4A.

Después de un estudio realizado a los sistemas abordados, se concluye que ninguna de estas herramientas se adaptan a las características del sistema a desarrollar, ya que poseen la limitante de no poderse integrar al *framework* Kivy, además, de que no permiten el completamiento y la depuración de código. Dicho estudio permitió obtener una visión general de la edición de código en los dispositivos móviles, lo que servirá de guía en el desarrollo del sistema que se desea obtener al concluir la presente investigación.

1.5 Framework Kivy

En la investigación realizada se hizo necesario especificar que el componente a desarrollar tuviese las siguientes características:

- ✓ **Usabilidad:** Funcionar en dispositivos con pantalla *multitouch*.
- ✓ **Multiplataforma:** La aplicación podrá ser ejecutada en varios sistemas operativos móviles.
- ✓ ***Software* libre y código abierto:** Debe brindar la posibilidad de ser modificado por otros desarrolladores.

Teniendo en cuenta las características identificadas anteriormente, se determina el uso del *framework* Kivy para el desarrollo del componente que se desea obtener al concluir el presente trabajo.

Kivy es un *framework* de código abierto y multiplataforma realizado en el lenguaje de programación Python que permite desarrollar aplicaciones *multitouch,* las mismas pueden ser ejecutadas en (21):

- ✓ Computadoras de escritorio: Mac OSX, Linux, Windows.
- ✓ Dispositivos con Android: tabletas, teléfonos.
- ✓ Dispositivos con iOS: iPad, iPhone.
- ✓ Cualquier otro dispositivo con entrada táctil que admita TUIO[12].

Kivy es desarrollado, respaldado y utilizado profesionalmente por Kivy Org (Organización Kivy). Puede ser empleado en el desarrollo de aplicaciones comerciales bajo los términos de la licencia de *software* MIT. Es un producto estable y tiene una API[13] bien documentada, además de una guía de programación para principiantes. El motor gráfico está construido sobre OpenGL, cuenta con un kit de herramientas con más de 20 *widgets*, todas altamente extensibles. Incluye además un lenguaje declarativo propio llamado *Kivy Language* (KV) que permite: la creación de prototipos muy rápidos, realizar ágiles cambios en la interfaz de usuario y una buena separación entre la lógica de la aplicación y su interfaz de usuario (21). A continuación se muestra una imagen en la que se ilustra la forma en que dicho *framework* controla los diferentes eventos de entrada (22).

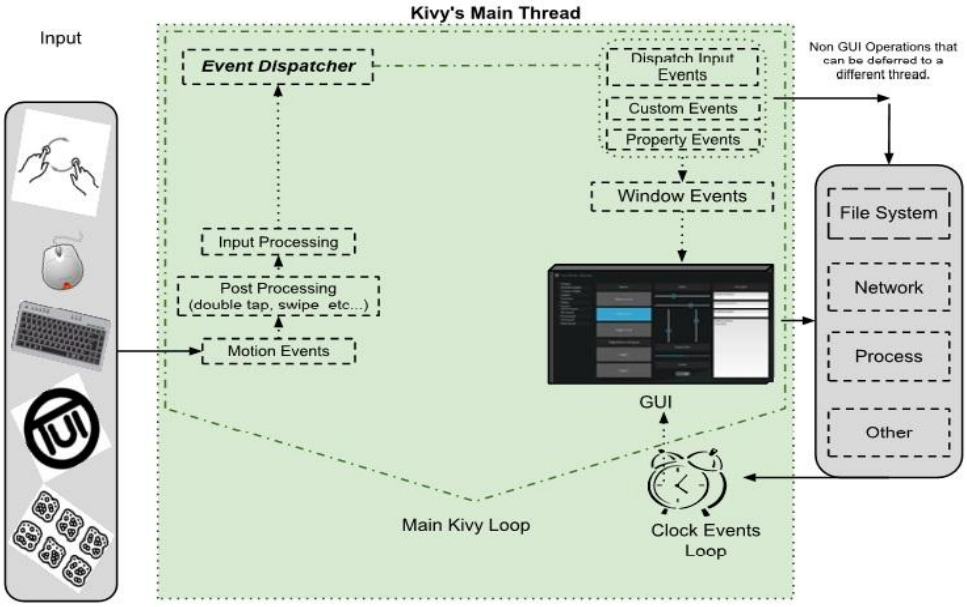


Figura 2: Manejo de eventos en Kivy.

[12] TUIO: Acrónimo de *Tangible User Interface Objects* (Objetos de interfaz de usuario tangible).
[13] API: Acrónimo de *Aplication Programing Interface.*

Una vez descrito el *framework* Kivy se hace necesario la selección del sistema operativo en el cual se validará la aplicación a desarrollar, teniendo como premisa que debe ser un OS para dispositivos móviles.

1.6 Selección del sistema operativo móvil

Los sistemas operativos son una parte esencial de cualquier sistema informático. Un sistema operativo es un programa que administra el *hardware* de un sistema informático. Además, proporciona los mecanismos apropiados para asegurar el correcto funcionamiento del sistema e impedir que los programas de usuario interfieran con el apropiado funcionamiento del sistema (23).

Existe una variada gama de sistemas informáticos para los que se diseñan sistemas operativos. Tal es el caso de los dispositivos móviles, en los cuales se tratan de añadir funciones propias de los ordenadores. Ejemplos de estos OS para dispositivos móviles son: iOS, Windows Phone y Android.

1.6.1 iOS

iOS es un sistema operativo desarrollado por la compañía Apple originalmente para su teléfono inteligente iPhone, pero lo emplean otros de sus productos como el iPod y el iPad. Dicha compañía no permite que iOS esté presente en dispositivos de terceras compañías. Su interfaz gráfica está diseñada para *touch screen*[14], con capacidad para gestos *multitouch*. Este sistema permite la adaptación total al dispositivo, puesto que ambos han sido concebidos como partes de un todo, por lo tanto, ofrecen la misma experiencia de uso para todos los consumidores (24).

La gran limitante en este caso es el férreo control sobre el *hardware*, siendo Apple la encargada de diseñar tanto iOS como el dispositivo donde se ejecuta, además del coste de adquisición y la nula variedad de dispositivos en el mercado, solo existe uno y no hay opciones de elección (24), siendo además un *software* propietario.

1.6.2 Windows Phone

Windows Phone es un sistema operativo móvil desarrollado por Microsoft, como sucesor de la plataforma Windows Mobile. Con Windows Phone Microsoft ofrece una nueva interfaz de usuario e integra varios servicios en el OS. Como fabricante del sistema, Microsoft requiere que todo dispositivo que desee ejecutar Windows Phone disponga de unas características mínimas, para asegurar la consistencia de todos los usuarios del

[14] Pantallas sensibles al tacto.

sistema, a partir de estas características los fabricantes de *hardware* son libres de ampliarlas en algunos casos y están obligados a cumplirlas con exactitud en otros (25).

El problema para los programadores que deseen utilizar aplicaciones creadas por ellos mismos, es que la única forma de instalar una aplicación en Windows Phone es mediante la tienda oficial de Microsoft: *Marketplace*; en la cual se deben registrar como desarrolladores para poder vender aplicaciones, debido a que dicho OS es *software* propietario. Es decir, si Microsoft no publica la aplicación de determinado desarrollador, este nunca podrá utilizarla, ni siquiera en su propio dispositivo (25).

1.6.3 Android

Android es un sistema operativo para dispositivos móviles basado en GNU/Linux. La presentación de la plataforma Android se realizó el 5 de noviembre de 2007 junto con la fundación *Open Handset Alliance*, un consorcio de compañías de *hardware*, *software* y telecomunicaciones comprometidas en la promoción de estándares abiertos para dispositivos móviles (26).

(…) es una plataforma de código abierto, esta plataforma permite el desarrollo de aplicaciones por terceros[15]*.La mayoría del código fuente de Android ha sido publicado bajo la licencia de software Apache, una licencia de software libre y código fuente abierto, además el mismo es un OS basado en el Kernel de Linux, orientado a dispositivos móviles tales como teléfonos inteligentes, tabletas y PDAs*[16] *(…)* (26).

A nivel internacional Android es tratado como un OS, cuando realmente es un paquete de *software* que permite ser instalado en cualquier equipo móvil. Este paquete de *software* incluye un OS Linux, provee además de una capa de bibliotecas escritas en C y C++ y un marco de trabajo para el desarrollo de aplicaciones (27).

Arquitectura de Android (28)**:**

La arquitectura interna de la plataforma Android está básicamente formada por cuatro componentes:

- ✓ Aplicaciones: Todas las aplicaciones creadas con la plataforma Android, incluyen como base un cliente de *email*[17], calendario, programa de SMS[18], mapas, navegador, contactos, y algunos otros servicios mínimos. Todas ellas escritas en el lenguaje de programación Java.

[15] Personas ajenas a Google.
[16] PDA. Acrónimo de *Personal Digital Assistants* (Asistentes Digitales Personales).
[17] Correo electrónico.
[18] Mensaje de texto corto que se puede enviar entre teléfonos celulares o móviles.

- ✓ Marco de aplicaciones: Todos los desarrolladores de aplicaciones para Android, tienen acceso total al código fuente usado en las aplicaciones base. Esto ha sido diseñado de esta forma para que no se generen cientos de componentes de aplicaciones distintas, que respondan a la misma acción, dando la posibilidad de que los programas sean modificados o reemplazados por cualquier usuario sin tener que empezar a programar sus aplicaciones desde el principio.
- ✓ Bibliotecas: Android incluye en su base de datos un set de bibliotecas C/C++ expuestas a todos los desarrolladores a través del *framework* de las aplicaciones *Android System C library*.
- ✓ Entorno de ejecución: Android incorpora un set de bibliotecas que aportan la mayor parte de las funcionalidades disponibles en las bibliotecas bases del lenguaje de programación Java. La máquina virtual está basada en registros y corre clases compiladas por el compilador de Java, que anteriormente han sido transformadas al formato .dex (*Dalvik Executable*) por la herramienta "dx".

La siguiente imagen muestra los componentes que conforman la arquitectura de la plataforma Android (29):

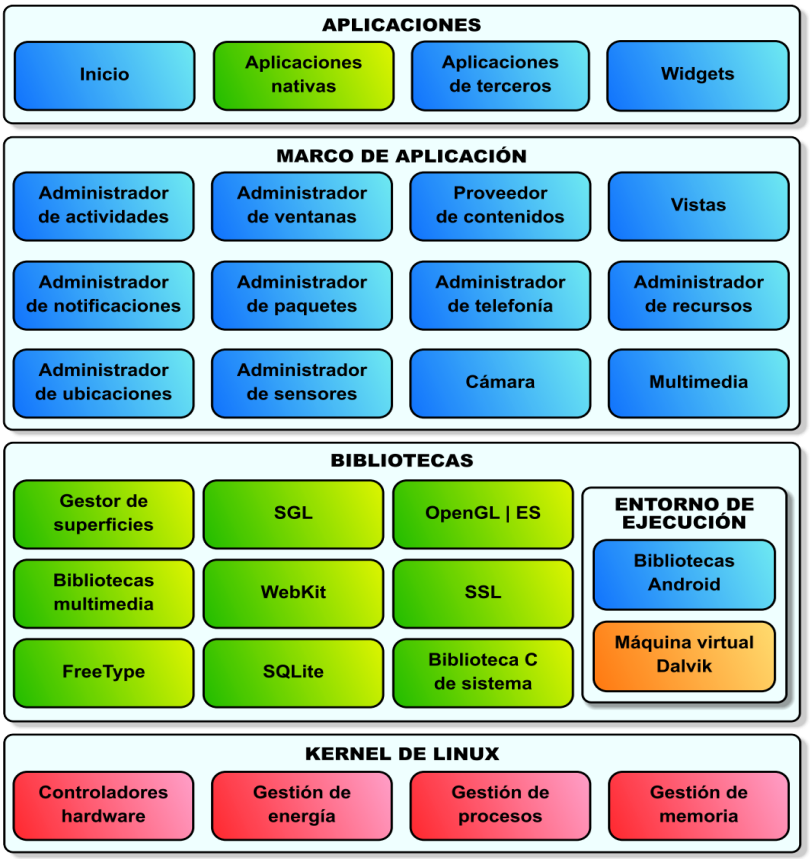


Figura 3: Arquitectura de Android.

Ventajas y Desventajas del uso de Android (30)

Ventajas

- ✓ Puede instalarse prácticamente en todo tipo de dispositivos, sean móviles, portátiles e incluso microondas.
- ✓ Puede adaptarse a todo tipo de necesidades.
- ✓ Es libre con licencia Apache y código abierto para que un desarrollador no solo pueda modificar su código sino también mejorarlo.
- ✓ Garantiza que, en caso de haber un error, sea detectado y reparado con mayor presteza, sin depender de nadie para pedir autorización a su cambio.
- ✓ Personalizable.

Desventajas

- ✓ Consumo de la batería: debido a la posibilidad de tener varias aplicaciones corriendo a la vez en el sistema, esto implica un gasto incremental en el consumo de la batería.

Los creadores de la tecnología Android han desarrollado un OS que cualquier fabricante puede licenciar e incluir en sus dispositivos, sin unos requerimientos mínimos de *hardware* ni límites en la personalización de la interfaz de usuario. Su principal ventaja es la cantidad de dispositivos, gracias a este modelo, el mercado puede ser inundado con cientos de modelos distintos y el usuario tiene libertad para elegir entre terminales de gama baja, media o alta.

Teniendo en cuenta las limitantes identificadas en iOS y Windows Phone, se decide utilizar Android para la validación del componente a desarrollar, siendo este un OS que posee un gran impacto y uso en el mercado mundial. Reafirma esta selección el hecho de que Cuba se encuentra inmersa en un proceso de migración a *software* libre, siendo la Universidad de las Ciencias Informáticas partícipe del mismo.

1.7 Metodología, herramientas, lenguajes y bibliotecas a utilizar

El desarrollo de una aplicación está determinado por la metodología, herramientas y tecnologías que se utilicen. El principal objetivo de estas, es guiar y facilitar el proceso de desarrollo, garantizando a su vez la calidad del producto final. A continuación se definen las utilizadas por el autor en la implementación de la propuesta solución.

1.7.1 Metodología de desarrollo

Una metodología de desarrollo de *software* es un conjunto ordenado de pasos a seguir para llegar a la solución de un problema u obtención de un producto, o sea el *software*, son también los pasos generales que sigue el proceso de desarrollo de un producto de *software*. Estas se basan en un conjunto de procedimientos, técnicas y herramientas que se utilizan para guiar el desarrollo de productos de *software* (31).

El desarrollo de *software* no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte se tienen aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Otra aproximación es centrarse en diferentes dimensiones, como por ejemplo el factor humano o el producto de *software*. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del *software* con iteraciones muy cortas. Este enfoque ha mostrado su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir los tiempos de desarrollo pero manteniendo una alta calidad.

Debido a las posibilidades que brindan las metodologías ágiles con respecto a las tradicionales (32) (Anexo 1) y a las características que posee el sistema a desarrollar (requisitos cambiantes y el cliente forma parte del equipo de trabajo) se decide usar como guía para el proceso de construcción del sistema a desarrollar una metodología ágil. En la actualidad existe un gran número de metodologías ágiles orientadas al desarrollo de *software*, entre las que se pueden mencionar XP y SCRUM, las cuales serán abordadas a continuación.

Extreme Programing (XP)

La Programación Extrema o Extreme Programming es una metodología ágil formulada por Kent Beck a partir de un conjunto de principios, prácticas y técnicas que facilitan de manera exitosa la finalización de proyectos. Esta surge como respuesta y posible solución a los problemas derivados del constante cambio en los requerimientos (33). Es la metodología más destacada en los procesos ágiles de desarrollo de *software* (34). La misma se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del *software*, promueve el trabajo en equipo y se basa en la retroalimentación entre el cliente y el equipo de desarrollo. Básicamente consiste en trabajar estrechamente con el cliente, haciendo pequeñas iteraciones[19], donde no exista más documentación que el código en sí; cada versión contiene las modificaciones pertinentes según el cliente vaya retroalimentando el sistema, por eso es necesaria la disponibilidad del cliente durante todo el desarrollo (34).

Los objetivos de XP son muy simples (35):

[19] Mini-entregas.

- ✓ La satisfacción del cliente. Esta metodología trata de dar al cliente el *software* que él necesita y cuando lo necesita.
- ✓ Potenciar al máximo el trabajo en equipo. Tanto los jefes de proyecto, como los clientes y desarrolladores forman parte del equipo y están involucrados en el desarrollo del *software*.

Características de XP (36):

- ✓ Desarrollo iterativo e incremental.
- ✓ Pruebas unitarias continuas.
- ✓ Frecuente interacción del equipo de programación con el cliente o usuario.
- ✓ Corrección de todos los errores antes de añadir una nueva funcionalidad.

El ciclo de vida ideal consta de 6 fases:

Fase I: Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo, el equipo de desarrollo se familiariza con las herramientas y tecnologías que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y la familiarización que tengan los programadores con la tecnología (32).

Fase II: Planificación de la Entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días (32).

Fase III: Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción (32).

Fase IV: Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase (32).

Fase V: Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura (32).

Fase VI: Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo (32).

SCRUM

SCRUM es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. Es una metodología ágil que propone realizar entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto (cliente). Por ello, está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados con urgencia, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales, además un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). SCRUM es muy fácil de aprender, ya que requiere de muy poco esfuerzo para utilizarlo, pero no se tiene una perspectiva global del proyecto más allá de una lista de tareas. Además, no genera toda la documentación que se obtiene con otras metodologías por lo que no es muy conveniente utilizarla sola (37).

A partir de una comparación realizada entre XP y SCRUM (32) (Anexo 2) se decide utilizar XP para guiar el proceso de construcción del sistema teniendo en cuenta el índice de agilidad de la misma, además de que:

- ✓ Es la que mejor se ajusta a las necesidades del proyecto en cuanto a recursos técnicos, humanos y tiempo de desarrollo.
- ✓ El equipo de trabajo es pequeño y el cliente forma parte de él, mitiga el riesgo y aumenta la probabilidad de éxito.
- ✓ Es adecuada para proyectos con requisitos imprecisos y cambiantes.
- ✓ Permite que en cualquier momento del proceso de desarrollo de la aplicación, se pueda regresar a operaciones anteriores sin afectar el ciclo de vida del sistema.

1.7.2 Lenguaje Unificado de Modelado (UML) V2.5

El Lenguaje Unificado de Modelado o UML no es un lenguaje de programación, es un lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de *software*. Se utiliza para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. Incluye conceptos semánticos, notación, y principios generales (38).

De manera general sus principales características son (38):

- ✓ Modelar sistemas utilizando técnicas orientadas a objetos.
- ✓ Documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, entre otros.).
- ✓ Especificar todas las decisiones de análisis, diseño e implementación construyéndose así modelos precisos, sin ambigüedades y correctos.

1.7.3 Herramienta para el modelado UML

Para la realización del modelado de datos y teniendo en cuenta el lenguaje de modelado seleccionado se decide emplear la herramienta Visual Paradigm.

Visual Paradigm v8.0

Visual Paradigm es una herramienta que utiliza como lenguaje de modelado UML, soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El *software* de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite

dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (39).

Entre las principales características de la herramienta están (40):

- ✓ Entorno de creación de diagramas para UML v2.5.
- ✓ Importación y exportación de ficheros.
- ✓ Uso de un lenguaje estándar y común para todo el equipo de desarrollo, que facilita la comunicación.

1.7.4 Herramienta para el diseño de prototipos de interfaz

Con el objetivo de proporcionar una mayor visión al cliente, del componente a desarrollar se decidió utilizar Balsamiq MocKups como herramienta para el diseño de prototipos de interfaz. A continuación se explicará en detalles la misma.

Balsamiq Mockups v2.1.16

Balsamiq Mockups es una herramienta que permite realizar una representación esquemática de la solución que se llevará adelante, sin entrar en etapas posteriores como el diseño gráfico o la programación. Permite acordar con el cliente aspectos claves de la solución a desarrollar, como la distribución general de los elementos, sus jerarquías y la navegación de los mismos. Puede ser usada tanto por clientes como por desarrolladores. Los clientes pueden hacer uso de esta sin tener ningún tipo de conocimiento técnico especial. Gracias a ello, pueden comunicar de una manera más eficiente sus ideas y necesidades al grupo de trabajo que realiza los desarrollos técnicos. Los desarrolladores pueden usarlo con el mismo propósito, pero al revés. Para comunicar rápidamente las propuestas de solución, sin invertir demasiada cantidad de tiempo en esta primera etapa (41).

1.7.5 Lenguaje de programación

Un lenguaje de programación es cualquier lenguaje artificial que pueda utilizarse para definir una secuencia de instrucciones para su procesamiento por un ordenador (42). Puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente (43).

Python

Es un lenguaje de programación de alto nivel de propósito general. Su filosofía de diseño hace hincapié en un programador productivo y un código fácil de leer. Tiene una sintaxis central minimalista con unos pocos comandos básicos y semántica simple, pero también

tiene una amplia y completa biblioteca estándar, que incluye una API para trabajar con las funciones subyacentes de los sistemas operativos. El código Python, aun cuando es minimalista, define objetos incorporados como listas enlazadas, tuplas, y enteros arbitrariamente largos (44).

Tiene un sistema de tipado dinámico y un manejo automático de memoria usando conteo referencial. Permite mantener de forma sencilla la interacción con el sistema operativo y resulta adecuado para manipular archivos de texto (44). Es multiplataforma: el mismo código funciona en cualquier arquitectura, la única condición es que se disponga de un intérprete de Python para dicha arquitectura.

Características de Python (45)

- ✓ Es un lenguaje sencillo con una sintaxis elegante y fácil de entender muy parecida al pseudocódigo.
- ✓ La escritura de Python promueve el código limpio y legible.
- ✓ Es un lenguaje interpretado o de *scripts* lo cual permite que sea modular y que sea ejecutado en diferentes intérpretes para diferentes arquitecturas computacionales.
- ✓ Es rápido y sencillo aprender dicho lenguaje.
- ✓ Hay un intérprete interactivo que permite hacer pruebas rápidas de código.
- ✓ Las variables no tienen que ser definidas como de un solo tipo, lo que facilita el desarrollo y reduce las palabras necesarias para escribir un programa.
- ✓ Orientado a Objetos.

Kivy Language

El lenguaje Kivy (a veces llamado kvlang o kv), perteneciente al *framework* Kivy, permite crear su árbol de *widget* en forma declarativa y enlazar las propiedades de *widgets*, lo que permite realizar devoluciones de llamadas a un *widget*. Permite la creación de prototipos ágiles, así como rápidos cambios a la interfaz de usuario. También facilita una buena separación entre la lógica de la aplicación y su interfaz de usuario (46).

En el presente trabajo se utilizará Python en su versión 2.7 como lenguaje de programación debido a que el *framework* seleccionado utiliza dicho lenguaje. Además de Python, se hará uso de KV 1.7.1 para el desarrollo de la interfaz gráfica del sistema.

1.7.6 Bibliotecas

Para dar solución al objetivo propuesto se realizó un estudio de varias bibliotecas, las que se profundizan a continuación.

Jedi

Jedi es una herramienta de autocompletado de código para Python que se puede utilizar en IDE / editores. Comprende todos los elementos básicos de la sintaxis de Python, incluyendo muchas de las funciones con las que cuenta dicho lenguaje. Utiliza una API simple para conectar con el IDE (47).

Ventajas de Jedi

- ✓ Posee una buena y extensa documentación.
- ✓ Utiliza una API simple para conectar con el IDE.

PySmell

PySmell es una biblioteca de autocompletado de código para Python, destinada a ser conectada en diferentes editores de código. Se trata de analizar estáticamente el código fuente de Python y generar información sobre la estructura de un proyecto (archivo *TAGS*), que es utilizada por herramientas de un IDE, con el propósito de realizar el completamiento de código (48). Esta biblioteca tiene como principal desventaja el hecho de que se debe generar un archivo *TAGS* para la realización del autocompletado de código de cada sistema a desarrollar, lo que resulta en ocasiones lento y engorroso.

Biblioteca estándar de Python

La biblioteca estándar de Python es muy extensa, ofrece una amplia gama de instalaciones. Esta contiene módulos escritos en el lenguaje de programación C, que dan acceso a las funcionalidades del sistema, que de otro modo serían inaccesibles para los programadores de Python; así como módulos escritos en Python que proporcionan soluciones estandarizadas para muchos de los problemas que se producen en la programación diaria. Algunos de estos módulos están diseñados explícitamente para fomentar y mejorar la portabilidad de los programas de dicho lenguaje, esta biblioteca posee funcionalidades que pueden ser utilizadas en la realización del depurador del sistema a implementar, lo que permitirá saber si ciertas líneas de código son correctas o no, en caso de no serlo se especifica el tipo de error y el lugar donde se encontró el mismo (49).

Para la realización del resaltado de sintaxis y autocompletado de código fue seleccionada la biblioteca Jedi en su versión 0.8 debido a que la misma comprende todos los elementos básicos de la sintaxis de Python, incluyendo muchas de las funciones con las que cuenta dicho lenguaje. Además, permite buscar información acerca de los objetos de Python, como la cadena de documentación, argumentos de la función y ubicación de código. Mientras que para solucionar lo referente a la depuración se hará uso de la biblioteca estándar de Python.

1.7.7 Entorno de Desarrollo Integrado

En el presente epígrafe será realizado un estudio con el objetivo de seleccionar el IDE a utilizar para guiar el proceso de implementación del sistema.

Eclipse

La plataforma Eclipse consiste en un Entorno de Desarrollo Integrado abierto y extensible que puede ser utilizado como IDE para Python y cuenta con numerosas herramientas de desarrollo de *software*. También brinda soporte a otros lenguajes de programación, como son: C/C++, Cobol, Fortran, PHP o Java. A la plataforma base de Eclipse se le pueden añadir *plugins* para extender sus funcionalidades (50).

Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows realizado por Microsoft. Soporta múltiples lenguajes de programación tales como C++, C#, *Visual Basic*, Java, Python, Ruby y PHP. Se utiliza para desarrollar aplicaciones de consola y de interfaz gráfica de usuario. Admite *plugins* que mejoran la funcionalidad en casi todos los niveles, incluyendo la adición de soporte para los sistemas de control de código fuente y la adición de nuevos conjuntos de herramientas, como editores y diseñadores visuales para idiomas (51).

Por lo antes expuesto se puede decir que Visual Studio ofrece un entorno de desarrollo unificado para crear aplicaciones dirigidas a toda la gama de plataformas de Microsoft, como Windows Phone, dicho entorno tiene como principal limitante el hecho de ser un *software* propietario.

PyCharm

PyCharm es un entorno de desarrollo integrado de código abierto y multiplataforma desarrollado por JetBrains, que se utiliza para programar en Python. Proporciona un análisis de código, depuración gráfica, probador de unidad integrada y apoya el desarrollo web con Django (52). El mismo incluye heurísticas sofisticadas para determinar lo que cada tipo de variable es y proporcionar sugerencias de autocompletado para dichas variables. También se puede aprovechar la información en tiempo de ejecución cuando se ejecuta la aplicación con el depurador integrado (14).

Después de un análisis realizado fue seleccionado PyCharm en su versión 3.0.1 como entorno de desarrollo integrado a utilizar durante el proceso de implementación del sistema, ya que el mismo presenta un editor de código inteligente, que entiende los detalles específicos de Python y ofrece extraordinarios mejoradores de la productividad en cuanto a:

- ✓ finalización de código.
- ✓ refactorizaciones.

- ✓ importación automática.
- ✓ navegación de código con un solo clic.

1.8 Conclusiones parciales

En el presente capítulo fueron abordados un grupo de conceptos y definiciones con el propósito de hacer más fácil y comprensible el presente trabajo. Además, fue seleccionado Kivy como *framework* de desarrollo a utilizar en la realización del componente gráfico, dicho *framework* está implementado en Python, que es el lenguaje de programación que se utilizará para dar solución al objetivo general planteado. Se realizó un estudio de los elementos que componen un IDE así como las características que poseen los mismos, seleccionándose a PyCharm como entorno de desarrollo para guiar el proceso de implementación del sistema y fue seleccionada XP como metodología para guiar el proceso de desarrollo del *software* que se pretende obtener al concluir el presente trabajo.

Capítulo 2: Propuesta de solución

2.1 Introducción

El objetivo principal de este capítulo es describir las funcionalidades del sistema. Se abarcarán las tres primeras etapas de la metodología seleccionada (exploración, planificación e iteraciones) con el fin de conocer el alcance, estimar los tiempos de entrega de cada iteración y los artefactos que se generan. Se hará uso de las Tarjetas CRC (Contenido, Responsabilidad y Colaboración) con el objetivo de estructurar las clases y a su vez definir las responsabilidades sobre las mismas, lo que se traduce en un mejor diseño de la propuesta de solución. Además, se expondrán los patrones de diseño y la arquitectura a seguir en el proceso de diseño e implementación del sistema a desarrollar.

2.2 Propuesta de solución

La solución que se propone en el presente trabajo consta de un componente gráfico para el *framework* Kivy, que será instalado en dispositivos móviles con el objetivo de crear y probar aplicaciones de forma dinámica y eficiente en el propio dispositivo. Además, el sistema contará con las siguientes funcionalidades:

2.2.1 Insertar elementos propios del lenguaje de programación Python

El sistema permitirá insertar elementos propios del lenguaje de programación Python (clase, función, while, if, for, return, lista, diccionario, tupla y print) a través de una interfaz de usuario y un pre-visualizador. Para la realización de esta funcionalidad se hará uso de un botón perteneciente al *framework* Kivy, que permitirá mostrar las posibles opciones a insertar en el editor de código, dichas opciones serán mostradas haciendo uso de un popup, que es un componente propio de dicho *framework*. Una vez mostradas las posibles opciones a insertar, el usuario será el encargado de seleccionar el elemento que desea adicionar.

2.2.2 Depuración de código

El sistema permitirá realizar la depuración del código línea a línea, haciendo uso del modo de depuración completa, mediante la funcionalidad settrace del módulo sys de Python. Además, brindará la posibilidad de que el usuario visualice la línea que se está depurando en cada momento, mostrando en cada una de ellas un mensaje para saber si la línea es correcta, en caso de no serlo se detendrá el proceso de depuración, mostrando en el componente listview del *framework* Kivy el error ocurrido.

2.2.3 Completamiento de código

Una vez que el usuario escriba la primera letra o letras de una palabra, el sistema predice una o varias palabras posibles como opciones. Para ello será modificado el componente codeinput, perteneciente al *framework* Kivy, con el objetivo de implementar los eventos de teclado correspondientes a dicho componente. Estos eventos serán los encargados de solicitarle a Jedi que realice el completamiento de código, pasándole a esta el código previamente editado (code_input) y la posición (x, y) del cursor y Jedi a su vez devolverá una lista con las posibles palabras a completar, dicha lista será almacenada en la lista resultados, que estará compuesta por tuplas que contendrán el nombre y el tipo de símbolo, según el ámbito en el que se solicite el completamiento. Las posibles palabras a seleccionar por el usuario serán mostradas en una lista haciendo uso de un listview, perteneciente a dicho *framework*.

2.2.4 Ejecutar código

Con el uso de esta funcionalidad el usuario podrá visualizar en una ventana el resultado que muestra su código una vez ejecutado. Para ello se utilizará la función exec perteneciente a la biblioteca estándar de Python, a la misma se le pasará el código previamente editado. Una vez que ocurra un error el sistema detendrá la ejecución del código, informando el tipo de error y la línea donde ocurrió el mismo, dicho error será mostrado haciendo uso de un listview, que es un componente perteneciente al *framework* Kivy. Para mostrar la interfaz gráfica que desarrolle el usuario, se hará uso de la clase AppContainer del módulo appcontainer.

A continuación se muestra una figura donde se ilustra a grandes rasgos la propuesta de solución que plantea el presente trabajo.

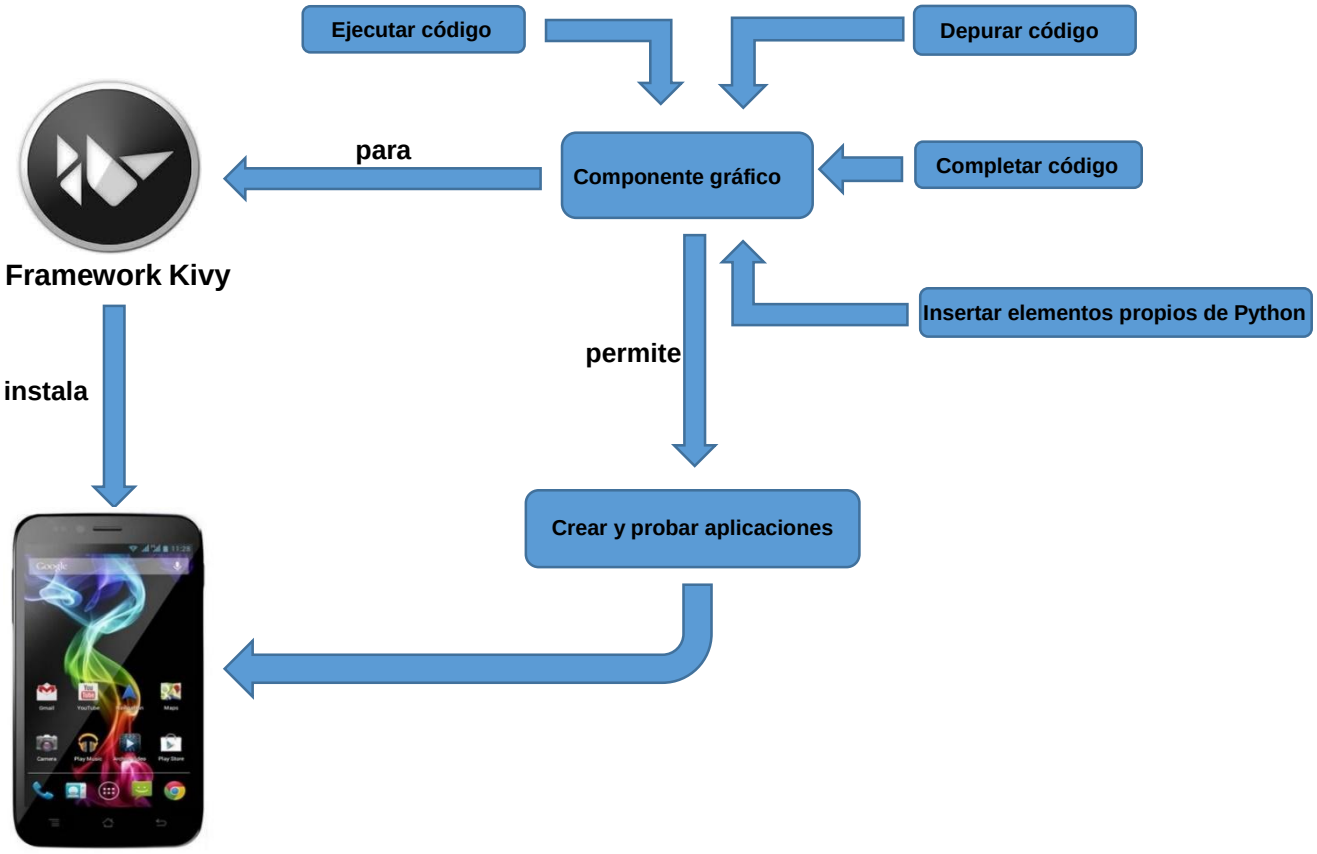


Figura 4: Propuesta de solución.

2.3 Persona relacionada con el sistema

Se define como persona relacionada con el sistema al usuario que interactúa con el mismo (cliente), con el objetivo de obtener un resultado específico. El componente no presenta restricciones de acceso a ciertas funcionalidades para un grupo específico de usuarios, sino que todos tienen los mismos privilegios sobre las funcionalidades internas de la aplicación. Estos usuarios deben poseer conocimientos previos de Kivy y Python.

Persona relacionada con el sistema	Descripción
Cliente	Interactúa con el sistema para ejecutar las funcionalidades del mismo.

Tabla 1. Definición de los usuarios del sistema.

2.4 Lista de reservas del producto

Según define la metodología XP, la lista de reservas del producto está compuesta por los requisitos no funcionales con que debe contar la aplicación para satisfacer con eficiencia las necesidades requeridas por el cliente. A continuación se muestra el listado de los mismos.

2.4.1 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estos requisitos en la aplicación han sido agrupados por categorías y se muestran a continuación.

RNF1: Usabilidad

- ✓ El sistema debe ser utilizado por usuarios con conocimientos previos de Kivy y Python.

RNF2: Portabilidad y operatividad

- ✓ El sistema debe ser compatible con el sistema operativo Android.

RNF3: *Hardware*

- ✓ Los dispositivos móviles que utilizarán el sistema deben tener: 256 megabyte (MB) de memoria RAM como mínimo.

RNF4: *Software*

- ✓ *Framework*: Kivy v1.7.1.

RNF5: Restricciones del diseño y la implementación.

- ✓ Como herramienta de modelado se empleará: Visual Paradigm v8.0.
- ✓ Como herramienta para el diseño de prototipos de interfaz de usuario se empleará: Balsamiq Mockups v2.1.16.
- ✓ Como lenguaje de programación se empleará: Python v2.7.
- ✓ Como biblioteca para el completamiento de código se utilizará: Jedi.
- ✓ Como biblioteca para depuración de código se empleará: La biblioteca estándar de Python.
- ✓ Como lenguaje para el diseño de la interfaz gráfica del sistema se utilizará: kv v1.7.1
- ✓ Como IDE se empleará: PyCharm.

2.5 Exploración

Como punto de partida en el ciclo de vida de un proyecto, XP propone la exploración como la primera fase en el proceso de construcción del producto a obtener. Este proceso se realiza mediante reuniones y tormentas de ideas obteniéndose como resultado las Historias de Usuario (HU) más significativas para la primera entrega del producto. De igual manera el equipo de desarrollo en este período se familiariza con las herramientas y tecnologías que serán utilizadas en el proyecto y se investigan las posibilidades de la arquitectura del sistema, obteniéndose como resultado un prototipo.

2.5.1 Historias de Usuario

Las Historias de Usuario constituyen una breve descripción de las funcionalidades del sistema. Estas son descritas por el cliente sin uso del lenguaje técnico y representan cada una de las principales características del proyecto. Las HU deben organizarse de

forma tal que prioricen las necesidades del cliente, permitiendo al equipo de trabajo identificar las más significativas a desarrollar durante el proceso de implementación de la solución. Se emplean para estimar el tiempo de vida del proyecto. Además, una de las mejores prácticas adoptadas en el desarrollo de *software* es la administración de requisitos. XP propone en este sentido hacer uso de las HU como técnica para especificar las funcionalidades que brindará el sistema y constituye una manera dinámica de realizar esta actividad, las HU son la base para las pruebas funcionales del sistema.

Las HU se clasifican según:

La prioridad en el negocio:

- ✓ **Alta:** Se le otorga a las HU que resultan funcionalidades fundamentales en el desarrollo del sistema.
- ✓ **Media:** Se le otorga a las HU que resultan para el cliente como funcionalidades a tener en cuenta, sin que estas tengan una afectación sobre el sistema que se esté desarrollando.
- ✓ **Baja:** Se le otorga a las HU que constituyen funcionalidades que sirven de ayuda al control de elementos asociados al equipo de desarrollo, a la estructura y no tienen relación con el sistema en desarrollo.

El riesgo en su desarrollo:

- ✓ **Alta:** Cuando en la implementación de la HU se considera la posible existencia de errores que conlleven a la inoperatividad del código.
- ✓ **Media:** Cuando pueden aparecer errores en la implementación de la HU que puedan retrasar la entrega de la versión.
- ✓ **Baja:** Cuando pueden aparecer errores que serán tratados con relativa facilidad, sin que traigan perjuicios para el desarrollo del proyecto.

Las HU son representadas mediante tablas divididas por las siguientes secciones:

- ✓ **Número:** Número de la HU que se describe.
- ✓ **Nombre de Historia de Usuario:** Identifica la HU que se describe entre los desarrolladores y el cliente.
- ✓ **Modificación de Historia de Usuario Número:** Cantidad de modificaciones realizadas a la HU.
- ✓ **Usuario:** Los programadores responsables de la historia de usuario.
- ✓ **Iteración Asignada:** Número de la iteración donde va a desarrollarse la HU.
- ✓ **Prioridad en el Negocio:** Se le otorga una prioridad (Alta, Media, Baja) a las HU de acuerdo con la necesidad de desarrollo.

- ✓ **Riesgo en Desarrollo:** Se le otorga una medida de (Alto, Medio, Bajo), a la ocurrencia de errores en el proceso de desarrollo de la HU.
- ✓ **Puntos Estimados:** Este atributo es una estimación hecha por el equipo de desarrollo del tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo. XP define una semana ideal como 5 días, trabajando 40 horas, es decir, 8 horas diarias.
- ✓ **Puntos Reales:** Representa el tiempo que se demoró en realidad el desarrollo de la HU.
- ✓ **Descripción:** Breve descripción de la HU.
- ✓ **Observaciones:** Señalamiento o advertencia del sistema.
- ✓ **Prototipo de Interfaz:** Interfaz que visualizará el usuario al interactuar con el sistema.

A continuación se exponen las HU definidas por el cliente en conjunto con el equipo de desarrollo.

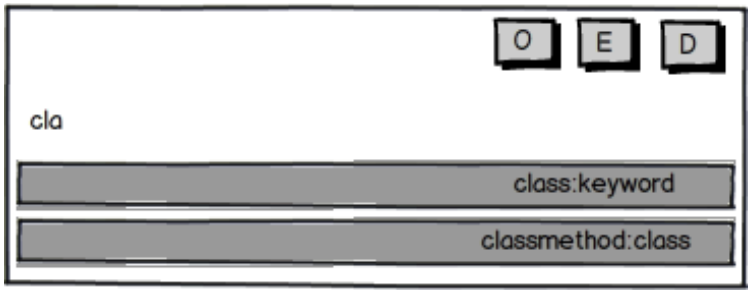
Historia de Usuario	
Número:1	**Nombre:** Completar código
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Luis Angel Rodríguez Martínez	**Iteración Asignada:** 1
Prioridad en el Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alta	**Puntos Reales:** 3
Descripción: Una vez que el usuario escriba la primera letra o letras de una palabra el sistema predice una o varias palabras como opciones, lo que le permite al usuario seleccionar una de estas, según le convenga.	
Observaciones:	
Prototipo: 	

Tabla 2. HU Completar código.

Historia de Usuario	
Número: 2	**Nombre:** Insertar elementos propios del lenguaje Python
Modificación de Historia de Usuario Número: 1	

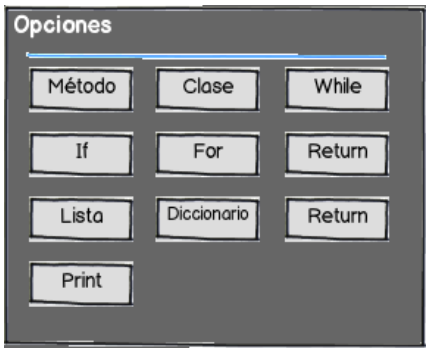
Usuario: Luis Angel Rodríguez Martínez	**Iteración Asignada:** 2
Prioridad en el Negocio: Media	**Puntos Estimados:** 1.5
Riesgo en Desarrollo: Media	**Puntos Reales:** 2
Descripción: Una vez que el usuario realice un toque sobre el botón O, son mostradas las posibles opciones a insertar, siendo el usuario el encargado de seleccionar el elemento que desea insertar. Después de seleccionado dicho elemento, se muestra el formulario correspondiente a la opción seleccionada, luego de ser llenado dicho formulario el sistema verifica que los datos introducidos sean correctos, en caso de serlo se inserta el elemento seleccionado, en caso contrario se muestra un mensaje informando la ocurrencia de un error.	
Observaciones:	
Prototipo: 	

Tabla 3. HU Insertar elementos propios del lenguaje Python.

Historia de Usuario	
Número: 3	**Nombre:** Ejecutar código
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Luis Angel Rodríguez Martínez	**Iteración Asignada:** 1
Prioridad en el Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alta	**Puntos Reales:** 3
Descripción: Una vez que el usuario realice un toque sobre el botón E, podrá visualizar en una ventana la salida que muestra su código una vez ejecutado. En caso de ocurrir un error se detiene la ejecución del código y se muestra el tipo y el lugar donde ocurrió dicho error.	
Observaciones:	
Prototipo:	

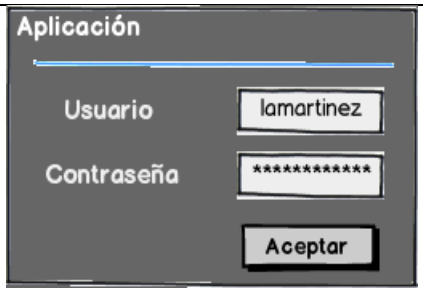


Tabla 4. HU Ejecutar código.

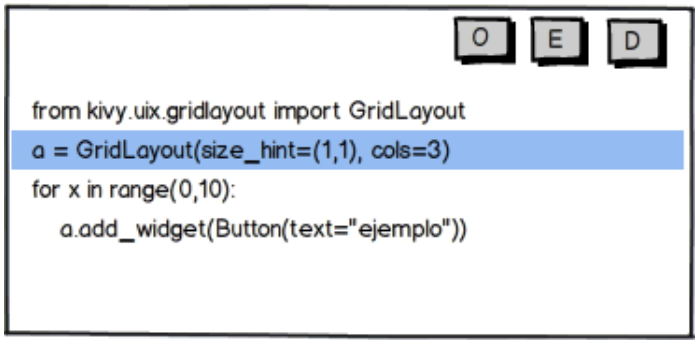
Historia de Usuario	
Número: 4	**Nombre:** Depurar código
Modificación de Historia de Usuario Número: 2	
Usuario: Luis Angel Rodríguez Martínez	**Iteración Asignada:** 2
Prioridad en el Negocio: Alta	**Puntos Estimados:** 3
Riesgo en Desarrollo: Alta	**Puntos Reales:** 3
Descripción: El sistema realiza la depuración del código línea a línea, haciendo uso del modo depuración completa, mediante el cual el depurador realiza el análisis del código previamente editado y se detiene solo en caso de existir algún tipo de error (informando el lugar donde ocurre el mismo) o después de analizar la última línea de código. Para el uso de esta funcionalidad el usuario debe realizar un toque sobre el botón D.	
Observaciones:	
Prototipo: 	

Tabla 5. HU Depurar código.

2.6 Planificación e Iteraciones

En el presente epígrafe serán abordadas las fases de Planificación e Iteraciones correspondientes a la metodología seleccionada, mostrándose los artefactos generados en cada una de estas fases.

2.6.1 Planificación

La fase de Planificación se realiza en pocos días, no siendo menos importante, ya que su objetivo principal es priorizar las HU. Los programadores estiman el esfuerzo necesario para desarrollar cada una de ellas, se define el cronograma y se acuerda el alcance de la entrega, el cual debe incluir varias iteraciones. El cliente prioriza las HU seleccionando cuáles se realizarán en cada iteración, logrando en la primera iteración un sistema con la arquitectura del proyecto. Otro aspecto a tener en cuenta durante esta fase es la velocidad del proyecto, la cual se estima utilizando el tiempo empleado en el desarrollo de las iteraciones terminadas.

2.6.2 Estimación del esfuerzo por HU

Las estimaciones del esfuerzo necesario para implementar las HU permiten tener una medida bastante real de la velocidad de progreso del proyecto y brindan una guía razonable a la cual ajustarse. Consiste en asignarle puntos a las HU y cada uno de estos puntos son equivalentes a una semana. Los programadores se basan para realizar la estimación en la complejidad que pueden tener las HU y en el tiempo hábil para el desarrollo de la aplicación.

No	Historias de Usuario	Puntos de estimación
1	Completar código	3
2	Insertar elementos propios del lenguaje Python	1.5
3	Ejecutar código	3
4	Depurar código	3

Tabla 6. Estimaciones del esfuerzo por HU.

2.6.3 Iteraciones

XP enfatiza en el carácter iterativo e incremental del desarrollo. Las iteraciones agrupan un conjunto de HU a implementar en un período de tiempo, generando al final de cada una un entregable funcional. La entrega con rapidez de módulos al cliente aumenta la retroalimentación y resulta más provechoso en términos de calidad del producto, que una entrega a largo plazo.

Las HU no cuentan con suficientes detalles como para permitir su análisis y desarrollo sin la presencia del cliente durante la fase. Por lo que se hace necesario al comienzo de cada iteración realizar las tareas necesarias de análisis en conjunto con cliente, consolidando los datos necesarios para la implementación. Culminar sin errores una iteración constituye un avance notable en el desarrollo del proyecto. Esta fase incluye

varias iteraciones sobre el sistema antes de ser entregado. Al final de la última iteración el sistema estará listo para entrar en producción.

2.6.4 Plan de iteraciones

Luego de identificar las HU y la estimación del esfuerzo por cada una de ellas, se procede a realizar el plan de iteraciones, en el cual estarán contenidas las HU en el orden a realizar por cada iteración, según su orden de relevancia y el total de semanas que durarán cada una de estas. A continuación se muestra el plan de iteraciones:

Iteración	Descripción de la iteración	HU a implementar	Duración total
1	En esta iteración se desarrollarán las HU de mayor importancia en el sistema a desarrollar. Al concluir el desarrollo de la misma se obtendrá una primera y aún incompleta versión del sistema, que ya puede ser probada.	HU.1 HU.3	6 semanas
2	Esta iteración tiene como principal objetivo realizar las HU de prioridad media. Al concluir se obtendrá una nueva versión que será el cliente el responsable de revisar y comunicar al equipo de desarrollo los cambios que son necesarios realizarle a la misma. Esta versión ya puede ser considerada un componente como tal y por tanto debe ser puesto en funcionamiento por un período determinado de tiempo y de esta forma ser probado por varios usuarios.	HU.4 HU.2	5 semanas

Tabla 7. Plan de iteraciones.

2.6.5 Plan de entregas

Es el resultado final de la fase de Planificación, acordado en una reunión entre todos los miembros del proyecto, estableciendo cuáles HU serán agrupadas para conformar una entrega del producto. Es un compromiso que establece el grupo de trabajo con el cliente. El plan de entregas es realizado en dependencia de las estimaciones de tiempos de desarrollo realizadas por los programadores. La estimación es uno de los temas más complicados en el desarrollo de un proyecto de *software*, es por ello que se recomienda luego de algunas iteraciones realizar nuevamente una reunión con los actores del

proyecto, para evaluar el plan de entregas y ajustarlo en caso de ser necesario. A partir del plan de iteraciones descrito anteriormente se procede a realizar el plan de entregas.

Iteración	Fecha de entrega
1	27 de abril de 2014
2	1 de junio de 2014

Tabla 8. Plan de entrega.

2.7 Arquitectura del sistema

La arquitectura de *software*, se relaciona con el diseño y la implementación de estructuras de *software* de alto nivel. Es el resultado de ensamblar cierto número de elementos arquitectónicos de forma adecuada para satisfacer las funcionalidades y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad y disponibilidad. Para el diseño del sistema propuesto se determinó utilizar el patrón arquitectónico: Arquitectura basada en componentes.

Este patrón arquitectónico describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver un problema. Se utiliza para diseñar aplicaciones a partir de componentes individuales, enfatiza en la descomposición del sistema en componentes lógicos o funcionales y define una aproximación al diseño a través de componentes que se comunican mediante interfaces que contienen métodos, eventos y propiedades.

Para la realización del sistema se diseñaron los siguientes componentes:

- ✓ El componente **completioncodeinput** se relaciona con los componentes **options**, **debugger** y **codeinput** a través de interfaces que contienen los métodos y eventos necesarios para la depuración, completamiento y edición de código.
- ✓ El componente **options** se comunica con los componentes **classs**, **tuple**, **iff**, **returnn**, **method**, **while**, **dictionary**, **forr**, **list** y **printt** mediante una interfaz que posee métodos y eventos encargados de crear instancias de cada uno de estos, lo que permite mostrar los posibles elementos propios del lenguaje Python a insertar.
- ✓ El componente **codeinput** utiliza la biblioteca **Jedi** para la obtención de una lista con las posibles palabras a completar. Para ello se hace uso de una interfaz propia de dicha biblioteca.

El uso de este patrón facilita el despliegue, pues permite sustituir un componente por su nueva versión sin afectar a otros componentes o al sistema y favorece la reusabilidad

de los componentes independientes del contexto, permitiendo que se empleen en otras aplicaciones y sistemas. A continuación se muestra un diagrama de componentes donde se ejemplifica la arquitectura del sistema.

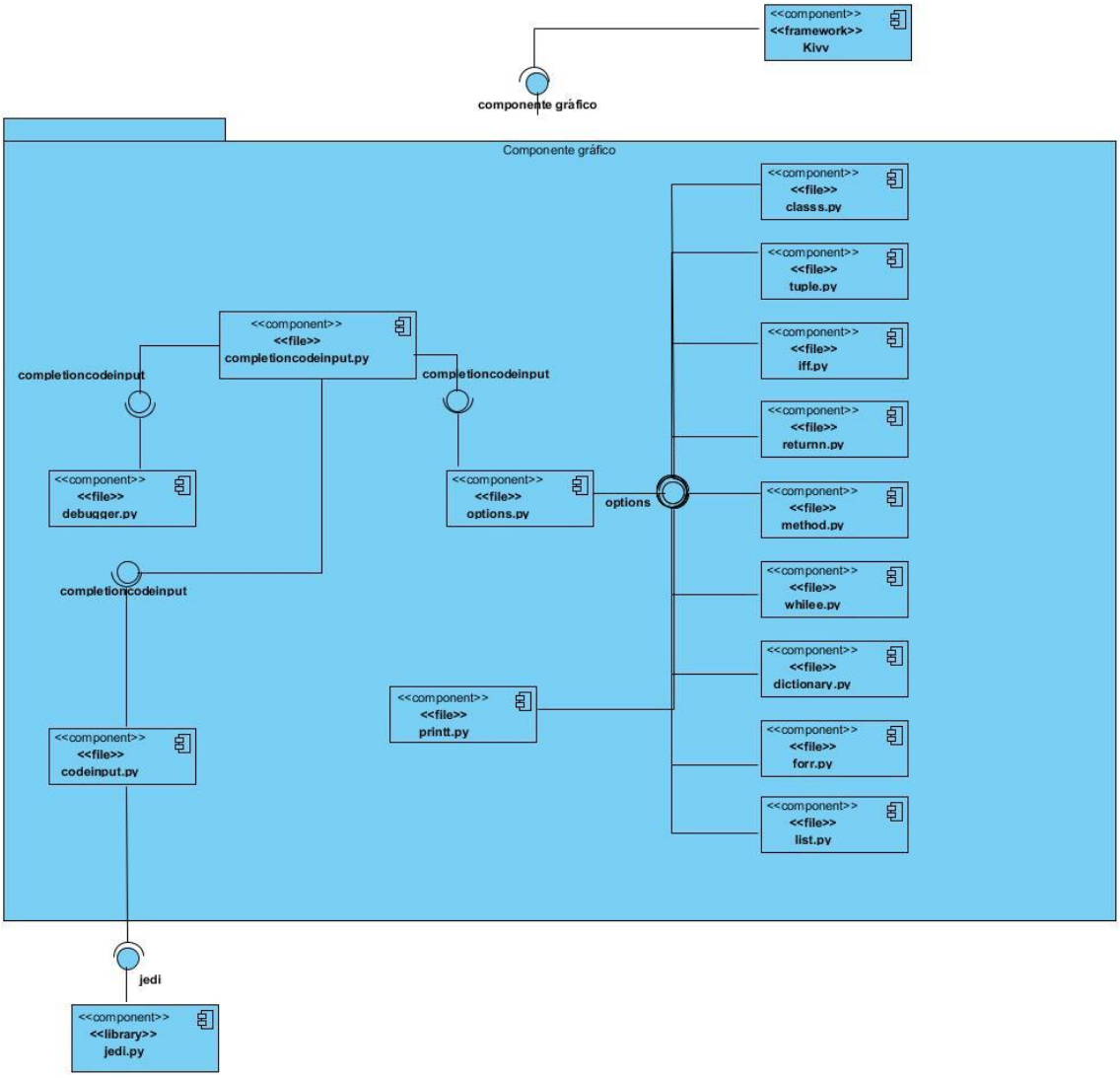


Figura 5. Diagrama de componentes.

2.8 Diseño de la propuesta de solución

La metodología XP, no requiere la descripción del sistema mediante diagramas de clase utilizando la notación UML, sino que en su lugar se guía por técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración). Esto no implica que no se utilicen dichos diagramas para obtener una mejor visión y comunicación entre el equipo de trabajo, siempre y cuando no posea una alta complejidad y defina información importante.

2.8.1 Tarjetas CRC.

Una tarjeta CRC no es más que una ficha que representa a una entidad del sistema. Estas se utilizan para estructurar las clases y a su vez definir las responsabilidades

sobre las mismas, así como la simulación de escenarios en el sistema, lo que brinda una ayuda al equipo de trabajo durante el diseño e implementación del sistema.

Clase: Nombre de la clase que se está modelando.	
Súper clase: Nombre de la clase padre en la herencia.	
Sub Clase(s): Nombre de la(s) clase(s) hija en la herencia.	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase.	**Colaboraciones:** Indica con cuáles clases se requiere relación para cumplir la responsabilidad.

Tabla 9. Plantilla para las tarjetas CRC.

A continuación se definen algunas de las tarjetas CRC del sistema, las restantes tarjetas CRC correspondientes al presente trabajo se encuentran en el anexo 3.

Clase: CodeInput	
Súper clase: CInput	
Sub Clase(s):	
Responsabilidades: __init__(self, **kwargs) on_selection_change(self, *args) update(self, data) get_text(self) check_enter(self) check_cursor(self, internal_action) update_list_view(self, init = " ") _completions(self, *args) get_last_word(self) _keyboard_on_key_down(self, window, keycode, text, modifiers) _key_down(self, key, repeat=False) on_touch_down(self, touch)	**Colaboraciones:** CompletionCodeInput

Tabla 10. Tarjeta CRC CodeInput.

Clase: CompletionCodeInput.	
Súper clase: FloatLayout	
Sub Clase(s):	
Responsabilidades: __init__(self, **kwargs) show_error_line(self, values) key_down(self, key, val) run(self, btn) select_lines(self, number) debugger(self, btn) show_options(self, a) timeD(self, time) hide_button(self, b)	**Colaboraciones:** TestApp

Tabla 11. Tarjeta CRC CompletionCodeInput.

Clase: TestApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: make_ics(self) build(self)	**Colaboraciones:** CompletionCodeInput.

Tabla 12. Tarjeta CRC TestApp.

Clase: Options.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self, code_input, **kwargs) close_popup(self, *largs, **kwargs)	**Colaboraciones:** CompletionCodeInput.

Tabla 13. Tarjeta CRC Options.

2.8.2 Patrones de diseño

Son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de *software*, permiten establecer un vocabulario común de diseño, cambiando el nivel de abstracción a colaboraciones entre clases y permitiendo comunicar experiencia sobre dichos problemas y soluciones (53).

Los patrones de diseño han sido propuestos en varios escenarios como una vía para comunicar de forma compacta el problema a resolver, capturando su esencia y la de su solución; sus componentes son las clases y objetos, así como los mecanismos de interacción son los mensajes.

Un patrón de diseño es (54)**:**

- ✓ Una solución estándar para un problema común de programación.
- ✓ Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- ✓ Un proyecto o estructura de implementación que logra una finalidad determinada.
- ✓ Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- ✓ Conexiones entre componentes de programas.

Patrones GRASP (Patrones Generales de *Software* para Asignar Responsabilidades): Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. En la implementación de la aplicación se utilizaron los siguientes:

Patrón Experto: Un experto es una clase que tiene toda la información necesaria para implementar una responsabilidad. La respuesta es asignar la responsabilidad a la clase que contenga la información necesaria para cumplir la responsabilidad, o sea, la clase debe ser la experta en la información. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos) (55). Por ejemplo en la aplicación este patrón se evidencia en la clase CodeInput. Esta clase tiene información sobre las entradas de teclado, siendo la responsable de realizar el completamiento de código.

Patrón Creador

Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. Indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear algo contenido o registrado (55). Por ejemplo en la aplicación se evidencia este patrón en la clase Options, que es la responsable de crear instancias de las opciones a insertar.

Patrón Alta Cohesión: Pretende que la información almacenada en una clase debe ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. Las clases tienen responsabilidades moderadas y colaboran con otras clases para llevar a cabo las tareas. Una clase con alta cohesión posee un número relativamente pequeño de métodos, con funcionalidades altamente relacionadas (55). Si la tarea es extensa se apoya en otros objetos. Es relativamente fácil de mantener, entender y reutilizar. Por ejemplo este patrón se evidencia en la clase KivyCodeTracer. Esta clase posee un número pequeño de métodos con funcionalidades altamente relacionadas con la depuración del código.

Patrón Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está relacionada a otras clases. Una clase con bajo o débil acoplamiento no depende de muchas otras. Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (55). La solución es asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases posibles, sin comprometer la funcionalidad. Por ejemplo en la aplicación la

clase ForApp solo se relaciona con la clase For. Esto permite que en caso de producirse alguna modificación en la clase ForApp, solo se afecte la clase For.

2.9 Conclusiones parciales

Con la culminación de este capítulo se logró el análisis de las principales características que debe cumplir el sistema, quedando definidas un total de 4 HU generadas a partir de la metodología seleccionada. Se precisó la prioridad de cada una de las HU que describen las funcionales del sistema, puntualizando el orden de su implementación y las iteraciones en que serán implementadas. Se realizó la planificación del *software* de manera exitosa, lo que posibilitará la entrega del producto al cliente en el tiempo acordado. Además, se identificaron los patrones de diseño Experto, Creador, Alta Cohesión y Bajo Acoplamiento como una buena práctica de programación que facilita la implementación y soporte del sistema.

Capítulo 3: Implementación y prueba

3.1 Introducción

En el presente capítulo se describe el proceso de implementación correspondiente a la fase de iteraciones. Además, se exponen los estándares de codificación a utilizar para el desarrollo de la solución y se identifican y describen las tareas de ingeniería. Son realizadas las pruebas pertenecientes a la etapa de producción, con el objetivo de validar las historias de usuario definidas por el cliente.

3.2 Tareas de la ingeniería

Para realizar una correcta implementación de las HU deben ser definidas por parte del equipo de desarrollo las Tareas de Ingeniería (TI) que se realizarán en cada una de las iteraciones, estas tareas definen las actividades que dan cumplimiento a cada HU. Las TI también conocidas como tareas de implementación permiten a los desarrolladores obtener un nivel de detalle más avanzado con respecto a las HU. Estas se realizan con el objetivo de resolver las HU, las que pueden tener una o más tareas de ingeniería en dependencia de la complejidad de la funcionalidad a desarrollar.

El desarrollo del *software* se planificó en dos iteraciones de trabajo. En el sistema propuesto se identificaron las siguientes tareas de la ingeniería por cada iteración:

3.2.1 Iteración 1

Tarea de Ingeniería	
Número de la Tarea: 1	**Número de la HU:** 1
Nombre de la Tarea: Modificar el evento _key_down del componente codeinput.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 17/03/2014	**Fecha Fin:** 23/03/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Comprobar los caracteres de entrada (espacio, punto e inicio de línea) para ejecutar el completamiento de código, haciendo uso de la función Script de la biblioteca Jedi.	

Tabla 14. Tarea No. 1 de la HU No.1.

Tarea de Ingeniería	
Número de la Tarea: 2	**Número de la HU:** 1
Nombre de la Tarea: Modificar el evento _key_up del componente codeinput.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 24/03/2014	**Fecha Fin:** 30/03/2014

Programador Responsable: Luis Angel Rodríguez Martínez
Descripción: Comprobar la combinación de caracteres (Control + Espacio) para ejecutar el completamiento de código, ejecutando la función Script de la biblioteca Jedi.

Tabla 15. Tarea No. 2 de la HU No.1.

Tarea de Ingeniería	
Número de la Tarea: 3	**Número de la HU:** 1
Nombre de la Tarea: Implementar la funcionalidad get_last_word en el componente codeinput.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 31/03/2014	**Fecha Fin:** 6/04/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Obtener en la línea de código que se esté editando la última palabra. Para implementar esta funcionalidad es necesario extraer todos los caracteres que se encuentren entre el penúltimo carácter de espacio, punto o inicio de línea y el carácter de espacio final o fin de línea.	

Tabla 16. Tarea No. 3 de la HU No.1.

Tarea de Ingeniería	
Número de la Tarea: 1	**Número de la HU:** 3
Nombre de la Tarea: Implementar método execute en la clase CompletionCodeInput.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 7/04/2014	**Fecha Fin:** 27/04/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Implementar el método execute, haciendo uso de función exec de la biblioteca estándar de Python. Se debe capturar el error de la ejecución del código en caso de que exista y mostrarse a través de un componte listview propio de Kivy.	

Tabla 17. Tarea No. 1 de la HU No.3.

3.2.2 Iteración 2

Tarea de Ingeniería	
Número de la Tarea: 1	**Número de la HU:** 4
Nombre de la Tarea: Implementar el método debugger en la clase CompletionCodeInput.	

Tipo de Tarea: Desarrollo	
Fecha Inicio: 28/04/2014	**Fecha Fin:** 18/05/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Implementar el método debugger haciendo uso de la función settrace del módulo sys de Python. Capturar las trazas de ejecución y los errores. Luego mostrar las líneas en ejecución a través de un tiempo de dos segundos, mediante el uso del componente clock de Kivy.	

Tabla 18. Tarea No. 1 de la HU No.4.

Tarea de Ingeniería	
Número de la Tarea: 1	**Número de la HU:** 2
Nombre de la Tarea: Implementar las clases correspondientes a los elementos propios de Python a insertar.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 19/05/2014	**Fecha Fin:** 25/05/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Implementar las clases (Class, Method, While, If, For, Return, List, Dictionary, Tuple y Print). Estas clases se encargarán de validar la estructura propia de cada uno de estos elementos y si no existen errores entonces se insertará el elemento en el componente completioncodeinput. En caso contrario se notificará el error de validación al usuario a través de un componente popup.	

Tabla 19. Tarea No. 1 de la HU No.2.

Tarea de Ingeniería	
Número de la Tarea: 2	**Número de la HU:** 2
Nombre de la Tarea: Implementar el módulo options.	
Tipo de Tarea: Desarrollo	
Fecha Inicio: 26/05/2014	**Fecha Fin:** 1/06/2014
Programador Responsable: Luis Angel Rodríguez Martínez	
Descripción: Implementar el módulo options, que se encargará de crear instancias de los elementos (clase, método, while, if, for, return, lista, diccionario, tupla y print) propios del lenguaje Python a insertar.	

Tabla 20. Tarea No. 2 de la HU No.2.

3.3 Estándares de codificación

Los estándares de codificación, también conocidos como estilos de programación o convenciones de código, son convenios para escribir código fuente en ciertos lenguajes de programación. Estos estándares facilitan el mantenimiento del código y sirven como punto de referencia para los programadores. Por lo que para lograr un mayor entendimiento del código a la hora de reutilizar algún componente del presente trabajo, se utilizarán los siguientes estándares de codificación.

3.3.1 Sangría

Se usarán cuatro espacios para cada nivel de sangría.

3.3.2 Tabuladores y espacios

La separación del código Python se realizará a través de espacios, tabulaciones o combinaciones de ambos.

3.3.3 Importaciones

Las importaciones usualmente deben estar en líneas diferentes, siempre se ponen en la parte superior del archivo, justo después de cualquier comentario del módulo y antes de cualquier módulo global:

- ✓ **from** kivy.uix.button **import** Button
- ✓ **from** kivy.uix.gridlayout **import** GridLayout

3.3.4 Espacios en blanco en expresiones y declaraciones

Evitar espacios en blanco en las situaciones siguientes:

Inmediatamente después de: paréntesis, corchetes o llaves.

- ✓ Si: self.__name = TextInput(multiline = False)
- ✓ No: self.__name = TextInput(multiline = False)

Inmediatamente antes de: dos puntos, coma o punto y coma.

- ✓ Si: **def** debugger(self, btn):
- ✓ No: **def** debugger(self , btn) :

Inmediatamente antes de: la apertura de paréntesis que inicializan la lista de argumentos de una llamada de función.

- ✓ Si: IfApp(App)
- ✓ No: IfApp (App)

Inmediatamente antes de: la apertura de paréntesis que inician un índice.

- ✓ Si: key_down(self, key, val)
- ✓ No: key_down (self, key, val)

Más de un espacio alrededor de una asignación.

- ✓ Si: x = Options(self.code_input)
- ✓ No: x = Options(self.code_input)

3.3.5 Comentarios

- ✓ Siempre se deben actualizar los comentarios cuando el código es cambiado.
- ✓ Los comentarios deben ser oraciones completas.
- ✓ Si el comentario es una frase o una oración, su primera palabra debe comenzar con mayúscula, a menos que sea un identificador que empiece con una letra minúscula.
- ✓ Si un comentario es corto, el punto al final puede ser omitido.
- ✓ Cada línea del bloque de comentario empieza con el signo # y un espacio simple.

3.3.6 Las cadenas de documentación

Se deben escribir cadenas de documentación para todas funciones, este comentario debe aparecer después de la línea “def”.

- ✓ **def** validate(self, some):
  “”” Es la función encargada de validar cada uno de los campos del formulario. “””

3.3.7 Estilos de nombres recomendados

Existen diferentes estilos de nombres, a continuación se detallarán cada uno de los estilos que se utilizarán en la elaboración del componente gráfico a desarrollar. Estos estilos cumplen los estándares internacionales de codificación para el lenguaje de programación Python.

Nombres de módulos

Los módulos deben tener nombres cortos y en minúsculas, sin subrayado (*underscore*).

- ✓ Si: main.py
- ✓ No: Main.py

Nombres de clases

Para nombrar las clases se debe usar el siguiente formato:
La primera letra de cada palabra debe comenzar con una letra mayúscula. Si el nombre de la clase es una combinación de palabras no se debe usar subrayado (*underscore*).

- ✓ Si: ForApp(App)
- ✓ No: For_App(App)

Nombres de funciones:

Los nombres de las funciones deben ser en minúsculas, con palabras separadas por subrayado como sea necesario para mejorar la legibilidad.

- ✓ Ejemplo: **def** key_down(self, key, val)

Nombres de variables:

Para evitar nombres que choquen con subclases se utiliza el subrayado, se utilizan dos subrayados al inicio de la variable para evitar este problema.

- ✓ __author__ = 'Luis Angel'

3.3.8 Nombramientos a evitar

Nunca se deben utilizar las letras "l", "o" u "O" como nombres simples de variables, en algunos tipos de fuentes estas letras no se distinguen de los números 1 y 0, cuando se tenga que utilizar la letra "l" minúscula, se debe sustituir por la "L".

3.4: Pruebas

El proceso de pruebas es uno de los pilares fundamentales de la metodología XP, las mismas ayudan al cliente a verificar y concretar las funcionalidades de las HU, lo que permite fortalecer la comunicación entre el cliente y el equipo de desarrollo. Esta filosofía permite aumentar la calidad de los sistemas, reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permiten aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar codificaciones y refactorizaciones. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones. (56)

XP divide las pruebas en dos grupos: pruebas unitarias y pruebas de aceptación. Las pruebas unitarias son desarrolladas por los programadores y se encargan de verificar el código automáticamente y las pruebas de aceptación están destinadas a verificar que al final de cada iteración las HU cumplen con la funcionalidad asignada y satisfagan las necesidades del cliente. Las pruebas de aceptación son más importantes que las pruebas unitarias dado que significan la satisfacción del cliente con el producto desarrollado, por esto el cliente es la persona adecuada para diseñar las pruebas de aceptación.

3.4.1 Pruebas unitarias

Uno de los niveles utilizados para realizar pruebas de *software* en la metodología XP son las pruebas unitarias. Estas consisten en hacer pruebas en pequeños fragmentos del código de la aplicación. Estos fragmentos deben ser unidades estructurales del programa encargados de una tarea específica, en programación orientada a objetos se

puede afirmar que estas unidades son los métodos o las funciones que se tienen definidas. El objetivo de estas pruebas es el aislamiento de partes del código y la demostración de que no contienen errores. Estas no generan artefactos y no son directamente palpables para el cliente (56).

Resultados de las pruebas unitarias

Las pruebas unitarias fueron desarrolladas constantemente cada vez que se implementaba alguna de las funcionalidades, probándola directamente en el entorno real, lo que posibilitó una correcta implementación de todas las funcionalidades del sistema y con ello la terminación de forma exitosa del sistema a desarrollar. A continuación se ilustra un ejemplo de una prueba unitaria realizada al método get_last_word.

```
class TestCodeInput(TestCase):
    def setUp(self):
        self.codeinput = CodeInput()
    def test_get_last_word(self):
        self.codeinput.text = "import andr"
        word = self.codeinput.get_last_word()
        self.assertEqual(word, "andr")

        self.codeinput.text = "from kivy impor     "
        word = self.codeinput.get_last_word()
        self.assertEqual(word, "impor")

        self.codeinput.text = "from kivy import"
        word = self.codeinput.get_last_word()
        self.assertEqual(word, "import")

        self.codeinput.text = "class MiClase:"
        word = self.codeinput.get_last_word()
        self.assertEqual(word, "MiClase:")

if __name__ == '__main__':
    unittest.main()
```

```
C:\Windows\System32\cmd.exe
[INFO              ] [GL          ] Texture max size <163
[INFO              ] [GL          ] Texture max units <32
[DEBUG             ] [Shader      ] Fragment compiled suc
[DEBUG             ] [Shader      ] Vertex compiled succe
[DEBUG             ] [ImagePygame ] Load <d:\Software\@De
raries\Kivy\kivy\kivy\data\glsl\default.png>
[INFO              ] [Window      ] virtual keyboard not
ot docked
[DEBUG             ] [Atlas       ] Load <d:\Software\@De
raries\Kivy\kivy\kivy\data\..\data\images\defaulttheme.at
[DEBUG             ] [Atlas       ] Need to load 1 images
[DEBUG             ] [Atlas       ] Load <d:\Software\@De
raries\Kivy\kivy\kivy\data\..\data\images\defaulttheme-0.
[DEBUG             ] [ImagePygame ] Load <d:\Software\@De
raries\Kivy\kivy\kivy\data\..\data\images\defaulttheme-0.
[INFO              ] [GL          ] NPOT texture support
.
----------------------------------------------------------------------
Ran 1 test in 2.227s

OK

c:\Users\kpiorno\Desktop\Test\@Tesis\tesis>
```

Figura 6. Ejemplo de una prueba unitaria.

3.4.2 Pruebas de aceptación

El objetivo de estas pruebas es verificar los requisitos, por tal motivo, los requisitos del sistema son la principal fuente de información a la hora de construir las pruebas de aceptación.

Estas son realizadas a partir de las HU. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. El cliente o usuario especifica los aspectos a validar cuando una historia de usuario ha sido correctamente implementada (56).

Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento y no se considera completa

hasta que no supera sus pruebas de aceptación. Esto significa que debe desarrollarse un nuevo *test* de aceptación para cada iteración o se considerará que el equipo de desarrollo no realiza ningún progreso. Es responsabilidad del cliente verificar la corrección de las pruebas de aceptación y tomar decisiones acerca de las mismas. La garantía de la calidad es una parte esencial en el proceso de XP (56).

La realización de este tipo de pruebas y la publicación de los resultados se deben realizar lo más rápido posible, para que los desarrolladores puedan realizar con mayor rapidez los cambios que sean necesarios. Un sistema está completamente aceptable cuando quedan satisfechos todos los requisitos funcionales especificados por el cliente, teniendo en cuenta además los requisitos no funcionales.

A continuación, aparecen las pruebas de aceptación realizadas a la propuesta de solución por cada iteración:

Iteración 1

Caso de prueba de aceptación	
Código:1	**Historia de Usuario:** Completar código
Nombre de la persona que realiza la prueba: Luis A. Rodríguez Martínez	
Descripción de la prueba: Permite que el usuario realice el completamiento de código. Una vez que el usuario escriba la primera letra o letras de una palabra el sistema predice una o varias palabras como opciones, siendo el usuario el encargado de seleccionar la palabra que desea utilizar.	
Condiciones de ejecución: El usuario debe realizar un toque sobre pantalla para que se muestre el teclado del dispositivo.	
Entrada/Pasos de ejecución: **1.** El usuario escribe una o varias letras de la palabra que desea escribir. **2.** En caso de ser parte de una de las palabras reservadas pertenecientes al lenguaje de programación Python, se muestra una lista con las posibles opciones a seleccionar por el usuario. **3.** Se muestra en el componente completioncodeinput la opción seleccionada por el usuario.	
Resultado esperado: Se muestra en el componente completioncodeinput la opción seleccionada por el usuario.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 21. Prueba de aceptación para la HU No.1.

Caso de prueba de aceptación	
Código:2	**Historia de Usuario:** Ejecutar código

Nombre de la persona que realiza la prueba: Luis A. Rodríguez Martínez
Descripción de la prueba: Ejecuta el código previamente definido por el usuario, mostrando en una ventana la salida que brinda la ejecución de dicho código.
Condiciones de ejecución: El usuario debe realizar un toque sobre el botón E.
Entrada/Pasos de ejecución: 1. El usuario escribe el código que desea ejecutar. **2.** El usuario realiza un toque sobre el botón E.
Resultado esperado: Se ejecuta el código previamente definido por el usuario y el resultado de su ejecución se muestra en una ventana. En caso de que ocurra un error, se detiene la ejecución del código y se muestra el tipo de error y el lugar donde ocurrió el mismo.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 22. Prueba de aceptación para la HU No.3.

Iteración 2

Caso de prueba de aceptación	
Código: 3	**Historia de Usuario:** Depurar código
Nombre de la persona que realiza la prueba: Luis A. Rodríguez Martínez	
Descripción de la prueba: Permite al usuario depurar el código previamente editado, realizando un seguimiento de código línea a línea, haciendo uso del modo depuración completa.	
Condiciones de ejecución: El usuario debe realizar un toque sobre el botón D.	
Entrada/Pasos de ejecución: 1. El usuario escribe en el componente completioncodeinput el código que desea depurar. **2.** El usuario realiza un toque sobre el botón D.	
Resultado esperado: Se realiza la depuración del código previamente editado.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 23. Prueba de aceptación para la HU No.4.

Caso de prueba de aceptación	
Código: 4	**Historia de Usuario:** Insertar elementos propios del lenguaje Python.
Nombre de la persona que realiza la prueba: Luis A. Rodríguez Martínez	
Descripción de la prueba: Permite que el usurario seleccione el elemento que desea insertar en el componente completioncodeinput.	
Condiciones de ejecución: El usuario debe realizar un toque sobre el botón O.	

Entrada/Pasos de ejecución: **1.** El usuario selecciona una de las posibles opciones a insertar. **2.** El usuario completa cada uno de los campos pertenecientes a la opción seleccionada. **3.** El usuario realiza un toque sobre el botón aceptar.
Resultado esperado: Se inserta en el componente completioncodeinput la opción seleccionada por el usuario.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 24. Prueba de aceptación para la HU No.2

3.4.3 Resultados obtenidos

Las pruebas aplicadas al sistema se desarrollaron en dos iteraciones. Se realizaron dos casos de prueba por cada iteración, lo que arroja un total de cuatro casos de prueba de aceptación. Se identificaron un total de siete no conformidades (NC). A continuación se muestra una gráfica que resume las iteraciones realizadas y los tipos de NC encontradas.

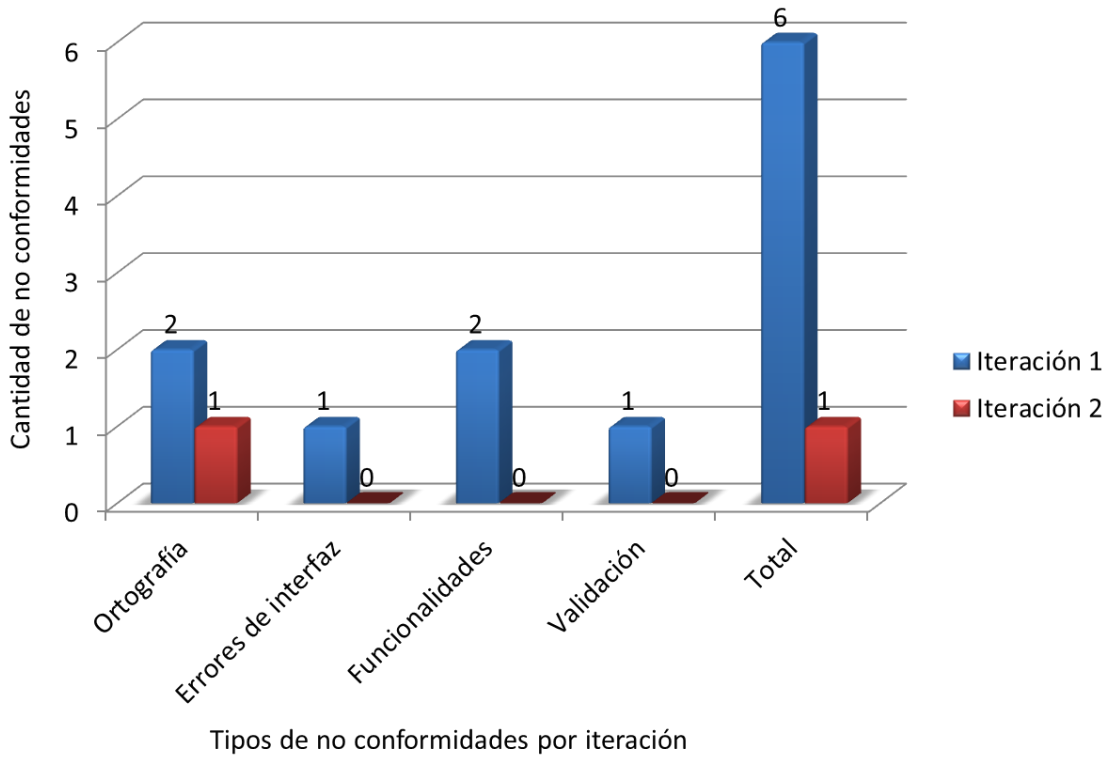


Figura 7: Tipos de NC por iteración.

Cada una de las NC identificadas fue solucionada en su totalidad, influyendo así en el mejoramiento de la propuesta de solución, lo que se traduce en la obtención de un componente gráfico libre de errores y la satisfacción de las necesidades del cliente.

3.5 Conclusiones parciales

En el presente capítulo fueron abordados los temas referentes a la implementación de la propuesta de solución y la estrategia de pruebas a seguir durante la elaboración del sistema. Además, se realizaron cuatro historias de usuario correspondientes al trabajo de diploma, quedando reflejadas en siete tareas de ingeniería. Fue definido el estándar de codificación a emplear para la implementación de la aplicación, permitiendo la correcta uniformidad y organización del código. Se realizaron las pruebas pertinentes (unitarias y de aceptación), para garantizar la obtención de un producto con la calidad esperada. Con la culminación de este capítulo se considera terminada la propuesta de solución del sistema.

Conclusiones generales

Después de concluida la investigación y ser analizados los resultados obtenidos se arribó a las siguientes conclusiones:

- ✓ Se obtuvo una visión general de la edición de código en los dispositivos móviles, a través un estudio realizado a los sistemas similares.
- ✓ La utilización de las herramientas y lenguajes definidos en el transcurso de la investigación permitieron el desarrollo del componente gráfico.
- ✓ La solución desarrollada arrojó como resultado un componente gráfico para el *framework* Kivy, que permite la depuración, completamiento y edición de código en dispositivos móviles.
- ✓ Se comprobó que la propuesta de solución desarrollada responde a los requisitos funcionales descritos, evaluando los resultados alcanzados a través de las pruebas realizadas.

Recomendaciones

Una vez concluido el desarrollo del presente trabajo se recomienda:

- ✓ Desarrollar una funcionalidad que permita mostrar la información resultante de la ejecución de código Python en la consola.
- ✓ Adicionar nuevas funcionalidades al sistema que permitan dar soporte a otros lenguajes de programación.

Referencias bibliográficas

1. **Soriano, Anaid Guevara.** Dispositivos Móviles. [En línea] 06 de 08 de 2010. [Citado el: 29 de 01 de 2014.] http://revista.seguridad.unam.mx/numero-07/dispositivos-m%C3%B3viles.

2. **Moro, César Tardáguila.** Dispositivos móviles y multimedias. [En línea] [Citado el: 17 de 06 de 2014.] openaccess.uoc.edu/webapps/o2/bitstream/10609/9164/1/dispositivos_moviles_y_multimedia.pdf.

3. Supervisión de la tecnología: Las aplicaciones móviles alcanzan un nuevo hito. [En línea] 12 de 08 de 2009. [Citado el: 11 de 12 de 2013.] http://www.itu.int/net/itunews/issues/2009/06/04-es.aspx.

4. EcuRed. EcuRed. [En línea] [Citado el: 01 de 01 de 2014.] http://www.ecured.cu/index.php/Framework.

5. Definicion de Depuración - ¿qué es Depuración. [En línea] [Citado el: 10 de 12 de 2013.] http://www.alegsa.com.ar/Dic/depuracion.php.

6. Kyle I. Murray, Jeffrey P. Bigham. *Beyond Autocomplete: Automatic Function Definition.* Department of Computer Science, University of Rochester.

7. Yurenia Hernández Blanco, Eduardo Alejandro Cuesta Llanes. *Módulo para la edición y depuración de código Octave integrado al ambiente EIDMAT.* Ciudad de la Habana : s.n., 2010.

8. Desarrollo, Entornos de. *Entornos de Desarrollo Integrado.*

9. Mora, Oscar Mauricio Caravaca. *Entornos de desarrollo integrados.* Instituto Tecnológico de Costa Rica. 2010.

10. Blanco, Carlos. Entornos de Desarrollo Integrado(IDE). [En línea] 2011. [Citado el: 21 de 1 de 2014.] http://carlosblanco.pro/category/prog-apli/.

11. Bolton, David. About.com. [En línea] [Citado el: 9 de 1 de 2014.] http://cplus.about.com/od/glossary/g/gloscompiled.htm.

12. Definition of Interpreter. [En línea] [Citado el: 15 de 01 de 2014.] http://cplus.about.com/od/introductiontoprogramming/g/interpreterdefn.htm..

13. What is Visual Studio .NET - Definition from Techopedia. [En línea] [Citado el: 15 de 1 de 2014.] http://www.techopedia.com/definition/15740/visual-studio-net.

14. JetBrains. Basic Concepts. [En línea] [Citado el: 19 de 1 de 2014.] http://www.jetbrains.com/pycharm/webhelp/basic-concepts.html.

15. DroidEdit Pro (code editor) - Aplicaciones Android en Google Play. [En línea] [Citado el: 14 de 4 de 2014.] https://play.google.com/store/apps/details?id=com.aor.droidedit.pro&hl=es_419.

16. Benavente, Pilar. T-ORGANIZA. [En línea] 28 de 1 de 2013. [Citado el: 19 de 2 de 2014.] http://t-organizassdocumentales.blogspot.com/.

17. 920 Text Editor - Text and HTML editor for Android. [En línea] [Citado el: 14 de 4 de 2014.] http://blog.dreamcss.com/android/920-text-editor-text-and-html-editor-for-android/.

18. touchqode - mobile code editor. [En línea] [Citado el: 19 de 2 de 2014.] http://www.touchqode.com/index.htm.

19. touchqode, editor de código para Android. [En línea] 18 de 9 de 2011. [Citado el: 13 de 4 de 2014.] http://webadictos.com/2011/09/18/touchqode-editor-codigo-android/.

20. Al-Chueyr, Tatiana. Desarrollo de aplicaciones móviles para Android con Python. [En línea] [Citado el: 26 de 1 de 2014.] http://revista.python.org.ar/5/es/html/desarrollo-de-aplicaciones-moviles-para-android-con-python.html.

21. Gonzalez, Yasiel Hurtado. Kivy framework para el desarrollo de aplicaciones multi-touch _ humanOS. [En línea] [Citado el: 16 de 2 de 2014.] http://humanos.uci.cu/2014/01/kivy-framework-para-el-desarrollo-de-aplicaciones-multi-touch/.

22. kivy org. *Kivy Documentation.*

23. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. *Fundamentos de sistemas operativos.* España : s.n., 2006.

24. SISTEMA OPERATIVO IOS. [En línea] 07 de 11 de 2012. [Citado el: 20 de 01 de 2014.] http://www.slideshare.net/TenshiDam/sistema-operativo-ios.

25. Windows Phone - ElOtroLado. [En línea] [Citado el: 23 de 01 de 2014.] http://www.elotrolado.net/wiki/Windows_Phone.

26. ANDROID-Definicion. *ANDROID-Definicion.* [En línea] [Citado el: 29 de 1 de 2014.] http://www.gsmspain.com/glosario/?palabra=ANDROID.

27. Salas, Danny. La historia y los comienzos de Android, el sistema operativo de Google. [En línea] 18 de 8 de 2011. [Citado el: 12 de 2 de 2014.] http://www.elandroidelibre.com/2011/08/la-historia-y-los-comienzos-de-android-el-sistema-operativo-de-google.html.

28. Vilchez, Angel. Que es Android Características y Aplicaciones. [En línea] 2 de 4 de 2009 . [Citado el: 9 de 12 de 2013.] http://www.configurarequipos.com/doc1107.html.

29. Vico, Ángel J. Arquitectura de Android « La columna 80. [En línea] 17 de 02 de 2011. [Citado el: 19 de 5 de 2014.] http://columna80.wordpress.com/2011/02/17/arquitectura-de-android/.

30. Ventajas y Desventajas de un Android - Notadiario - Lo interesante _ Notadiario – Lo interesante. [En línea] 14 de 12 de 2011. [Citado el: 1 de 3 de 2014.] http://www.notadiario.com/sci-tech/ventajas-y-desventajas-de-un-android/.

31. Metodología de Desarrollo de Software. [En línea] [Citado el: 13 de 2 de 2014.] http://es.scribd.com/doc/12983329/Metodologia-de-Desarrollo-de-Software.

32. Patricio Letelier, Carmen Penadése: Xtreme Programming (XP). *Métodologías ágiles para el desarrollo de software:.*

33. Riola, J.C. Carvajal. *Metodologías ágiles: Herramientas y modelo de desarrollo para aplicaciones Java.* 2008.

34. Ivar Jacobson, Grady Booch,James Rumbaugh. *EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE.*

35. Solís, Manuel Calero. *Una explicación de la programación extrema (XP),V Encuentro usuarios xBase 2003 MADRID.* 2003.

36. Roberth G. Figueroa, Camilo J. Solís,Armando A. Cabrera. *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.*

37. Qué es SCRUM _ proyectos Ágiles. [En línea] [Citado el: 19 de 2 de 2014.] http://www.proyectosagiles.org/que-es-scrum.

38. James Rumbaugh, Ivar Jacobson, Grady Booch. *El Lenguaje Unificado de Modelado.*

39. Visual Paradigm for UML (ME) - (Paradigma Visual para UML (ME)) (Visual Paradigm for UML (ME)) por Visual Paradigm International Ltd. - reporte y descarga. [En línea] 5 de 3 de 2007. [Citado el: 19 de 2 de 2014.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

40. Caballero, A. *Una herramienta CASE para ADOO; Visual Paradigm.* 2007.

41. Balsamiq Mockups, una herramienta para realizar Wireframes. [En línea] 13 de 2 de 2011. [Citado el: 18 de 3 de 2014.] http://www.glidea.com.ar/blog/balsamiq-mockups-una-herramienta-para-realizar-wireframes.

42. LENGUAJES DE PROGRAMACION _ El Mundo Informático. [En línea] 2007. [Citado el: 1 de 3 de 2014.] http://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion.

43. Definición de lenguaje de programación - Qué es, Significado y Concepto. [En línea] [Citado el: 2 de 3 de 2014.] http://definicion.de/lenguaje-de-programacion.

44. Python premiado como el mejor lenguaje de programación. [En línea] 12 de 12 de 2011. [Citado el: 3 de 3 de 2014.] http://www.gacetatecnologica.com/empresas/novedades/2044-phyton-premiado-como-el-mejor-lenguaje-de-programacion-.html.

45. programming Python. [En línea] [Citado el: 20 de 2 de 2014.] http://ricardowong.tumblr.com/post/2693948431/python.

46. Kv language — Kivy 1.8.1-dev documentation. [En línea] [Citado el: 16 de 3 de 2014.] http://kivy.org/docs/guide/lang.html.

47. davidhalter_jedi · GitHub. [En línea] [Citado el: 23 de 2 de 2014.] https://github.com/davidhalter/jedi.

48. pysmell 0.7.3 Python Package Index. [En línea] [Citado el: 23 de 2 de 2014.] https://pypi.python.org/pypi/pysmell.

49. The Python Standard Library — Python 3.4.0 documentation. [En línea] [Citado el: 23 de 2 de 2014.] http://docs.python.org/3/library/.

50. SearchSOA. What is Eclipse - Definition from WhatIs.com. [En línea] [Citado el: 19 de 1 de 2014.] http://searchsoa.techtarget.com/definition/Eclipse.

51. Introducción a Visual Studio. [En línea] Microsoft. [Citado el: 15 de 1 de 2014.] http://msdn.microsoft.com/es-es/library/fx6bk1f4%28v=vs.90%29.aspx.

52. JetBrains. JetBrains PyCharm Quick Start Guide. [En línea] [Citado el: 17 de 1 de 2014.] http://www.jetbrains.com/pycharm/quickstart/.

53. Patrones de Fabricación Fábricas de Objetos. [En línea] [Citado el: 12 de 3 de 2014.] http://msdn.microsoft.com/es-es/library/bb972258.aspx.

54. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Patrones de diseño. Elementos de software orientado a objetos reutilizable.* España : s.n., 2003.

55. Larman, Craig. *UML Y PATRONES, Introducción al análisis y deseño orientado a objetos.*

56. Maite Rodríguez Corbea, Meylin Ordóñez Pérez. *La Metodología XP Aplicable al Desarrollo del Software.* 2007.

Bibliografía

Alsina, Roberto. *Python no Muerde.*

Ambler, Scott W. 1998. *An Introduction to Process Patterns.* 1998.

Andres Marzal, David Llorens e Isabel Gracia. *Aprender a programar con Python: una experiencia docente.*

Andres Marzal, Isabel Gracia. 2003. *Introducción a la programación.* 2003.

Arambillet, Félix Prieto. 2004. *Patrones de diseño.* Valladolid : s.n., 2004.

David Trowbridge, Dave Mancini, Dave Quick. 2003. *Enterprise Solution Patterns.* 2003.

Drake, Guido van Rossum, Fred L. 200. *Guía de aprendizaje de Python.* 200.

Duque, Raúl González. *Python para todos.* España : s.n.

Francisco Luis Gutiérrez, José Luis Isla, Miguel Gea. *Modelando Patrones de Organización.* España : s.n.

J. M. Burgos, J. Galve, J. García. *Organización del Conocimiento.* Madrid : s.n.

James Rumbaugh, Ivar Jacobson, Grady Booch. 2000. *El lenguaje unificado de modelado. Manual de referencias.* España : s.n., 2000. ISBN:88-7829-037-0.

Kivy org. *Kivy Documentation.*

Larman, Craig. *UML y patrones. Introducción al análisis y diseño de la programación orientada a objeto.*

Mark Pilgrim, Francisco Callejo Giménez, Ricardo Cárdenes Medina. 2005. *Inmersión en Python.* 2005.

Mestras, Pedro A. Gonzalez Calero, Juan Pavon. 2000. *Patrones de diseño orientado a objeto.* 2000.

Patrones del "Gang of Four". Madrid : s.n.

2002. *Patrones y Patrones de diseño.* 2002.

Pressman, Roger S. *Ingenieria de software. Un enfoque practico.*

Roberth G. Figueroa, Camilo J. Solís, Armando A. Cabrera. *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.*

Rossum, Guido van. 2009. *El tutorial de Python.* 2009.

Solis, Ricardo Tercero. 2010. *XP Programación Extrema para Desarrollo de Sistema Basados en Web.* 2010.

Zayas, Dr. Cs. Carlos Alvarez de. 1995. *METODOLOGIA DE LA INVESTIGACION CIENTIFICA.* SANTIAGO DE CUBA : s.n., 1995.

Glosario de términos

MIT: es una licencia que permite reutilizar el *software* así licenciado tanto para ser *software* libre como para ser *software* no libre, permitiendo no liberar los cambios realizados al programa original.

GPS (*Global Positioning System*): es un Sistema Global de Navegación por Satélite que permite fijar a escala mundial la posición de un objeto.

TAGS: es un archivo generado por la biblioteca PySmell donde se guardan un conjunto de etiquetas propias del lenguaje Python, que posibilitan el completamiento de código.

Emulador de Arquitectura: es un *software* que permite ejecutar programas en una plataforma (sea una arquitectura de *hardware* o un sistema operativo) diferente de aquella para la cual fueron creados originalmente.

Sistema de tiempo real: es un sistema que responde a un estímulo externo dentro de un tiempo especificado. Su eficiencia no solo depende de la exactitud de los resultados de cómputo, sino también del momento en que los entrega.

Multitouch(multitáctil): consiste en una pantalla táctil o *touchpad* que reconoce simultáneamente múltiples puntos de contacto, así como el *software* asociado a esta que permite interpretar dichas interacciones simultáneas.

JDT: significa herramientas de desarrollo Java. Es uno de los proyectos que conforman la plataforma Eclipse.

Interprete: es un programa informático capaz de analizar y ejecutar otros programas.

Simulador de arquitectura: es una herramienta que permite reproducir el comportamiento de un programa en una arquitectura determinada.

Volcados de memoria: es un registro no estructurado del contenido de la memoria en un momento concreto, generalmente utilizado para depurar un programa que ha finalizado su ejecución incorrectamente.

OpenGl: es una biblioteca gráfica que permite crear efectos gráficos en cualquier OS.

Interfaz de usuario tangible: es una interfaz de usuario en la que una persona interactúa con la información digital a través de la física del entorno.

Conteo referencial: es una técnica que se utiliza cuando se necesita mantener varias referencias a un mismo sitio, o más concretamente a un mismo objeto sin tener que tener varias copias de dicho objeto en memoria.

***Widget*:** es una aplicación, usualmente presentada en archivos o ficheros pequeños que son ejecutados por un motor de *widgets* o *Widget Engine*. Entre sus objetivos está dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

Anexos

Anexo 1: Diferencias entre las metodologías ágiles y las tradiciones

Metodología Ágil	Metodología Tradicional
Pocos Artefactos. El modelado es prescindible, modelos desechables.	Más Artefactos. El modelado es esencial, mantenimiento de modelos.
Pocos Roles, más genéricos y flexibles.	Más Roles, más específicos.
No existe un contrato tradicional, debe ser bastante flexible.	Existe un contrato prefijado.
Cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Orientada a proyectos pequeños. Corta duración, equipos pequeños (< 10 integrantes) y trabajando en el mismo sitio.	Aplicables a proyectos de cualquier tamaño, pero suelen ser especialmente efectivas/usadas en proyectos grandes y con equipos posiblemente dispersos.
La arquitectura se va definiendo y mejorando a lo largo del proyecto.	Se promueve que la arquitectura se defina tempranamente en el proyecto.
Énfasis en los aspectos humanos: el individuo y el trabajo en equipo.	Énfasis en la definición del proceso: roles, actividades y artefactos.
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Se esperan cambios durante el proyecto.	Se espera que no ocurran cambios de gran impacto durante el proyecto.

Tabla 25. Diferencias entre las metodologías ágiles y las tradiciones.

Anexo 2: Comparación entre SCRUM y XP

	SCRUM	XP
Sistema como algo cambiante	5	5
Colaboración	5	5
Características Metodología(CM)		
-Resultados	5	5
-Simplicidad	5	5
-Adaptabilidad	4	3
-Excelencia técnica	3	4
-Prácticas de colaboración	4	5

Media CM	4.2	4.4
Media Total	4.7	4.8

Tabla 26. Ranking de "agilidad" (Los valores más altos representan una mayor agilidad).

Anexo 3: Tarjetas CRC

Clase: Class.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: def__init__(self, code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options

Tabla 27. Tarjeta CRC Class.

Clase: ClassApp	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Class

Tabla 28. Tarjeta CRC ClassApp.

Clase: Dictionary.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self, code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options

Tabla 29. Tarjeta CRC Dictionary.

Clase: DictionaryApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Dictionary

Tabla 30. Tarjeta CRC DictionaryApp.

Clase: For.
Súper clase: GridLayout
Sub Clase(s):

Responsabilidades:	**Colaboraciones:**
__init__(self,code_input, **kwargs) validate(self, some)	Options

Tabla 31. Tarjeta CRC For.

Clase: ForApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades:	**Colaboraciones:**
build(self)	For

Tabla 32. Tarjeta CRC ForApp.

Clase: If.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades:	**Colaboraciones:**
__init__(self, code_input, **kwargs) validate(self, some)	Options

Tabla 33. Tarjeta CRC If.

Clase: IfApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades:	**Colaboraciones:**
build(self)	If.

Tabla 34. Tarjeta CRC IfApp.

Clase: List.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades:	**Colaboraciones:**
__init__(self,code_input, **kwargs) validate(self, some)	Opitions.

Tabla 35. Tarjeta CRC List.

Clase: ListApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades:	**Colaboraciones:**
build(self)	List.

Tabla 36. Tarjeta CRC ListApp.

Clase: Method.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self, code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options.

Tabla 37. Tarjeta CRC Method.

Clase: MethodApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Method.

Tabla 38. Tarjeta CRC MethodApp.

Clase: OptionsApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Options.

Tabla 39. Tarjeta CRC OptionApp.

Clase: Print.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self,code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options.

Tabla 40. Tarjeta CRC Print.

Clase: PrintApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Print.

Tabla 41. Tarjeta CRC PrintApp.

Clase: Return.

Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self,code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options

Tabla 42. Tarjeta CRC Return.

Clase: ReturnApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Return

Tabla 43. Tarjeta CRC ReturnApp.

Clase: Tuple.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self,code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options

Tabla 44. Tarjeta CRC Tuple.

Clase: TupleApp.	
Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** Tuple

Tabla 45. Tarjeta CRC TupleApp.

Clase: While.	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: __init__(self,code_input, **kwargs) validate(self, some)	**Colaboraciones:** Options

Tabla 46. Tarjeta CRC While.

Clase: WhileApp

Súper clase: App	
Sub Clase(s):	
Responsabilidades: build(self)	**Colaboraciones:** While

Tabla 47. Tarjeta CRC WhileApp.

Clase: KivyCodeTracer.	
Súper clase:	
Sub Clase(s):	
Responsabilidades: __init__(self) exec_code(self, python_code) trace_lines(self, frame, event, arg) trace_calls(self, frame, event, arg)	**Colaboraciones:** CompletionCodeInput

Tabla 48. Tarjeta CRC KivyCodeTracer.

Clase: AppContainer	
Súper clase: GridLayout	
Sub Clase(s):	
Responsabilidades: show(self, content) create(self, contentx)	**Colaboraciones:** CompletionCodeInput

Tabla 49. Tarjeta CRC AppContainer.