Sistema de Medición de Signos Vitales basado en Arduino y telefonía móviles

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

Autor

Jorge Lazaro Amaro Rodriguez

Tutor

Ing. Ignais La Paz Trujillo

22 de junio de 2016
"Año 58 de la Revolución"

"Mi ambición ha sido siempre hacer realizables los sueños."

Bill Gates

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Jorge Lazaro Amaro Rodriguez

_____

Tutor: Ing. Ignais La Paz Trujillo

_____

Dedicatoria y agradecimientos

A mi novia Wendys, a quien le agradezco por todo el apoyo incondicional que me brindó durante toda la carrera, en cada examen y sobre todo en los momentos más difíciles por lo que transité. Por sus atenciones, sus cuidados, su amor, sus ocurrencias. Por darme una ayuda invaluable en el desarrollo de mi trabajo de diploma.

A mi mamá, quien espero que se sienta orgullosa de mis logros porque gracias a ella hoy pude llegar hasta donde estoy y convertirme en la persona que soy. Nunca podré pagarte por todo lo que has hecho por mí, muchas gracias.

A mi papá, por todo el apoyo, las palabras de ánimo, la confianza depositada en mí y creer que podía lograr este sueño.

A mi tío Rene, a quien considero mi segundo padre, mi abuela Ludy, a Tony, a Saray y Richard, gracias a cada uno de ustedes fue posible lograr mi meta.

A otras personas que con el tiempo también se han convertido en parte de mi familia: mis suegros Naty y Papito, Walbe, Ismaray, Felicia, Ileana, Benito, gracias por todas sus enseñanzas en general.

A mis mejores amigos de toda la vida José y Gabriel.

A mis amigos más cercanos de la universidad Keiger, Leduan, Papo y Víctor.

A Javier, Tony y Osvel por su ayuda en el desarrollo de mi tesis. Gracias por su paciencia, su tiempo y su disposición para atenderme cuando los necesitaba.

A mis compañeros de apartamento y del equipo de básquet, con los que compartí muy buenos momentos.

A los profesores Julio, Tony, Yordan, a mi tutor Ignais, gracias por su ayuda.

A la profe Anelis. Gracias por todo su apoyo.

A la profe de inglés Zaida, que estuvo al pendiente de mi recuperación el tiempo que me ausenté de la escuela por problemas de salud.

Al resto de los profesores que me han impartido clases a lo largo de la carrera. Gracias por todas sus enseñanzas.

Resumen

Actualmente en Cuba, no se cuenta con un Sistema de Medición de Signos Vitales en las ambulancias que sea capaz de comunicarse con dispositivos móviles, por lo cual el objetivo principal del presente trabajo es desarrollar un Sistema de Medición de Signos Vitales (SMSV) que cubra esta necesidad. Para dar cumplimiento al objetivo planteado, se hizo una investigación sobre la existencia de antecedentes de este software a nivel mundial. Se explican las principales tecnologías y herramientas usadas para el desarrollo del mismo. Se realiza la gestión de requerimientos y el análisis y diseño de la arquitectura propuesta, además de una serie de pruebas para validar el correcto funcionamiento de la aplicación.

Palabras claves: signos vitales, móviles, SMSV.

Índice

Contenido

Índice de tablas

Índice de ilustraciones

Introducción

En la actualidad, sigue siendo un reto para los especialistas de la medicina hacer diagnósticos asertivos, por ello, es importante en la toma de decisiones contar con la información necesaria del paciente, porque de allí depende un tratamiento adecuado a la sintomatología que presenta la persona. Por tanto, es fundamental la constante actualización de los especialistas sobre el estado del paciente, teniendo en cuenta sus signos vitales, usualmente conocidos como: presión arterial, temperatura, frecuencia cardiaca y respiratoria

En el mundo actualmente existen disímiles tecnologías capaces de realizar esta función, lo que ayuda en gran medida a la labor de los médicos, por ejemplo: el *Welch Allyn 1500*, que es un monitor de pacientes desarrollado por la marca *Propaq® de Welch Allyn* que ha ganado reputación a nivel mundial en cuanto a monitoreo y observación de pacientes.

A nivel mundial existe un crecimiento y desarrollo de la informática y las comunicaciones, acompañado de un gran impacto social. Debido a esto, crece el número de organizaciones e instituciones empresariales que han optado por realizar o actualizar aplicaciones que gestionen sus informaciones con eficiencia y calidad, encontrando en esta gran revolución una vía para dar solución a sus necesidades. Para las grandes empresas del mundo de hoy, contar con la información adecuada para la toma de mejores decisiones en el momento preciso resulta algo fundamental.

En Cuba el desarrollo de la informática es un sector muy importante, ejemplo de ello es el surgimiento de la Universidad de las Ciencias Informática (UCI), por el comandante Fidel Castro, en el 2002. Es la única de su tipo en el país, al tener vinculado el carácter Productivo – Formativo. En la misma se desarrollan distintos proyectos que realizan *software* para la comercialización tanto a nivel nacional como internacional, contribuyendo así al proceso de informatización de la sociedad cubana, aplicándose de forma gradual e integral las nuevas tecnologías de la información y las comunicaciones (TIC). También forma parte de este proceso la informatización del Sistema Nacional de Salud (SNS), lo cual permitirá una mejor gestión, transmisión y control de la información en este sector, ofreciendo un mayor beneficio para la población.

El sistema de salud cubano es universal, gratuito y accesible a todos los ciudadanos, lo cual se manifiesta en su red de unidades asistenciales en el territorio nacional y en su sistema de atención médica a través de los diferentes programas priorizados. Por estas razones es de vital importancia realizar acciones para contribuir a su mejora.

En disímiles ocasiones surge la necesidad de brindar atención priorizada e inmediata a quien lo necesite, es decir, prestarle al paciente una atención de urgencia. En varias circunstancias esto puede llegar a ser un poco lento, debido a la situación actual del sector de salud en las instituciones sanitarias cubanas, ya que:

- ✓ En caso de emergencia el médico no tiene la forma de conocer el estado fisiológico del paciente antes de llegar al hospital, lo que provoca una pérdida de tiempo muy valiosa a la hora de darle el tratamiento al enfermo.
- ✓ Los Sistemas de Medición de Signos Vitales (SMSV) actualmente en funcionamiento, no son accesibles desde cualquier ubicación geográfica, por lo que el médico no tiene acceso a la información en todo momento.
- ✓ No se cuenta con un SMSV capaz de comunicarse con dispositivos móviles.

Teniendo en cuenta las dificultades antes expuestas, se centrarán los esfuerzos en darle solución al siguiente **problema**: ¿Cómo mejorar los sistemas de atención al paciente en la salud cubana?

A partir del problema anterior se puede inferir que el **objeto de estudio** lo constituyen los SMSV, enmarcado en el **campo de acción** SMSV basado en microcontroladores.

Para dar solución al problema planteado se hace preciso desarrollar un sistema de medición de signos vitales que permita mejorar la atención al paciente lo cual representa el **objetivo general** de la investigación.

Hipótesis:

Con el desarrollo de un Sistema de Medición de Signos Vitales basado en Arduino y telefonía móvil, será posible mejorar la atención al paciente en Cuba.

Durante el desarrollo de este trabajo se trazan las siguientes **tareas** para darle cumplimiento a los objetivos:

- ✓ Conformación de la fundamentación teórica partiendo del estado del arte de los Sistemas de Medición de Signos Vitales.
- ✓ Definición de los requerimientos funcionales y no funcionales.
- ✓ Selección del microcontrolador y los sensores para la solución.
- ✓ Desarrollo del *firmware*.
- ✓ Diseño y ejecución de pruebas.

Se emplearon los siguientes métodos:

Métodos Teóricos

Analítico-sintético: se utilizó en el proceso de análisis de la bibliografía y realizando una síntesis de la misma.

Histórico-lógico: como resultado de la utilización de este método, se lograron conocer los antecedentes y tendencias actuales de los SMSV y su evolución, tomando dichas observaciones como guía para el desarrollo de la presente investigación.

Métodos Empíricos

Observación: se utilizó en la investigación para determinar el problema a resolver, las necesidades del cliente y comprender cómo funciona actualmente la atención a los pacientes.

El contenido de la investigación está estructurado en tres capítulos:

Capítulo 1 Fundamentación teórica: se presentan los elementos teóricos que sirven de base al problema planteado. Se realiza un estudio del estado del arte de los sistemas que apoyan la medición de signos vitales y además se hace referencia a las diferentes tecnologías que se utilizaron para hacer el *software*.

Capítulo 2 Análisis de la solución propuesta: en este capítulo se capturan los requerimientos necesarios para el desarrollo del *software* y se describe de una forma más detallada el sistema presentando la arquitectura base a partir de las tecnologías seleccionadas para su construcción.

Capítulo 3 Implementación y pruebas: se tratan los aspectos relacionados con la implementación de la solución propuesta. Además, se valida la solución propuesta mediante las pruebas.

Capítulo 1: Fundamentación Teórica

En el presente capítulo se exponen los principales conceptos de la investigación para lograr una mejor comprensión del tema a tratar. Se realiza un estudio del estado del arte de los sistemas que apoyan la medición de signos vitales, los cuales están formados por cuatro elementos fundamentales, que se definen y caracterizan a continuación con el objetivo de profundizar en los términos empleados durante el desarrollo del trabajo.

1.1 Estado de arte

Actualmente se cuenta con muchos métodos para obtener los signos vitales sin necesidad de acudir al médico. Por ejemplo: en la rama del deporte, fuera de los centros de servicio sanitarios, hay dispositivos portables similares a los usados profesionalmente que permiten a los deportistas evaluar sus signos vitales. Algunos de ellos son los monitores del ritmo cardíaco, los cuales se emplean para calcular y controlar su pulso durante un determinado ejercicio. Son denominados pulsómetros y muestran adicionalmente las calorías totales que el deportista quema durante un tiempo específico.

En el campo tecnológico existen compañías médicas que desarrollan sistemas dirigidos fundamentalmente a centros de salud, hospitales y clínicas, que permiten una rápida acción una vez que ocurre un acontecimiento de riesgo con pacientes. Estos equipos están disponibles en el mercado mundial y brindan muchas facilidades a la medicina. Entre ellos se encuentran:

Welch Allyn

Desarrollado por la marca *Propaq*® en Nueva York Estados Unidos y se centra en los departamentos de urgencias, y pisos de hospitalización. Además cuenta con una pantalla que mantiene una profundidad compacta y proporciona una buena visibilidad de la información del paciente. Brinda la posibilidad de agregar otros parámetros como la hemoglobina. (1)

Monitor de Paciente C50

Algunas características: (2)

- ✓ Diseño a prueba de agua tipo IPX1, posibilitando buen desempeño en condiciones diversas.

- ✓ Puerto especial de presión no invasiva con tapa protectora.
- ✓ Baterías de Ion-Litio de 4000mAh, más de 4 horas de autonomía para proteger los datos frente a un corte de energía.
- ✓ Con características tales como análisis de arritmias.
- ✓ Cálculo de dosis de medicamento.

DOCTUS-VI

Desarrollado por la colaboración de informáticos, automáticos, cibernéticos, matemáticos, físicos, diseñadores de equipos del Instituto Central de Investigación Digital (ICID), del Ministerio de la Informática y las Comunicaciones de Cuba. (3)

Algunas características:

- ✓ Pantalla de 10,4 o 15 pulgadas.
- ✓ 72 horas de tendencia almacenadas.
- ✓ Detección, clasificación y almacenamiento de hasta 32 eventos de arritmia.
- ✓ Una hora de trabajo con batería para proteger los datos frente a un corte de energía.

Todo ello lo hace posible la integración de muchos sensores trabajando en conjunto para lograr el funcionamiento correcto de estos equipos. Los cuales pueden ser adquiridos rápidamente en el mercado.

1.2 Signos Vitales (SV)

Los SV constituyen una herramienta valiosa como indicadores del estado funcional de una persona. Es necesario acentuar que tener los valores de estos, no tiene ningún significado si no se interpretan adecuada y oportunamente, puesto que ayudan al médico a decidir conductas de manejo.

La Organización Mundial de Salud (OMS) plantea que los SV son los parámetros o manifestaciones objetivas que pueden ser observadas, medidas y monitoreadas para evaluar el nivel de funcionamiento físico de un individuo. Sus rangos normales varían según la edad, la condición física, el sexo, el peso y la tolerancia al ejercicio ya sea físico o mental. (4)

Los rangos normales para un adulto sano promedio mientras está en reposo son:

- ✓ Presión arterial: 90/60 mm/Hg hasta 120/80 mm/Hg.
- ✓ Respiración: 12 a 18 respiraciones por minuto.

- ✓ Pulso: 60 a 100 latidos por minuto.
- ✓ Temperatura: 36.5-37.2° C (97.8-99.1° F)/promedio de 37º C (98.6° F)

Los SV son indicadores que reflejan el estado fisiológico de los órganos vitales (cerebro, corazón, pulmones).Expresan de manera inmediata los cambios funcionales que suceden en el organismo, que de otra manera no podrían ser cualificados ni cuantificados. Los cuatro principales son:

- ✓ Frecuencia Cardiaca que se mide por el pulso en latidos/minutos.
- ✓ Frecuencia respiratoria.
- ✓ Tención (presión) arterial.
- ✓ Temperatura.

Para realizar una correcta medición de los signos vitales se debe tener en cuenta los siguientes aspectos: (5)

- ✓ Reconocer la relación que existe entre los signos vitales, la actividad fisiológica y los cambios fisiopatológicos.
- ✓ Conocer la naturaleza periódica de actividades fisiológicas como base para evaluar la medición de signos vitales.
- ✓ Utilizar la información obtenida por la medición de los signos vitales como factor determinante para valorar la evolución del cliente, la respuesta al tratamiento y las intervenciones de enfermería.
- ✓ Reconocer y evaluar la respuesta individual del enfermo a los factores ambientales, internos y externos, según se manifiestan por la medición de los signos vitales.
- ✓ Vigilar los signos vitales con mayor frecuencia de la ordenada si el estado del paciente lo requiere.
- ✓ Comunicar los datos de los signos vitales a los médicos con la terminología correcta y registros adecuados para mejor tratamiento.

1.2.1 Ritmo Cardiaco o Frecuencia Cardiaca (RC-FC)

MedicineNet.com plantea que “la frecuencia cardíaca es el número de pulsaciones por unidad de tiempo, por lo general por minuto. Se basa en el número de contracciones de los ventrículos (las cavidades inferiores del corazón). Estas pueden ser demasiado rápidas (taquicardia) o demasiado lentas (bradicardia). El pulso es una protuberancia de una arteria de las oleadas de sangre, que se toma a menudo en la muñeca para estimar el ritmo cardíaco.” (6)

El Ritmo Cardíaco es, como lo dice su nombre, el ritmo o la regularidad con que ocurren los latidos del corazón, normalmente son dos ruidos y deben ser rítmicos y regulares, ocurriendo de 60 a 100 veces en un minuto, esto último es lo que se denomina frecuencia cardíaca. El ritmo del corazón viene dado por dos fases que se alternan, una fase corresponde al llenado del corazón y se denomina diástole, mientras que la otra corresponde a la fase de expulsión de la sangre llamada sístole. (7)

Instrumentos más comunes para la toma del ritmo cardiaco

Existen diferentes tipos de instrumentos para medir el pulso, a continuación se presentan y explican los más comunes.

Estetoscopio

Este instrumento es usado principalmente por personal médico para oír los pulsos cardíacos en la toma de la presión. Su uso es totalmente analógico y el registro de pulsos por minutos se hace por medio del conteo de números escuchando el corazón.

Ilustración 1 Estetoscopio.

Pulsómetros de muñeca

Principalmente usado por deportistas para mantener un registro de su ritmo cardiaco durante sus secciones de ejercicio, este elemento mide constantemente el pulso de una persona en minuto.

Ilustración 2 Pulsómetro de muñeca.

1.2.2 Frecuencia Respiratoria (FR)

Es la medición del proceso mediante el cual se toma oxígeno del aire ambiente y se expulsa el CO2 (Anhídrido carbónico) del organismo. Este proceso se realiza a través de ciclos respiratorios, comprende una fase de inspiración y otra de espiración.

La frecuencia respiratoria es la cantidad de respiraciones que una persona hace por minuto, se mide por lo general cuando una persona está en reposo y consiste simplemente en contar la cantidad de respiraciones durante un minuto cada vez que se eleva el pecho. La frecuencia respiratoria puede aumentar con la fiebre, las enfermedades y otras afecciones médicas. Cuando se miden las respiraciones, es importante tener en cuenta también si la persona tiene dificultades para respirar.

Los valores normales de un adulto en estado de reposo son: FR: 15 a 20 respiraciones por minuto.

Cuando la frecuencia es superior de 25 respiraciones por minuto o inferior de 12 (en reposo) se podría considerar anormal, sin embargo en este último, el entrenamiento físico y mental puede hacer y lograr disminución de la frecuencia sin ser patológico. (8)

1.2.3 Tensión (Presión) Arterial (TA-PA)

La presión arterial es la fuerza que ejerce la sangre contra las paredes de las arterias y puede ser medida con un tensiómetro y un estetoscopio por una enfermera u otro proveedor de atención médica. Cada vez que el corazón late, bombea sangre hacia las arterias, lo que produce una presión sanguínea más alta cuando el corazón se contrae. No puede tomarse su propia presión arterial a menos que utilice un tensiómetro electrónico. Los tensiómetros electrónicos también pueden medir el ritmo cardíaco o el pulso. (9)

Cuando se mide la presión arterial se registran dos números. El número más elevado, la presión sistólica, es la presión dentro de la arteria cuando el corazón se contrae y bombea sangre a través del cuerpo; mientras que el número más bajo, la presión diastólica, es la presión dentro de la arteria cuando el corazón está en reposo y llenándose con sangre. Ambas presiones se registran en "mm de Hg" (milímetros de mercurio). Este registro representa cuán alto la presión sanguínea eleva la columna de mercurio en un tensiómetro antiguo (como el manómetro o el esfigmomanómetro de mercurio). (10)

Actualmente existe una gama de equipos que realizan las lecturas de forma digital por lo que el valor importante es la cifra sin la medida de mercurio, evitando así tal y como recomienda la OMS el uso contaminante del mercurio en equipos médicos (tensiómetros, termómetros)

La medición y el conocimiento de la presión arterial, permite saber si está o no normal. Si esta alta, es decir, sobre los valores normales, entonces hablamos de hipertensión. Esto sucede cuando las arterias ofrecen una mayor resistencia al flujo de la sangre, con lo que al corazón le resulta más difícil hacerla circular. Controlarse la hipertensión evita el riesgo de cardiopatía coronaria (infarto) y de ictus (accidentes cerebrovasculares)

Presión sanguínea alta: Hipertensión arterial PA: 140 /90

- ✓ Presión sistólica: igual o mayor de 140
- ✓ Presión diastólica: igual o mayor de 90

Instrumentos más comunes para la toma de la temperatura corporal

Existen diferentes tipos de instrumentos para medir la presión arterial, a continuación se presentan y explican los más comunes.

Baumanómetro de brazo

Posee un indicador de aguja donde se observa la presión ejercida sobre el brazo.

Ilustración 3 Baumanómetro de brazo.

Baumanómetro de muñeca

Este tipo de baumanómetro se coloca en la muñeca preferiblemente del brazo izquierdo, consta principalmente de una pantalla donde se muestran los resultados de la medición y un pequeño brazalete que se infla al tomar la medición.

Ilustración 4 Baumanómetro de muñeca.

1.2.4 Temperatura (T)

"La temperatura se define como "una magnitud física que puede ser determinada por un termómetro y que caracteriza de manera objetiva el grado de temperatura corporal." (11)

Es el grado de calor sensible o frío, expresada en términos de una escala específica. Se mide mediante un termómetro clínico y representa un equilibrio entre el calor producido por el cuerpo y el calor que pierde. Aunque la producción y pérdida del calor varían en función de las circunstancias, el cuerpo las regula manteniendo una temperatura notablemente constante por lo que se puede llamar fiebre a un aumento anormal de la temperatura del cuerpo. (12)

Valores normales de la temperatura: (5)

- ✓ Recién Nacidos: 36.6ºc a 37.8ºc
- ✓ Lactantes: 36.5º c _ 37º c

- ✓ Preescolar y escolar: 36º _ 37º c
- ✓ Adolescentes: 36º - 37º c
- ✓ Edad adulta: 36.5º c
- ✓ Vejez: 36º c

Cuando ocurren alteraciones en los valores anteriores, se produce pirexia, hipertermia o fiebre: la temperatura fuera de los valores normales.

- ✓ Hiperexia o hipertermia: 41º c
- ✓ Febril: tiene fiebre 38º c
- ✓ Afebril : no tiene fiebre (37º c)
- ✓ Hipotermia : 35.5º c
- ✓ Febrícula: 37.5º (subfebril)

Instrumentos más comunes para la toma de la temperatura corporal

Existen diferentes tipos de instrumentos para medir la temperatura corporal, a continuación se presentan y explican los más comunes.

Termómetro de vidrio de mercurio

Es un cilindro de vidrio hueco con un depósito de mercurio en el fondo y en el extremo superior cerrado. Tiene una escala graduada que va desde los 35 °C hasta los 42 °C. En un termómetro se distinguen dos partes: el tallo, que comprende la zona de la escala graduada y el bulbo, que es donde se aloja el mercurio.

Al aumentar la temperatura el mercurio se dilata y asciende por el capilar, la escala graduada permite leer rápidamente el valor de la temperatura. Este termómetro es el más usado, aunque no el más preciso.

Ilustración 5 Termómetro de mercurio.

Termómetro digital

Es una alternativa segura frente a los termómetros de vidrio con mercurio, ya que no hay riesgo de vidrio roto o intoxicación por mercurio. La lectura es muy sencilla y rápida.

Se utiliza de la misma manera que el termómetro de vidrio, lo que la lectura se realiza a través de una pantalla.

Ilustración 6 Termómetro digital.

Termómetro de oído

Son termómetros digitales y la lectura se realiza a través de una pantalla a los pocos segundos de situado el instrumento en el canal auditivo.

Ilustración 7 Termómetro digital de oído.

1.3 Protocolo GPRS

En la actualidad se utilizan mecanismos que permiten la comunicación entre teléfonos celulares. Ejemplo de los mismos son los protocolos GSM (Sistema Global para comunicaciones Móviles) y GPRS (Paquete General de Radio Servicio), siendo este último el más utilizado actualmente debido a las características que se ofrecen a continuación: (13)

- ✓ Es un servicio de datos móvil orientado a paquetes.
- ✓ Está disponible para los usuarios del GSM
- ✓ Permite velocidades de transferencia de 56 a 114 kbps.
- ✓ La transferencia de sus datos se cobra por *megabyte* de capacidad, mientras que la comunicación de datos a través de conmutación de circuitos tradicionales se factura por minuto de tiempo de conexión, independiente de si el usuario utiliza la capacidad o está en un estado de inactividad.
- ✓ Da mejor rendimiento a la conmutación de paquetes de servicios.

Diferencias con GSM: La tecnología *GPRS* mejora y actualiza a *GSM* con los servicios siguientes:

- ✓ Servicio de mensajes multimedia (*MMS*).
- ✓ Mensajería instantánea.
- ✓ Servicio de mensajes cortos (*SMS*).

1.4 Herramientas y tecnologías de desarrollo

Las herramientas y tecnologías escogidas para el desarrollo de un proyecto deben ser seleccionadas cuidadosamente, ya que pueden suponer el fracaso de este o pueden aumentar su complejidad, para lo cual se debe conocer cuáles son las distintas alternativas y las necesidades del mismo. A continuación se realiza una descripción de las seleccionadas para llevar a cabo la presente investigación.

1.4.1 Arduino: plataforma de prototipos electrónica

Es una plataforma de prototipos electrónica de código abierto basada en *hardware* y *software* flexibles y fáciles de usar. Está pensado para artistas, diseñadores, como *hobby* y para cualquiera interesado en crear objetos o entornos interactivos. Arduino a diferencia del ordenador que usas normalmente, no tiene pantalla ni teclado, se necesita un programa externo ejecutado en otro ordenador para poder escribir programas para la placa Arduino. Este *software* es lo que llamamos Arduino *IDE* (Entorno de Desarrollo Integrado). Se escribe el programa en el *IDE*, se carga en el Arduino y se ejecuta en la placa.

Las placas se pueden ensamblar a mano o encargarlas pre ensambladas; el *software* se puede descargar gratuitamente. Los diseños de referencia del *hardware* están disponibles bajo licencia de código abierto, por lo que se puede adaptar a las necesidades de los usuarios.

Existen diferentes tipos de placas Arduino: Arduino UNO, Arduino Zero, Arduino Yun, Arduino Leonardo, Arduino Pro, Arduino Esplora, Arduino BT, entre otras, seleccionándose para el desarrollo del sistema la Arduino Micro, debido a las características que posee y que se muestran a continuación.

Arduino Micro

Diseñado por Adafruit y pensado para una autonomía elevada y con un reducido tamaño. Su precio es bajo con respecto a otros modelos. Sin embargo cuenta con características similares a otros diseños, como un microcontrolador ATmega32u4 a 16Mhz, 20 pines digitales, 7 de ellos Modulación por ancho de pulso (PWM) y 12 analógicos. En muchos aspectos es similar a Arduino Leonardo, pero con capacidad de comunicación USB *built-in*, eliminando la necesidad de un segundo procesador. (14)

Ilustración 8 Placa Arduino Micro.

1.4.2 Sensores

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, movimiento, etcétera. El mundo real no es digital y, a veces, no puedes traducir los cambios del el entorno en lecturas digitales. Por ejemplo, la temperatura no cambia sólo de frío a caliente, cambia en un rango de valores distintos y, normalmente, estos cambios ocurren muy lentamente. Éste es el motivo por el cual, se utilizaron sensores analógicos para leer los parámetros del entorno tal como, la temperatura. Esta información resultante es almacenada como datos digitales secuenciales. Dado que Arduino no puede manejar la información como los humanos, necesitamos traducir los datos analógicos para que Arduino pueda entenderlos. Los sensores analógicos pueden transformar los datos del entorno en un valor de voltaje comprendido entre 0V y 5V. Estos valores son distintos de los altos y bajos (*High* o *Low*) que utilizan los sensores digitales. Para los pines digitales, *High* y *Low* significan 5V y 0V respectivamente, y nada más. Sin embargo los pines analógicos pueden diferenciar cualquier valor intermedio. Dentro de los sensores investigados para el desarrollo del sistema fueron seleccionados de acuerdo a estabilidad, precisión, seguridad, rapidez y costo los siguientes: (15)

Ritmo cardiaco

El sensor de oxígeno para adultos Durasensor de Nellcor modelo DS-100A está indicado para la vigilancia continua no invasiva de la saturación de oxígeno arterial y la frecuencia del pulso en pacientes de más de 40 kg de peso. Cuenta con un conector DB9 que facilita el procesamiento de datos, además de ser uno de los más conocidos en estos medios. (16)

Características técnicas

Tipo	***Valores***
Intervalos de medición	
Intervalo de saturación de SP02	De 1% a 100%
Intervalo de frecuencia del pulso	De 20 a 250 latidos por minutos (lpm)
Intervalo de perfusión	De 0,03% a 20%
Exactitud de las medidas	
Exactitud de frecuencia de pulso	De 20 a 250 latidos por minutos (lpm) +/- 3 dígitos
SP02 exactitud de saturación	De 70% al 100% latidos por minutos (lpm) de +/-2 a +/-3 dígitos
Intervalo de funcionamiento y disipación	
Longitud de onda de luz roja	Aproximadamente 660 nm
Longitud de onda de luz infrarroja	Aproximadamente 900 nm
Potencia de salida óptica	Menos d 15 mW
Disipación de alimentación	52,5 mW

Tabla 1 Características del sensor

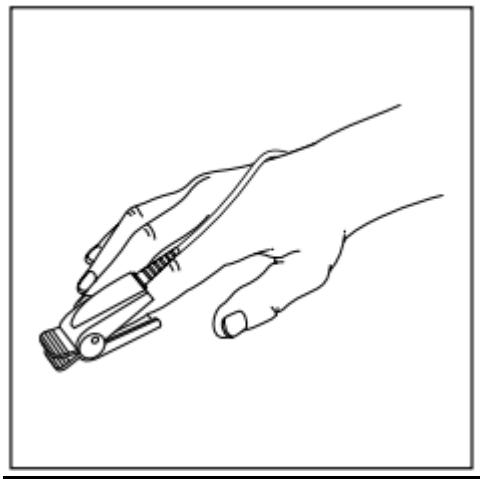

Ilustración 9 Sensor Nellcor DS-100A.

Temperatura

Termistor: Es una resistencia térmicamente sensible, existen dos tipos donde según la variación de la resistencia/coeficiente de temperatura pueden ser negativo (NTC) o positivos (PTC).

Son fabricados a partir de los óxidos de metales de transición (manganeso, cobalto, cobre y níquel) los termistores NTC son semiconductores dependientes de la temperatura. Operan en un rango de -200º C a + 1000° C. Un *termistor* NTC debe elegirse cuando es necesario un cambio continuo de la resistencia en una amplia gama de temperaturas. Ofrecen estabilidad mecánica, térmica y eléctrica, junto con un alto grado de sensibilidad. (17)

La excelente combinación de precio y el rendimiento ha dado lugar a una amplia utilización de los termistores NTC en aplicaciones tales como medición y control de temperatura, compensación y medición del flujo de fluidos.

Ilustración 10 Sensor Termistor.

Presión arterial

MPX5050DP: es un transductor piezoresistivo de la familia MPXx5050xx diseñado para utilizarlo en varias aplicaciones, particularmente empleando un microcontrolador o un microprocesador como entrada en un convertidor analógico-digital (A/D).

Este combina principalmente avanzadas técnicas de micro máquinas y procesos bipolares, bajo un nivel de señal analógica de salida que es proporcional a la aplicación de la presión, además de cubrir el rango de presión que se requiere.

Características técnicas:

- ✓ Presión de 0 a 50 kPa.
- ✓ Voltaje de salida de 0.2 a 4.7 V.
- ✓ Error máximo de 2.5% entre los 0º C a los 85º C.
- ✓ Temperatura de trabajo de 40º C a los 125º C.

Ilustración 11 Sensor MPX5050DP.

Configuración de sensores

Después de tener el código del sensor listo y cargado en el *IDE* es necesario hacer coincidir el pin al cual está conectado con el escrito en el código. Ejemplo:

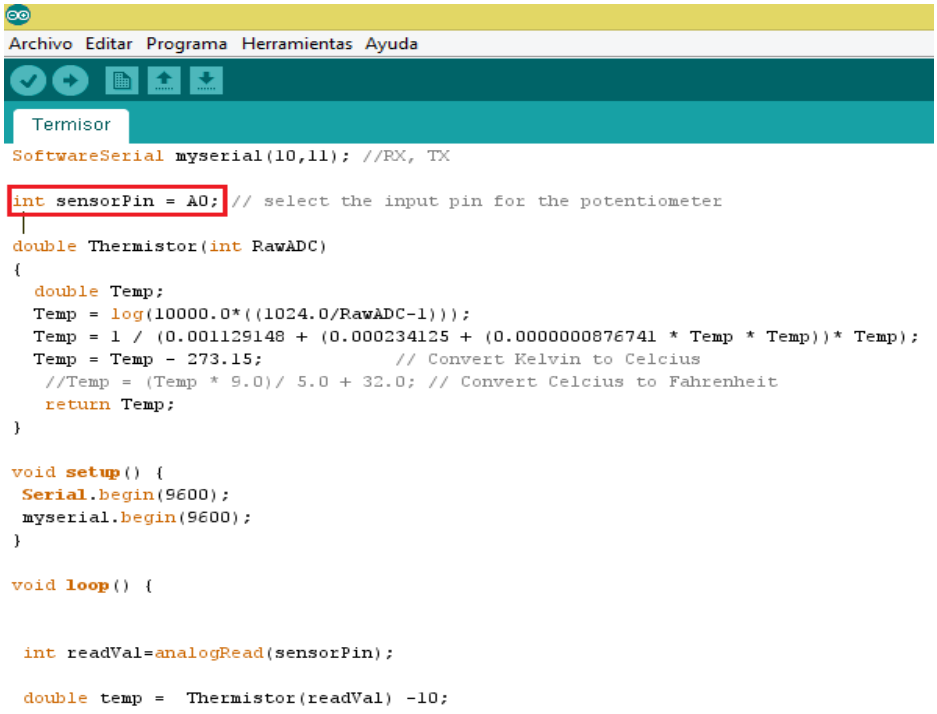

Ilustración 12 Pin utilizado en el código.

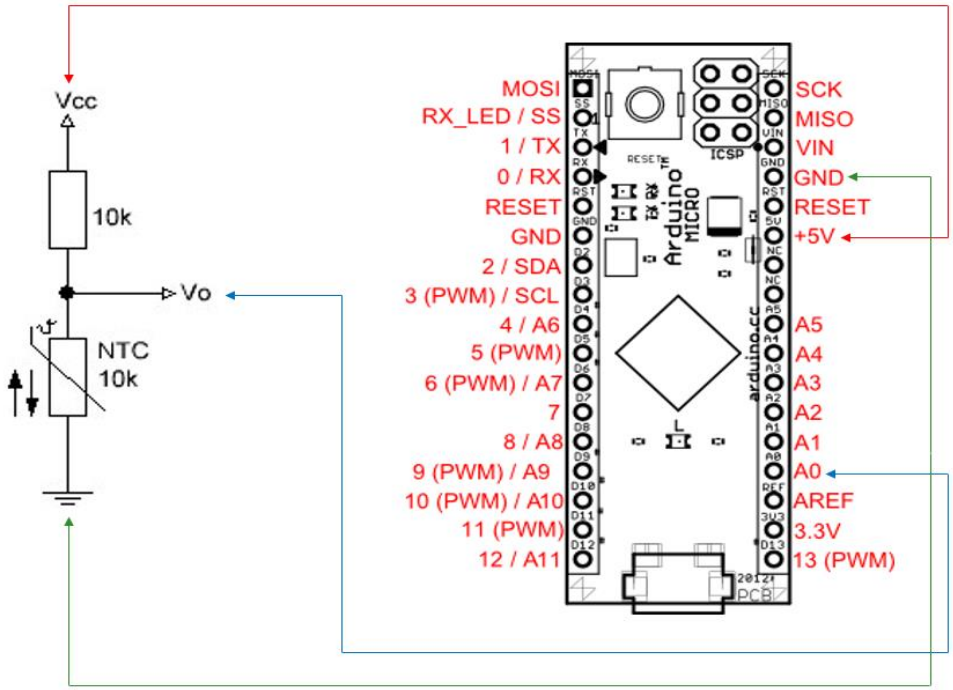

Ilustración 13 Conexión física.

Esto es importante para que exista una correspondencia física entre el código y la conexión en el microcontrolador, es decir, conectar las salidas analógicas del sensor con las entradas analógicas del Arduino. Además se debe realizar la conexión física de la alimentación de voltaje del sensor y el pin tierra, ya que de no realizarse, no será posible obtener los valores del mismo.

1.4.3 Módulo Bluetooth

El módulo HC-06 dispone de cuatro pines y está diseñado solo para actuar como esclavo, es decir que los dispositivos se conectan a él.

Características Técnicas: (18)

- ✓ Especificación *bluetooth* v2.0 + EDR (*Enhanced Data Rate*).
- ✓ Modo esclavo (Solo puede operar en este modo).
- ✓ Puede configurarse mediante comandos AT (Deben escribirse en mayúscula).
- ✓ Chip de radio: CSR BC417143.
- ✓ Frecuencia: 2.4 GHz, banda ISM.
- ✓ Perfiles: Puerto serial *Bluetooth*

Aplicaciones:

- ✓ Comunicación inalámbrica entre microcontroladores.
- ✓ Comunicación inalámbrica entre computadoras y microcontroladores.
- ✓ Comunicación inalámbrica entre teléfonos móviles o tabletas y microcontroladores.

1.4.4 Lenguaje de programación

Java

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de *Sun Microsystems* (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma *Java* de *Sun Microsystems*. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de *Java* son generalmente compiladas a *bytecode* (clase *Java*) que puede ejecutarse en cualquier máquina virtual *Java* (*JVM*) sin importar la arquitectura de la computadora subyacente. (19) (20)

Se creó con cinco objetivos principales:

- ✓ Usar el paradigma de la programación orientada a objetos.
- ✓ Permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- ✓ Incluir por defecto soporte para trabajo en red.
- ✓ Diseñarse para ejecutar código en sistemas remotos de forma segura.
- ✓ Fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Su diseño, robustez, respaldo de la industria y fácil portabilidad, han hecho de *Java* uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. Algunas de sus aplicaciones se muestran a continuación:

- ✓ Dispositivos móviles y sistemas embebidos
- ✓ Navegador *web*
- ✓ Sistemas de servidor
- ✓ Aplicaciones de escritorio

Arduino

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel *Processing* que es similar a C++. Este está basado en C y soporta todas las funciones del estándar C y algunas de C++.

1.4.5 Entorno de desarrollo Integrado (IDE)

Un *IDE* es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios.

Arduino IDE

Programa la tarjeta Arduino desde un ambiente gráfico que permite para fines didácticos avanzar con mayor velocidad. Sus características fundamentales son las siguientes:

- ✓ Tiene un lenguaje simple, basado en C/C++.
- ✓ Permite desde un primer contacto estar programando directamente el *hardware*.
- ✓ Es un proyecto de código abierto, por lo que debido a su precio podemos probar y experimentar sobre la misma tarjeta.

Android Studio IDE

Es un entorno de desarrollo integrado para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el *IDE* oficial para el desarrollo de aplicaciones para Android. Entre sus principales características se encuentra: (21)

- ✓ Renderización en tiempo real.
- ✓ Plantillas para crear diseños comunes de Android y otros componentes.
- ✓ Vista previa en diferentes dispositivos y resoluciones.
- ✓ Editor de diseño que muestra una vista previa de los cambios realizados directamente en el archivo *XML*.

Requerimientos del sistema:

Windows	***Mac OS***	***Linux***
Microsoft Windows 8/7/Vista/2003 (32 o 64 *bit*)	*Mac OS* X 10.8.5 o superior, hasta la 10.9 (Mavericks)	*GNOME* o entorno de escritorio *KDE*
Mínimo de 2 *GB* de *RAM*, recomendado 4 *GB* de *RAM*		
400 *MB* de espacio en disco		
Necesita de al menos 1 GB para Android *SDK*, emulador de imágenes del sistema, y cachés		

Resolución mínima de pantalla de 1280 x 800
Java Development Kit (*JDK*) 7 o superior

Tabla 2 Requerimientos del sistema.

Java SE JDK

Es el conjunto de recursos *Java* o plataforma, que conforman la denominada *Java Standard Edition*, fundamental para el desarrollo de aplicaciones *Java*. Este conjunto de recursos no es propio de la plataforma Android, pero debe ser instalada en el computador para poder desarrollar aplicaciones.

Android SDK

Es el conjunto de recursos fundamentales que dispone la plataforma Android para el desarrollo de aplicaciones *Java*. Posteriormente se debe configurar el *IDE* Android Studio para que localice el *SDK* de Android y disponga de sus recursos en el entorno de desarrollo. Entre los recursos de que dispone el *SDK* de Android se encuentra un emulador de dispositivos, para probar los desarrollos sin necesidad de realizar instalaciones sobre un dispositivo físico. (22)

Crear un Android Virtual Device

Es la configuración de dispositivo que será utilizada por el emulador de Android. Mediante *AVD* es posible definir los característicos *hardware* y *software* del dispositivo que se va a emular y para el cuál se quiere ejecutar la aplicación en desarrollo. (22)

Emulador Android

El *SDK* de Android trae integrado el emulador, el cual simula un dispositivo Android completo, dicho sistema trae integrado, aplicaciones básicas, un teclado, botones de volumen y encendido. De este modo basta con configurar un nuevo dispositivo virtual para poder hacer uso de sus capacidades, permitiendo de manera fácil y rápida probar y depurar aplicaciones desarrolladas sin tener que usar un dispositivo físico, ya que puede configurarse con diferentes características de *hardware*, como por ejemplo, el tamaño de la pantalla, y otras como lo es la versión de Android que ejecuta.

Ilustración 14 Emulador de Android.

1.4.6 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje de modelado visual que es usado para especificar, visualizar, construir y documentar documentos y artefactos de un sistema de *software*. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas de ciclo de vidas, dominios de aplicación y medios. Incluye conceptos semánticos, notación y principios generales. Es usado para describir grandes sistemas con la calidad requerida y que estos puedan ser entendidos por los usuarios. Además no es solo de gran utilidad para el usuario sino también para los desarrolladores ya que permite tener una idea más clara de lo que se quiere realizar y el cómo se va a realizar. (23)

1.4.7 Herramienta CASE

Las herramientas *CASE* (acrónimo de *Computer Aided Software Engineering*, en español: Ingeniería de Software Asistida por Computadora), son diversas aplicaciones informáticas destinadas a aumentar la efectividad en el desarrollo de *software* y sistemas en términos de productividad y calidad, contribuyendo a una reducción de costos de producción. Incluyen programas para el diseño de sistemas, ingeniería inversa, generadores de código y otros. (24)

Se realizó una investigación de diferentes herramientas *CASE*, entre ellas: *Microsoft Project* (Privada), *Rational Rose* (Privada), 2.3 *JDeveloper* (Gratuita desde 2005), *Visual Paradigm* (Gratuita), entre otras. En la selección de la herramienta se hizo énfasis en las libre y de código abierto por el ahorro económico que representan. Finalmente se

decidió utilizar *Visual Paradigm* teniendo en cuenta las características que se detallan a continuación.

Visual Paradigm 5.0

Herramienta *CASE* que acelera el desarrollo de aplicaciones, sirviendo de intermediario visual entre arquitectos, analistas y diseñadores de *software*, mediante un ambiente de modelado superior que posibilita un trabajo más fácil y dinámico. Permite modelar con *UML* 2.1, sincronizar modelos y código, realizar ingeniería inversa, generar código automáticamente (con el estilo de codificación deseado), así como diseñar bases de datos y generar sus esquemas (con el *DDL* del Sistema Gestor de Base de Datos seleccionado). (25)

Soporta varios lenguajes de programación como *Java*, *C++*, *C#*, *VB.NET*, *PHP*, *Delphi*, *Python*, varios Sistemas Gestores de Bases de Datos como *Oracle*, *Microsoft SQL Server*, *MySQL*, *PostgreSQL* y *SQLite*. Además, es integrable con varios entornos de desarrollo como *Eclipse*, *NetBeans* y *Microsoft Visual Studio*, permitiendo aumentar la velocidad en el análisis, captura, plan, desarrollo, comprobación y despliegue de los requisitos. Es multiplataforma y cuenta con una versión libre para la comunidad (*Comunity Edition*). (25)

Genera la documentación del sistema en los formatos *PDF*, *HTML* y el formato de documentos de *Microsoft Word* y permite importar proyectos de otras herramientas de modelado como *Rational Rose*, *Erwin y Microsoft Visio*. Soporta la revisión ortográfica, brindando sugerencias para los idiomas: inglés, español, francés, alemán y portugués. (25)

1.5 Metodología de desarrollo de software

Las metodologías de desarrollo de *software* surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de *software*. Su objetivo es guiar a un equipo de proyecto durante la creación de uno nuevo, pero los requisitos a tener en cuenta para ello son tan variados que han dado lugar a diferentes enfoques o interpretaciones sobre el proceso de desarrollo de *software*. Las metodologías se clasifican en dos grandes grupos:

Metodologías tradicionales: orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.

Metodologías ágiles: orientadas a la interacción con el cliente y el desarrollo incremental del *software*, mostrando versiones parcialmente funcionales del *software* al cliente en determinados intervalos de tiempo, para que pueda evaluar y sugerir cambios en el producto.

La Tabla 3 recoge esquemáticamente estas diferencias que no se refieren solo al proceso en sí, sino también al contexto de equipo y organización que es más favorable a cada uno de estas filosofías de procesos de desarrollo de *software*.

Metodologías Ágiles	***Metodologías Tradicionales***
Basadas en heurísticas proveniente de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (menos de diez integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Muchos artefactos.
Pocos roles.	Muchos roles.
Menos énfasis en la arquitectura de *software*.	La arquitectura de *software* es esencial y se expresa mediante modelos.

Tabla 3 Diferencias entre metodologías ágiles y tradicionales. *(26)*

Ventajas del uso de una metodología

Son muchas las ventajas que puede aportar el uso de una metodología. A continuación se van a exponer algunas de ellas, clasificadas desde distintos puntos de vista.

Desde el punto de vista de gestión:

- ✓ Facilita la tarea de planificación.
- ✓ Facilita la tarea del control y seguimiento de un proyecto.
- ✓ Mejora la relación coste/beneficio.
- ✓ Optimiza el uso de recursos disponibles.
- ✓ Facilita la evaluación de resultados y cumplimiento de los objetivos.
- ✓ Facilita la comunicación efectiva entre usuarios y desarrolladores.

Desde el punto de vista de los ingenieros del software:

- ✓ Ayuda a la comprensión del problema.
- ✓ Optimiza el conjunto y cada una de las fases del proceso de desarrollo.
- ✓ Facilita el mantenimiento del producto final.
- ✓ Permite la reutilización de partes del producto.

Desde el punto de vista del cliente o usuario:

- ✓ Garantía de un determinado nivel de calidad en el producto final.
- ✓ Confianza en los plazos de tiempo fijados en la definición del proyecto.
- ✓ Define el ciclo de vida que más se adecue a las condiciones y características del desarrollo.

A partir de las características analizadas de ambas metodologías anteriormente expuestas, se selecciona el enfoque propuesto por las metodologías ágiles para guiar el proceso de desarrollo del *software*. La base de esta elección es la siguiente:

- ✓ El equipo de desarrollo es pequeño, por lo que es necesario realizar todo el desarrollo con pocos roles.
- ✓ Se dispone de poco tiempo de desarrollo (entre 5 y 7 meses).
- ✓ Alta probabilidad de que los requisitos de *software* estén en constante cambio, por lo que se necesita un proceso flexible.
- ✓ La existencia de un cliente comprometido y exigente con el desarrollo, por lo que se hace necesario realizar liberaciones frecuentes del *software*.

Seleccionándose como metodología de desarrollo ágil Variación *AUP-UCI*. Porque logra estandarizar el proceso de desarrollo de *software* hablando un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos.

Variación AUP-UCI

El Proceso Unificado Ágil de Scott Ambler o *Agile Unified Process* (*AUP*) en inglés, es una versión simplificada del Proceso Unificado de *Rational* (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio, usando técnicas ágiles y conceptos que aún se mantienen válidos en *RUP*.

De las 4 fases que propone *AUP* (Inicio, Elaboración, Construcción, Transición), se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de *AUP* en una sola, a la cual se le llama Ejecución y se agrega la fase de Cierre. (27)

Adaptada a las características de la UCI quedaría de la siguiente manera:

Fases AUP	***Fases Variación AUP-UCI***	***Objetivos de las fases(Variación AUP-UCI)***
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
Construcción		
Transición		
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Tabla 4 Faces de AUP.

1.6 Conclusiones

En este capítulo se investigaron, definieron y analizaron los conceptos básicos que tienen relación con el diseño teórico planteado en la introducción, además se realizó un estudio del estado del arte a nivel nacional e internacional sobre los sistemas similares al que se desarrolla en el presente trabajo. También se describieron las metodologías, lenguajes y tecnologías utilizadas en la elaboración del *software*, explicando sus principales características y ventajas.

Capítulo 2: Análisis de la solución propuesta

En el presente capítulo se procede a realizar la descripción de la propuesta de solución. Se particularizan las funcionalidades y características del sistema a desarrollar, en conjunto con su arquitectura base y diagrama de despliegue a partir de las tecnologías dadas para su construcción.

2.1 Personal relacionado con el sistema

Personal relacionado con el sistema	
Actores	**Justificación**
Paramédico	Persona encargada de tomar los signos vitales del paciente.
Doctor	Persona que recibe la información del paciente para darle tratamiento.

Tabla 5 Personal que interactúa con el sistema.

2.2 Descripción de la propuesta de solución

La propuesta de solución consiste en desarrollar un *firmware* que capture los SV de la persona usando la plataforma de *hardware* libre Arduino, y además envíe los datos mediante *bluetooth* a un dispositivo móvil que este seguidamente se encarga de enviárselo por medio de un mensaje, usando el protocolo *GPRS*, al médico.

La aplicación brinda facilidad de uso para cualquier usuario. Se diferencia de otros SMSV, en que presenta la ventaja de ser portable, lo que permitirá realizar dichas operaciones desde una ambulancia, generando un mayor aprovechamiento del tiempo para darle atención al paciente.

El usuario luego de acceder a la aplicación podrá hacer uso de las opciones que se brinda:

- ✓ Vincular Dispositivos: Encargado de listar los dispositivos *bluetooth* vinculados con el teléfono.
- ✓ Enviar datos: Encargado de enviar los datos de los signos vitales recibidos del Arduino.

Esta se conforma de dos áreas fundamentales. En la primera se encuentra la ambulancia, la cual dispone del Arduino que realiza la captura de los SV al paciente usando los sensores correspondientes y por medio de un módulo de *bluetooth* envía los datos recogidos al dispositivo Android.

La segunda es en el hospital, donde se encuentra el médico que recibirá la información de los signos vitales de teléfono a teléfono por medio de la red G disponible en nuestro país.

2.3 Disciplina modelado de negocio

Es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el *software* desarrollado va a cumplir su propósito. Para modelar el negocio se proponen las siguientes variantes:

- ✓ Casos de Uso del Negocio (CUN).
- ✓ Descripción de Proceso de Negocio (DPN).
- ✓ Modelo Conceptual (MC).

A partir de las variantes anteriores se condicionan cuatro escenarios para modelar el sistema en la disciplina Requisitos, seleccionándose el escenario cuatro que es el que se adecúa a las características del sistema.

2.4 Disciplina Requisitos

Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto. Existen tres formas de encapsular los requisitos:

- ✓ Casos de Uso del Sistema (CUS)
- ✓ Historias de usuario (HU)
- ✓ Descripción de requisitos por proceso (DRP)

Requisitos del sistema

Unos de los puntos fundamentales en el desarrollo de sistemas son la correcta obtención de los requisitos o ingeniería de requerimientos, la cual trata de establecer las funcionalidades fundamentales que debe cumplir el sistema, así como sus restricciones. Los requerimientos para un sistema son la descripción de los servicios proporcionados

por el sistema y sus restricciones operativas. Estos requerimientos reflejan las necesidades de los clientes de realizar un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información. (28)

Requisitos funcionales

Son declaraciones de los servicios que debe brindar el sistema y la manera en que deben de reaccionar ante una situación particular o estimulo del exterior.

A continuación se presentan los requisitos funcionales del sistema:

RF 1: tomar signos vitales.

RF 2: activar *bluetooth* en dispositivo Android.

RF 3: conectar Arduino con dispositivo Android.

RF 4: pasar los signos vitales del Arduino al dispositivo Android.

RF 5: mostrar en pantalla datos recibidos.

RF 6: enviar signos vitales de dispositivo a dispositivo.

Requisitos no funcionales

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. (28)

Para la aplicación fueron capturados los siguientes requisitos no funcionales:

Atributo de Calidad	Usabilidad
Sub-Atributos/Sub-Características	
Objetivo	La solución debe ser de fácil comprensión, configuración, y utilización por los paramédicos encargados del sistema.
Origen	Externo
Artefacto	SMSV
Entorno	Ejecución del Sistema.
Estimulo	***Respuesta: Flujo de eventos (Escenarios)***
1.a Iniciar la aplicación	
	La aplicación a la espera de los clientes.

Medida de respuesta

Tabla 6 Requisito no funcional 1.

Atributo de Calidad	Adaptabilidad
Sub-Atributos/Sub-Características	
Objetivo	La aplicación debe funcionar en las versiones de Android a partir de la 2.3.5.
Origen	Externo
Artefacto	SMSV
Entorno	Ejecución del Sistema.
Estimulo	***Respuesta: Flujo de eventos (Escenarios)***
1.a Iniciar la aplicación	
	Se inicia la aplicación normalmente.
Medida de respuesta	
Independientemente del entorno donde se esté ejecutando la aplicación, esta debe mostrarse correctamente y permitir acceder a todas las funcionalidades que brinda.	

Tabla 7 Requisito no funcional 2.

Atributo de Calidad	Portabilidad
Sub-Atributos/Sub-Características	
Objetivo	La aplicación debe ser de fácil movilidad y utilizar poco espacio de almacenamiento.
Origen	Externo
Artefacto	SMSV
Entorno	
Estimulo	***Respuesta: Flujo de eventos (Escenarios)***
1.a Interacción con el Sistema	
Medida de respuesta	

Tabla 8 Requisito no funcional 3.

Validación de requisitos

Con el objetivo de verificar si los requisitos del *software* obtenidos definen el sistema que el cliente desea, se llevó a cabo un proceso de validación de los mismos, para el cual se empleó la siguiente técnica:

- ✓ **Revisión de los requisitos:** se realizaron revisiones a cada uno de los requisitos por parte del equipo de desarrollo y por el cliente (AREX). Las revisiones internas generaron un total de 3 no conformidades de tipo técnicas (error al enviar el mensaje, fallaba la conexión del teléfono con el módulo de *bluetooth*, la vista principal no satisfacía el gusto del cliente), las cuales fueron corregidas satisfactoriamente en tiempo y aprobadas por el cliente.

De los cuatro escenarios propuesto por *AUP* el No.4 es el más acorde al sistema, por las características que se describen a continuación.

2.4.1 Escenario No.4

Proyectos que no modelen negocio solo pueden modelar el sistema con HU. Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información.

Historias de Usuario

Se utilizan para especificar los requisitos de las aplicaciones *software* en las metodologías ágiles (*SCRUM*, *XP*, *FDD*, *ASD*, *AUP*, *LD*, etc.). Las historias de usuarios son tarjetas en dónde el interesado describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que se pueda implementar en unas tres semanas aproximadamente, para no superar el tamaño de una iteración, que es el tiempo estimado para una entrega de un componente de desarrollo de manera incremental. Las historias de usuario se descomponen en tareas de programación (*task card*) que los programadores implementan y se representan en tablas que contienen los siguientes campos: (29)

- ✓ **Número:** Las siglas de HU más un número consecutivo, este permite Identificar la historia.
- ✓ **Usuario:** Rol encargado.

- ✓ **Nombre de historia:** Nombre que identifica la HU.
- ✓ **Prioridad en negocio:** Esta característica es dada por el cliente con los valores: alta, media o baja en dependencia de la importancia y orden en que desean que sean implementadas.
- ✓ **Riesgo en Desarrollo:** Depende de la complejidad que posea la funcionalidad que se va a desarrollar.
- ✓ **Tiempo Estimado:** Tiempo estimado en semanas que se le asignará.
- ✓ **Iteración Asignada:** Número de la iteración en la cual se desarrollará la HU.
- ✓ **Programador responsable:** Persona que se encarga de desarrollar la HU.
- ✓ **Descripción:** Breve descripción del proceso que define la historia.
- ✓ **Observaciones:** Alguna acotación importante de señalar acerca de la historia.

Historia de Usuario	
Número: 1	**Usuario:** Desarrollador
Nombre historia: Capturar signos vitales	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Tiempo estimado: 4	**Iteración asignada:** 1
Programador responsable: Jorge Lazaro Amaro	
Descripción: El paramédico le medirá los signos vitales al paciente mediante el Arduino y los sensores ya instalado previamente en la ambulancia, además de practicarle los primero auxilios en caso necesario.	
Observaciones:	

Tabla 9 HU-Tomar signos vitales.

Historia de Usuario

Número: 2	**Usuario:** Desarrollador
Nombre historia: Activar *bluetooth* en el dispositivo Android.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Tiempo estimado: 2	**Iteración asignada:** 2
Programador responsable: Jorge Lazaro Amaro	
Descripción: Al iniciar la aplicación el sistema debe ser capaz de brindar la opción de activar el *bluetooth*.	
Observaciones:	

Tabla 10 HU-Activar bluetooth en el dispositivo Android.

Historia de Usuario	
Número: 3	**Usuario:** Desarrollador
Nombre historia: Conectar Arduino con dispositivo Android.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Tiempo estimado: 2.5	**Iteración asignada:** 1
Programador responsable: Jorge Lazaro Amaro	
Descripción: El sistema luego de haber obtenido los signos vitales del paciente debe ser capaz de conectarse con el dispositivo Android para enviar los datos.	

Observaciones:

Tabla 11 HU-Conectar Arduino con dispositivo Android.

Historia de Usuario	
Número: 4	**Usuario:** Desarrollador
Nombre historia: Mostrar en pantalla datos recibidos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Tiempo estimado: 3.5	**Iteración asignada:** 2
Programador responsable: Jorge Lazaro Amaro	
Descripción: Luego de haber conectado el dispositivo Android con el Arduino, el sistema debe mostrar en pantalla los datos enviados por *bluetooth* desde el Arduino hacia el teléfono.	
Observaciones:	

Tabla 12 HU-Mostrar en pantalla datos recibidos.

Historia de Usuario	
Número: 5	**Usuario:** Desarrollador
Nombre historia: Enviar información de los signos vitales	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Media
Tiempo estimado: 2	**Iteración asignada:** 1

Programador responsable: Jorge Lazaro Amaro
Descripción: El paramédico luego de haber tomado los signos vitales del paciente, envía la información obtenida mediante un dispositivo móvil al médico encargado del caso ubicado en el hospital.
Observaciones:

Tabla 13 HU-Enviar información de signos vitales.

Se confeccionaron 5 HU, tomando para los puntos estimados la unidad como una semana de trabajo. Se definieron 3 HU con prioridad alta y 2 HU con prioridad media.

2.5 Estimación de esfuerzo por Historias de Usuario

Las historias de usuarios seleccionadas para entregar en cada iteración, son desarrolladas y probadas en su ciclo correspondiente, teniendo en cuenta el orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. A continuación, en la tabla se resume la estimación del esfuerzo realizada por parte de los desarrolladores.

Historias de usuario	*Estimación por puntos*
Capturar signos vitales	4.0
Activar *bluetooth* en el dispositivo Android	2.0
Conectar Arduino con dispositivo Android	2.5
Mostrar en pantalla datos recibidos	3.5
Enviar información de los signos vitales	2.0

Tabla 14 Estimación por puntos.

2.6 Plan de iteraciones

Como resultado de determinar dicho flujo, aparece la plantilla plan de iteraciones, en la que se recogen las iteraciones a realizar con sus características, además del orden de las historias de usuario con su planificación estimada para ser implementadas.

En la tabla plan de iteraciones, se incluyen los siguientes campos: **Número de iteración**: que contiene el identificador de la iteración que se va a desarrollar; **Descripción de la iteración**: breve descripción del objetivo de la iteración; **Orden**: contiene los identificadores de las HU a implementar en la iteración, en el mismo orden en que se deben realizar; y **Duración total**: cantidad de semanas que durará realizar la iteración, la que depende del tiempo estimado de las HU propuestas.

Número de iteración	***Descripción de la iteración***	***Orden***	***Duración total***
Iteración 1	En esta iteración se desarrollan las funcionalidades priorizadas como altas, es decir, las HU que se consideran con una mayor complejidad de desarrollo.	HU-1 HU-3 HU-4	10
Iteración 2	En esta iteración se desarrollan las funcionalidades priorizadas como medias y bajas, es decir, las HU que tienen un nivel de complejidad de desarrollo medio o bajo.	HU-2 HU-5	4

Tabla 15 Iteraciones.

2.6.1 Plan de entrega

A partir de las iteraciones realizadas anteriormente se procede a elaborar el plan de entregas. El mismo tiene como objetivo definir las diferentes entregas y orden que tendrán las mismas.

Historia de Usuario	***Final 1ra Iteración 24 de marzo***	***Final 2da Iteración 12 de mayo***
Capturar Signos Vitales	V 1.0	
Activar *bluetooth* en el dispositivo Android		V 1.1
Conectar Arduino con dispositivo Android.	V 1.0	

Mostrar en pantalla datos recibidos	V 1.0	
Enviar información de los signos vitales		V 1.1

Tabla 16 Plan de Entrega.

2.7 Disciplina Análisis y diseño

En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura).

2.7.1 Descripción de la arquitectura del sistema

Debido a que los sistemas de *software* crecen de forma tal que resulta muy complicado que sean diseñados, especificados y entendidos por un solo individuo, se hace necesaria e importante la arquitectura de *software* como disciplina. Esta puede ser vista como: (30)

- ✓ Estructura del sistema en función de la definición de los componentes y sus interacciones.
- ✓ Puente entre los requisitos del sistema y la implementación.
- ✓ Plan de diseño del sistema, debido a que es usada como guía para el resto de las tareas de la etapa de desarrollo.

Para la creación del sistema se utilizó el patrón n-capas en el dispositivo móvil, que luego se conecta a la plataforma Arduino por medio de la tecnología *bluetooth.*

El módulo de *bluetooth* del teléfono se conecta al módulo de *bluetooth* que está conectado en el Arduino. Posteriormente la información recibida es mostrada en la capa de presentación, para luego en la capa de negocio preparar los datos que serán enviados en la de comunicación hacia el otro dispositivo móvil a través del protocolo GPRS.

En la plataforma Arduino se representa el código de forma estructurada y no responde a ninguna arquitectura. Aquí está presente la configuración de los sensores para capturar los signos vitales, se realiza la recopilación de los datos provenientes del mismo, a través de una función se lleva a cabo un procesamiento de esos datos y

finalmente haciendo uso de las librerías <SoftwareSerial.h> por vía serial y la tecnología bluetooth se realiza la conexión con el dispositivo móvil. Para una mejor compresión se representa la descripción antes expuesta en la siguiente figura.

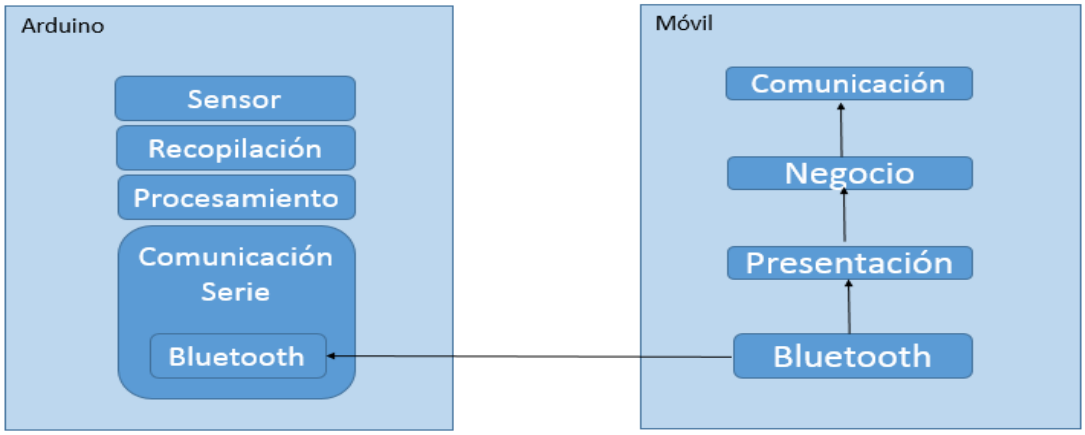

Ilustración 15 Arquitectura del sistema.

2.7.2 Patrón de diseño

Un patrón de diseño nombra, abstrae e identifica los aspectos clave de un diseño estructurado común, que lo hace útil para la creación de diseños orientados a objetos reutilizables. Definen una posible solución correcta para un problema de diseño dentro de un contexto dado, describiendo las cualidades invariantes de todas las soluciones.

Además del uso del patrón arquitectónico n-capas, en la solución propuesta se emplea otro tipo, el de diseño:

Patrones de Diseño de Asignación de Responsabilidades o GRASP (Patrones Generales de Software para Asignar responsabilidades)

Experto

Este patrón es aplicado en todas las clases debido a que cada una de ellas es experta pues contienen la información necesaria para cumplir con las responsabilidades que le fueron asignadas; facilitando así el entendimiento de la misma. Ejemplo de este patrón se evidencia en la clase *SMSActivity*. (31)

2.8 Diagrama de despliegue

Los diagramas de despliegue muestran la disposición física de los distintos nodos que entran en la composición de un sistema y el reparto de los programas ejecutables sobre estos nodos. Todo sistema se describe con uno o más diagramas de despliegue. (32)

El diagrama de despliegue de la aplicación representa tres nodos interconectados (Arduino micro, Dispositivo móvil en la ambulancia y Dispositivo móvil en el hospital). En el Arduino está cargado el *firmware* que mediante los sensores, captura los signos vitales de la persona y utilizando un módulo de *bluetooth* se comunica con el dispositivo móvil que se encuentra disponible en la misma. Este dispositivo móvil proporciona una interfaz para recibir, mostrar y enviar los datos hacia el otro teléfono en el hospital usando el protocolo *GPRS*[1]. En la Fig. 8 se visualiza el diagrama de despliegue de la solución que se propone.

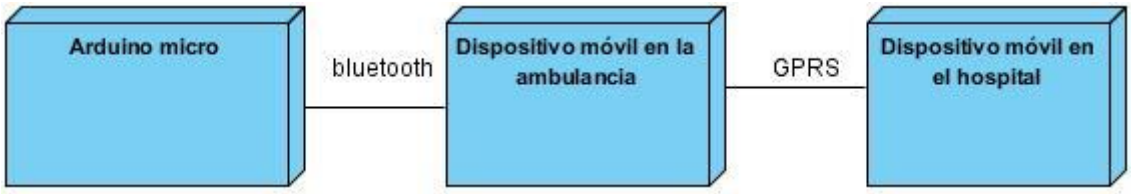

Ilustración 16 Diagrama de despliegue.

2.9 Conclusiones parciales

En este capítulo se analizaron aspectos correspondientes a la propuesta de solución para el desarrollo de la investigación. Se describieron de manera detallada los requerimientos funcionales y la arquitectura propuesta, que permitieron obtener un diseño robusto de la aplicación. Además las historias de usuario sentaron las bases para la implementación de la misma y se aplicaron los patrones necesarios para lograr un buen diseño. Finalmente se representó el diagrama de despliegue que muestra cómo están distribuidos los dispositivos de *software* entre los diferentes nodos para un mayor entendimiento de cómo funciona el negocio.

[1] GPRS: General Packet Radio Service (Paquete General de Radio Servicio)

Capítulo 3: Implementación y pruebas

En el actual capítulo se describen los resultados obtenidos durante la ejecución de las pruebas, teniendo en cuenta el flujo de implementación del sistema, con el objetivo de definir la organización del código, la implementación de los elementos de diseño y la integración de los diferentes componentes, generando un ejecutable entregable o producto final. Cabe señalar que luego de su culminación, este debe cumplir con todos los requisitos funcionales y no funcionales del *software*.

3.1 Implementación

La implementación constituye una de las fases más importantes del desarrollo de *software*. En ella se toman como punto de partida los resultados obtenidos en el diseño, implementándose el sistema en términos de componentes como ficheros de código binario, código fuente, *scripts* y ejecutables. Su importancia reside en que se obtiene como consecuencia un sistema ejecutable, siendo esto uno de los principales objetivos en el desarrollo de *software*. (33)

3.1.1 Tareas de Ingeniería o programación

Para alcanzar los objetivos de una iteración es necesario completar las HU que están presentes en estas, por lo que se hace necesario saber cuáles son las tareas de ingeniería que componen cada una, las cuales harán posible el cumplimiento de los objetivos de cada HU. En ellas están presentes los siguientes campos: **Número de tarea:** que contiene un número consecutivo en base a la historia de usuario correspondiente; **Número historia de usuario:** que contiene el identificador de la HU a la que pertenece esta tarea; **Nombre tarea:** contiene un nombre que identifica a la tarea; **Tipo de tarea:** contiene el tipo de tarea, que puedes ser de desarrollo, corrección, mejora, o la especificación de otra; **Puntos estimados:** estimación en semanas de la duración de la tarea; **Fecha inicio:** contiene la fecha de inicio de la tarea; **Fecha fin:** contiene la fecha de fin de la tarea y **Descripción:** que contiene la descripción de la tarea.

Tareas por HU de la iteración 1

Numero	*Historia de Usuario*	*Tareas por Historias de Usuario*
1	Capturar Signos Vitales.	1. Configurar los sensores encargados de capturar los signos vitales.
3	Conectar Arduino con dispositivo Android.	1. Implementar la clase encargada de realizar la conexión con el módulo de *bluetooth*. 2. Crear la vista correspondiente para conectar el teléfono móvil con el módulo de *bluetooth*.
4	Mostrar en pantalla datos recibidos.	1. Implementar la clase encargada de recibir la información y mostrarla en pantalla. 2. Crear la vista correspondiente para mostrar los datos recibidos por *bluetooth*.

Tabla 17 Tareas por HU.

Tarea 1 de HU-1

Tarea de ingeniería	
Numero de tarea: 1	**Numero de Historia de Usuario:** 1
Nombre de la tarea: Configurar los sensores encargados de capturar los signos vitales.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 4.0
Fecha inicio: 6/2/2016	**Fecha fin:** 6/3/2016
Descripción: Se configuran los sensores para capturar los signos vitales.	

Tabla 18 Tarea 1.

Tarea 1 de HU-3

Tarea de ingeniería	
Numero de tarea: 1	**Numero de Historia de Usuario**: 3

Nombre de la tarea: Implementar la clase encargada de realizar la conexión con el módulo de *bluetooth*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.25
Fecha inicio: 6/3/2016	**Fecha fin:** 15/3/2016
Descripción: Se implementa la clase para conectar el teléfono móvil con el modulo *bluetooth*.	

Tabla 19 Tarea 1.

Tarea 2 de HU-3

Tarea de ingeniería	
Numero de tarea: 2	**Numero de Historia de Usuario:** 3
Nombre de la tarea: Crear la vista correspondiente para conectar el teléfono móvil con el módulo de *bluetooth*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.25
Fecha inicio: 16/3/2016	**Fecha fin:** 23/3/2016
Descripción: Se implementa la vista que muestra los botones necesarios para conectar el teléfono con el módulo de *bluetooth*.	

Tabla 20 Tarea 2.

Tarea 1 de HU-4

Tarea de ingeniería	
Numero de tarea: 1	**Numero de Historia de Usuario:** 4
Nombre de la tarea: Implementar la clase encargada de recibir la información y mostrarla en pantalla.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.75
Fecha inicio: 24/3/2016	**Fecha fin:** 1/3/2016
Descripción: Se implementa la clase que recibe los datos por *bluetooth* y los muestra en la pantalla.	

Tabla 21 Tarea 1.

Tarea 2 de HU-4

Tarea de ingeniería	
Numero de tarea: 2	**Numero de Historia de Usuario:** 4

Nombre de la tarea: Crearla vista correspondiente para mostrar los datos recibidos por *bluetooth*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.75
Fecha inicio: 4/4/2016	**Fecha fin:** 18/4/2016
Descripción: Se implementa la vista que recibe los datos por *bluetooth* y los muestra en la pantalla.	

Tabla 22 Tarea 2.

Tareas por HU de la iteración 2

Numero	*Historia de Usuario*	*Tareas por Historias de Usuario*
2	Activar *bluetooth* en el dispositivo Android.	1. Implementar la clase encargada de realizar la activación del *bluetooth* en el teléfono móvil.
5	Enviar información de los signos vitales	1. Implementar la clase encargada de enviar el mensaje con los datos. 2. Crear la vista correspondiente para enviar los datos.

Tabla 23 Tareas por HU.

Tarea 1 de HU-2

Tarea de ingeniería	
Numero de tarea: 1	**Numero de Historia de Usuario:** 2
Nombre de la tarea: Implementar la clase encargada de realizar la activación del *bluetooth* en el teléfono móvil.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2.0
Fecha inicio: 20/4/2016	**Fecha fin:** 3/5/2016
Descripción: Se implementa la clase que activa el *bluetooth* en el teléfono móvil.	

Tabla 24 Tarea 1.

Tarea 1 de HU-5

Tarea de ingeniería	
Numero de tarea: 1	**Numero de Historia de Usuario:** 5

Nombre de la tarea: Implementar la clase encargada de enviar el mensaje con los datos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha inicio: 10/52016	**Fecha fin:** 17/5/2016
Descripción: Se implementa la clase que envía los datos de los signos vitales en un mensaje.	

Tabla 25 Tarea 1.

Tarea 2 de HU-5

Tarea de ingeniería	
Numero de tarea: 2	***Numero de Historia de Usuario:*** 5
Nombre de la tarea: Crear la vista correspondiente para enviar los datos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha inicio: 18/5/2016	**Fecha fin:** 25/5/2016
Descripción: Se implementa la vista que contiene los campos necesarios para enviar los datos en el mensaje.	

Tabla 26 Tarea 2.

3.1.2 Estándar de código

Un estándar de programación es una forma de "normalizar" la programación para que al trabajar en un proyecto cualquiera, las personas involucradas en el mismo tengan un acceso rápido y una correcta comprensión del código. Un estándar nos define la escritura y organización del código fuente de un programa.

Ventajas que brinda:

- ✓ Facilitan el mantenimiento de una aplicación. Dicho mantenimiento constituye el 80% del coste del ciclo de vida de la aplicación.
- ✓ Permite que cualquier programador entienda y pueda mantener la aplicación. En muy raras ocasiones una misma aplicación es mantenida por su autor original.
- ✓ Los estándares de programación mejoran la legibilidad del código, al mismo tiempo que permiten su compresión rápida.

Para la implementar el presente sistema se utilizó el siguiente estándar de código:

1. *Clases:*

a. Para todo nombre de clase, la primera letra debe ser Mayúscula, si son varias palabras se debe de intercalar entre mayúsculas y minúsculas, este mecanismo de nombre es llamado ***carmelCase***. Por ejemplo: *MainActivity*.

b. Intentar mantener los nombres de las clases descriptivos y simples. Usar palabras completas, evitar acrónimos y abreviaturas, a no ser que la abreviatura sea mucho más conocida que el nombre completo,

```
public class MainActivity extends Activity {

    private Button b, btnDis;


    private StringBuilder recDataString = new StringBuilder();
    Handler bluetoothIn;

    private ConnectedThread mConnectedThread;
```

Ilustración 17 Nombre de clase.

2. *Métodos:*

a. Para los métodos de clases, la primera letra debe ser minúscula, si son varias palabras se debe intercalar entre minúsculas y mayúsculas, para el caso de los métodos de clases aplica el mecanismo del ***cemelCase***. Por ejemplo: *getNombre*, *setOnClickListener*.

```
b = (Button) findViewById(R.id.lachy3);
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent(MainActivity.this, DeviceList.class));
    }
});

btnDis.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Disconnect(); //close connection
    }
});
```

Ilustración 18 Nombre del método.

3. *Sangría*:
    b. Como norma general se establecen 4 caracteres como unidad de sangría.

```
public class MainActivity extends Activity {

    private Button b, btnDis;

    private StringBuilder recDataString = new StringBuilder();
    Handler bluetoothIn;

    private ConnectedThread mConnectedThread;


    String address = null;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;

    final int handlerState = 0;
    TextView txtString;
    //SPP UUID. Look for it
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");


    @Override
```

Ilustración 19 Sangría.

4. *XML*: Todos los elementos *XML* deben tener una etiqueta de cierre.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Paried Device"
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
```

Ilustración 20 Etiqueta de cierre.

3.2 Disciplina Pruebas

La prueba es un proceso de ejecución de un programa con la intención de descubrir un error, no puede asegurar la ausencia de defectos, sino demostrar la presencia de los mismos en el *software*, por lo que constituye el único instrumento adecuado para determinar el status de la calidad de un producto. En este proceso se ejecutan pruebas

dirigidas a componentes del *software* o al sistema en su totalidad, con el objetivo de medir el grado en que este cumple con los requerimientos. En ellas se usan casos de prueba, especificados de forma estructurada mediante diferentes técnicas.

Entre sus objetivos están:

- ✓ Verificar que todos los requisitos se han implementado correctamente.
- ✓ Detectar defectos en el *software*.
- ✓ Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el *software* al cliente.
- ✓ Diseñar casos de prueba que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo en el menor tiempo y esfuerzo posible.

3.2.1 Pruebas Internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción incluyendo tanto las construcciones internas como las intermedias, así como las versiones finales a ser liberadas. Se genera el artefacto de prueba diseño de casos de prueba. (27)

Pruebas de aceptación

Las pruebas de aceptación son pruebas funcionales, pero vistas directamente desde el cliente, demostrándole que la funcionalidad está terminada y correcta, es decir, se realizan con el fin de verificar si el producto ha sido desarrollado de acuerdo con las normas o criterios establecidos y que cumplen con todos los requisitos especificados por el cliente.

Uno de los métodos utilizados en la realización de esta prueba es el de **caja negra**, el cual se centra en lo que se espera de un módulo, intentando encontrar casos en que el sistema no se atiene a su especificación. El probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo internamente. (34)

A continuación se visualizan las pruebas utilizando el dispositivo móvil.

Caso de Prueba	Activar *bluetooth* en el dispositivo Android.

Descripción de la prueba.	Se inicia la aplicación, posteriormente el sistema debe comenzar la activación del *bluetooth.* Inicialmente será sin tener activado el *bluetooth* con anterioridad para comprobar que la aplicación muestra los mensajes correspondientes, luego en el caso contrario para comprobar el correcto funcionamiento sin activar el *bluetooth*.
Entrada	-
Resultado Esperado	La aplicación debe mostrar el siguiente mensaje "una aplicación de teléfono está solicitando permiso para activar la función *bluetooth*. ¿Quiere realizar esta opción?", y seguidamente "Activando bluetooth" en caso de tener activado el mismo previamente no debe mostrar ningún mensaje.
Resultado de la prueba	Satisfactorio.
Condiciones	-

Tabla 27 HU Activar bluetooth en el dispositivo Android.

Caso de Prueba	Conectar dispositivo Android con Arduino.
Descripción de la prueba.	Se presiona el botón vincular dispositivos en la vista correspondiente, en la lista de dispositivos *bluetooth* que aparecerá se seleccionara modulo corresponde. Inicialmente será con un dispositivo vinculado previamente para comprobar que la aplicación se conecta, luego sin dispositivos vinculados para comprobar las validaciones de la aplicación.
Entrada	-
Resultado Esperado	La aplicación debe mostrar un mensaje que diga "*Conect*" en caso de tener dispositivos disponibles en la lista. En caso contrario debe mostrar un mensaje que diga: "No hay dispositivos disponibles para vincular".
Resultado de la prueba	Satisfactorio.
Condiciones	-

Tabla 28 HU Conectar Arduino con dispositivo Android.

Caso de Prueba	Mostrar en pantalla datos recibidos.
Descripción de la prueba.	Una vez establecida la conexión con el módulo *bluetooth*, se debe mostrar en pantalla los datos de los signos vitales que enviará el Arduino por medio de dicho módulo.
Entrada	-
Resultado Esperado	La aplicación debe mostrar los datos recibidos, en caso contrario debe mostrar un mensaje: "No hay *string* para para leer".
Resultado de la prueba	Satisfactorio.
Condiciones	- El Arduino debe enviar los datos.

Tabla 29 HU Mostrar en pantalla datos recibidos.

Caso de Prueba	Enviar información de signos vitales.
Descripción de la prueba.	Se presiona el botón enviar datos de la vista correspondiente, se introduce el número de teléfono del destino y el texto con la información de los signos vitales.
Entrada	Número de teléfono del destino y texto con la información de los signos vitales.
Resultado Esperado	La aplicación debe mostrar un mensaje que diga "*Send*".
Resultado de la prueba	Satisfactorio.
Condiciones	Debe ser válido el número del destino.

Tabla 30 HU Enviar información de signos vitales.

Resultado de las pruebas

La siguiente figura muestra los resultados obtenidos luego de aplicar las pruebas.

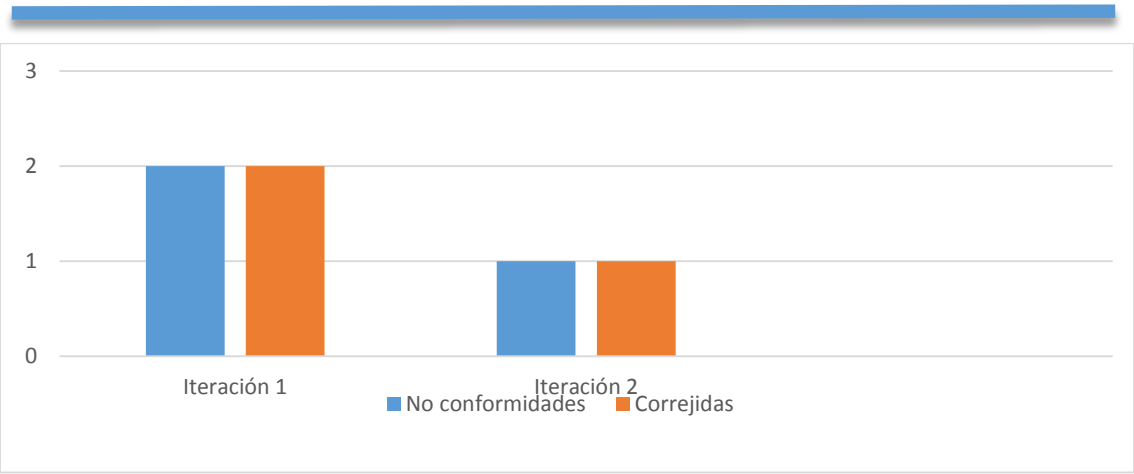

Ilustración 21 Resultado de las pruebas.

Para comprobar que con la realización de la aplicación se logró mejorar la atención al paciente, se tuvieron en cuenta los siguientes aspectos: (26)

Criterios	Resultados del desarrollo de la aplicación
Corrección	Cumple con las funciones requeridas y se realizaron de forma correcta.
Usabilidad	Es "amigable al usuario", debido a que resulta fácil la interacción y comprensión del sistema.
Facilidad de prueba	Es posible asegurarse fácilmente mediante la realización de pruebas, que el sistema cumple con las funciones pretendidas.
Portabilidad	Se puede mover de un dispositivo móvil a otro debido al poco espacio de almacenamiento que requiere.
Consistencia	Se utilizó un estándar de código.

Tabla 31 Resultados del desarrollo de la aplicación.

3.3 Valoración del comportamiento de las variables de la investigación

Antes del desarrollo del sistema, el médico no tenía la forma de conocer el estado fisiológico del paciente antes de llegar al hospital. Esto provocaba una pérdida de tiempo muy valiosa a la hora de dar tratamiento. Además, los SMSV que existen actualmente en Cuba, no contaban con la ventaja de comunicarse con dispositivos móviles.

Todas estas deficiencias afectaban directamente la mejora de la atención al paciente en Cuba**.**

Variable Dependiente: mejorar la atención al paciente en Cuba.

Variable Independiente: Sistema de medición de Signos Vitales basado en Arduino y telefonía móvil.

A partir de los resultados del diagnóstico inicial obtenido una vez desarrollada la aplicación, se arriba a la siguiente conclusión: se logró mejorar la atención al paciente en Cuba, lo cual se evidencia en los resultados emitidos por el sistema luego de realizar las pruebas.

Se registra una reducción considerable de tiempo en la atención al paciente ya que el médico conoce su estado con antelación, permitiendo crear las condiciones necesarias antes de su llegada para darle tratamiento.

3.4 Conclusiones parciales

En este capítulo se definieron varios aspectos que son importantes para el desarrollo de la aplicación. Se presenta un estándar de código con el objetivo de alcanzar una buena organización y también fueron realizados casos de prueba utilizando el método de caja negra, verificando que se cumplieron los requerimientos y funcionalidades del sistema para proporcionar un nivel de calidad de pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones iníciales del sistema.

Conclusiones Generales

Al finalizar el desarrollo de la presente investigación se puede arribar a las siguientes conclusiones:

- ✓ Se realizó el marco teórico de la investigación, donde se establecieron las tendencias marcadas en cuanto al desarrollo de *software* para medición de signos vitales y las herramientas que estos utilizan, lo que permitió fundamentar la necesidad de desarrollar dicho sistema.
- ✓ Se desarrolló un sistema de medición de signos vitales que permitió capturar los signos vitales garantizando un mayor aprovechamiento del tiempo para atender al paciente.
- ✓ Las pruebas realizadas a sistema, demuestran que es posible su despliegue, cumpliendo con los requisitos definidos por el cliente.

Recomendaciones

Para dar continuidad a la presente investigación se recomienda lo siguiente:

- ✓ Añadir al sistema las funcionalidades de observación y monitoreo al paciente.
- ✓ Migrar de la conexión por *bluetooth* a *Wifi* teniendo en cuenta las nuevas tendencias.
- ✓ Usar *bluetooth* de menor consumo para ahorrar batería en el teléfono móvil.
- ✓ Agregar la funcionalidad de localización geográfica al sistema.

Glosario de términos

A

***APIs*:** es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software. Las siglas API vienen del inglés Application Programming Interface. En español sería Interfaz de Programación de Aplicaciones.

Arduino: es una plataforma de prototipos electrónica de código abierto basada en *hardware* y *software* libre.

Arritmia: un trastorno del ritmo cardíaco o arritmia cardíaca, es una alteración en la sucesión de latidos cardíacos. Puede deberse a cambios en la frecuencia cardíaca, tanto porque se acelere o disminuya, que no son necesariamente irregulares sino más rápidas o más lentas.

***AUP*:** acrónimo de *Agile Unified Process* (Proceso Unificado Ágil), es una metodología de desarrollo de *software*, cuyo objetivo es guiar a un equipo de proyecto durante la creación de un nuevo sistema.

B

***Bluetooth*:** el *Bluetooth* es un estándar de comunicación inalámbrica que permite la transmisión de datos a través de radiofrecuencia en la banda de 2,4 GHz. Existen muchos módulos bluetooth para usarlos en nuestros proyectos de electrónica, pero los más utilizados son los módulos de JY-MCU, ya que son muy económicos y fáciles de encontrar en el mercado. Son módulos pequeños y con un consumo muy bajo que nos permitirán agregar funcionalidades *Bluetooth* a nuestro Arduino. Estos módulos contienen el chip con una placa de desarrollo con los pins necesarios para la comunicación serie.

C

C++: su nombre fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que es una extensión de C.

CO2: acrónimo de dióxido de carbono, es un gas muy común, compuesto por dos átomos de oxígeno y uno de carbono, que no se puede ver ni olfatear pero que incluso se emite cada vez que exhala la persona.

Código abierto: en inglés; *Open Source*, es el término con el que se conoce al *software* distribuido y desarrollado libremente. El código Abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código.

F

Fisiológico: es un adjetivo que indica que algo es perteneciente o relativo a la Fisiología. Esta palabra indica, por lo tanto que algo está relacionado con el funcionamiento biológico de los seres vivos. Deriva de la palabra 'Fisiología', formada con los términos griegos *φυσις* (physis, 'naturaleza') y *λογος* (logos, 'conocimiento', 'estudio') y el sufijo '-ico', que forma adjetivos que indican relación, propiedad o pertenencia.

***Firmware*:** es un bloque de instrucciones de máquina para propósitos específicos, grabado en una

memoria, normalmente de lectura/escritura**,** que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

G

***GENOME*:** este entorno de escritorio es uno de los más conocidos que no solo está presente en Linux. También se puede encontrar en otros sistemas *Unix* como *BSD* y *Solaris*. *Gnome* (*GNU Network Object Model Environment*) tuvo su origen en los mejicanos Miguel de Icaza y Federico Mena en 1999, estando traducido actualmente en más de 166 idiomas.

***GPRS*:** (*General Packet Radio Service*). Permite como máximo 80 Kbps, es decir 0,08 "*Megas*" de velocidad. Es la conexión más lenta, pero también la de más cobertura; por poner un símil, es como poner "AM" en vez de "FM" en la radio. Se oye peor, pero alcanza más distancia. La velocidad es similar a los antiguos *módems* telefónicos de PC de 56 Kbps.

H

Hipertermia: Aumento de la temperatura corporal por encima de lo normal: 37 °C.

Hipotermia: Es una temperatura corporal peligrosamente baja, por debajo de 95ºF (35ºC).

J

***JDK*:** acrónimo de *Java Development Kit*, es un ambiente de desarrollo de *software* usado para las aplicaciones de *Java* en vías de desarrollo. Incluye el tiempo de ejecución de ambiente de *java* (*Java Runtime Environment*),un interpretador o cargador (interpreter/*loader*) (el *java*), un recopilador (javac), un archiver (frasco), un generador de la documentación (*javadoc*) y otras herramientas necesarias en el desarrollo de *Java*.

L

***Linux*:** sistema operativo libre.

M

***MacOS*:** en inglés; *Macintosh Operating System*, en español; Sistema Operativo *Macintosh*. Sistema operativo creado por la empresa Apple para sus computadoras.

multiplataforma: es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de *software*, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en los sistemas operativos *Windows*, en *GNU*/*GNU*/*Linux* y en *MacOS*.

Máquina virtual *Dalvik*: (*DVM*) permite ejecutar aplicaciones programadas en *Java*. La *DVM* no afirma ser una máquina virtual de *java* (*JVM*) debido a que le ocasionaría problemas de licenciamiento, sin embargo cumple ese propósito.

R

Red G: La G significa generación, este es un sistema de conexión de la telefonía celular que se ha ampliado y desarrollado, paralelamente con la evolución tecnológica de los propios teléfonos móviles, cuyo proceso se identifica por generaciones de acuerdo a los avances que se van introduciendo. Hasta los momentos se cuentan cuatro:(1G-Redes análogas, 2G-Globalización digital, 3G-Alta transmisión, 4G-Velocidades futuristas.

Referencias Bibliográficas

1. **Equipos de Signos Vitales y Monitoreo del Paciente. [En línea] 2015. https://www.welchallyn.com/content/welchallyn/latam/es.html.**

2. Integrales, Soluciones Medicas. Equipo médico electrónico. [En línea] 2016. http://www.monitordesignosvitales.com.mx/detalles_MONITORES-DE-SIGNOS-VITALES-Monitor-de-Paciente-Multiparametros-C50,1583,53,0.htm.

3. García Valdés , Mario J. Desarrollo de Monitores de Paciente en Cuba. La Habana : s.n.

4. Buenas Tareas. [En línea] 2015. http://www.buenastareas.com/materias/signos-vitales-segun-oms/0.

5. Docencia Enfermeria. [En línea] 15 de mayo de 2010. http://docenciaenenfermeria.blogspot.com.

6. Medicinenet.Com. [En línea] 24 de noviembre de 2015. http://www.medicinenet.com/script/main/art.asp?articlekey=3674.

7. Definicion de ritmo cardiaco. [En línea] 2007-2015. www.definicionabc.com.

8. Fadlallah Sulbarán, Rosa Emilia. Medicina preventiva santa fe. [En línea] 30 de septiembre de 2014. http://medicinapreventiva.info/generalidades/3881/sabes-que-son-los-signos-vitales-y-para-que-sirven-drafadlallah/.

9. MedlinePlus. *Presión arterial alta.* [En línea] 2 de junio de 2016. https://www.nlm.nih.gov/medlineplus/spanish/highbloodpressure.html.

10. Signos vitales (temperatura corporal, pulso, frecuencia respiratoria y presión arterial). [En línea] 2016. https://www.urmc.rochester.edu/Encyclopedia/Content.aspx?ContentTypeID=85&ContentID=P03963.

11. Leonard, Roberto F. *Aprendizaje Clínico Temprano.* 2007.

12. The free diccionario medical . [En línea] http://medical-dictionary.thefreedictionary.com/.

13. Ma. Joaristi, Jose. *Diferencias entre GSM, GPRS, Bluetooth y Wi-Fi.* México D.F : s.n.

14. Isaac PE. Análisis comparativo de las placas Arduino. [En línea] 29 de Julio de 2014 . http://comohacer.eu/analisis-comparativo-placas-arduino-oficiales-compatibles/.

15. Arduino. [En línea] 2016. http://playground.arduino.cc/Es/OSW041.

16. Ronquillo Ordoñez, Juan Carlos y Salgado Espinoza, Pamela Alexandra. *Diseño y construcción de un oxímetro de pulso portatil.* Cuenca : s.n., 2013.

17. Ayuda Electrónica. [En línea] 27 de 5 de 2009. http://ayudaelectronica.com/que-es-un-termistor/.

18. Electrónicos CALDAS. [En línea] http://www.electronicoscaldas.com/modulos-rf/482-modulo-bluetooth-hc-06.html.

19. Gosling, James . Java creator James Gosling: 'Google totally slimed Sun'. [En línea] 30 de abril de 2012. [Citado el: 18 de febrero de 2013.] http://www.cnet.com/news/java-creator-james-gosling-google-totally-slimed-sun/.

20. Java. *¿Qué es la tecnología Java y para qué la necesito? .* [En línea] [Citado el: 18 de febrero de 2016.] https://www.java.com/es/download/faq/whatis_java.xml.

21. Vacas, José Antonio. Academia Android. [En línea] 11 de diciembre de 2014. http://academiaandroid.com.

22. Developers. [En línea] https://developer.android.com/index.html.

23. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 2000.

24. Cuervo Callejas, Mauro y Baquero Moreno , Oscar Yobany. *Herramientas libres para modelar software.* Tunja : s.n., 2005.

25. Visual Paradigm. [En línea] http://www.visual-paradigm.com..

26. *Métricas en el desarrollo de software.*

27. Rodríguez Sánchez, Tamara . *Metodología de desarrollo para la Actividad productiva de la UCI.*

28. Sommerville, Ian. *Ingeniería de software Séptima edición.* Madrid : Pearson Educación, 2005.

29. Villamizar Suaza, Katerine. *Definición de equivalencias entre historias de usuario y especificaciones en UN-LENCEP para el desarrollo ágil de software .* Medellin; Colombia : s.n., 2013.

30. Badillo Astudillo, Guillermo E. . *Arquitectura de Software.* 2014.

31. Grosso, Andres. Patrones GRASP. [En línea] 21 de marzo de 2011. http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/.

32. Visconti, Marcello y Astudillo, Hernán. *Fundamentos de Ingeniería de Software.*

33. Slide Share. [En línea] 20 de Septiembre de 2011. http://es.slideshare.net/NAHAMA19/fase-de-implementacin-de-sistemas-de-informacin.

34. A Mañas, José. Prueba de Programas. [En línea] 16 de Marzo de 1994. http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/testing.htm#s22..

35. Basterra, y otros. *Android OS Documentation.* 2016.

36. Rubén Dario. Mantenimiento de computadores. [En línea] 20 de marzo de 2014. http://rdsoporteymantenimientodepc.blogspot.com/2014/03/metodologias-de-desarrollo-agiles-vs.html.