Universidad de Cienfuegos

Facultad de Ingeniería

Carrera Ingeniería Informática

Guía para medir la Sostenibilidad de un producto de software.

Trabajo de diploma para optar por el título de Ingeniero Informático

Autor:

José Javier Galindo Sánchez

Tutores:

MsC. Dailyn Sosa López

Cienfuegos, Cuba

Curso 2021 – 2022

Dedicatoria

A mi familia y mis amigos que me apoyaron en todo momento.

Resumen

Se tiene un software sostenible cuando su desarrollo y uso tienen un impacto mínimo negativo o positivo en el ámbito ecológico, económico y social. En la actualidad existen un gran número de normas y métricas que ayudan a medir la sostenibilidad de un producto software y, por tanto, su impacto en el medio ambiente, ya sea directa o indirectamente. El objetivo de esta investigación es proporcionar una guía para medir el grado de sostenibilidad que va a tener un software, antes de su implementación. Actualmente no se considera un punto importante la sostenibilidad a la hora de desarrollar un software por parte de los estudiantes, a pesar de que es un elemento muy importante dado que cada día se deteriora más el medio ambiente. Esta guía servirá para incentivar a los desarrolladores con propuestas de sostenibilidad.

Palabras claves: software sostenible, guía, desarrollo de software.

Summary

There is sustainable software when its development and use have a minimal negative or positive impact in the ecological, economic and social spheres. There are currently a large number of standards and metrics that help measure the sustainability of a software product and, therefore, its impact on the environment, either directly or indirectly. The objective of this research is to provide a guide to measure the degree of sustainability that a software will have, before its implementation. Currently, sustainability is not considered an important point when developing software by students, despite the fact that it is a very important element given that the environment deteriorates more every day. This guide will serve to encourage developers with sustainability proposals.

Keywords: sustainable software, guide, software development.

Índice

Introducción

El desarrollo de las Tecnologías de la Información y Comunicación (TIC) se ha dado de forma exponencial en los últimos 50 años debido a la inversión destinada a su investigación y desarrollo. Este desarrollo tecnológico ha traído consigo muchas ventajas que no han pasado desapercibidas para la humanidad. En la Cumbre Mundial de la Sociedad de la Información (CMSI) se colocaron a las TIC en un primer plano de importancia, definiéndolas como las principales herramientas generadoras de la información necesaria para generar el conocimiento que traerá consigo el desarrollo del hombre, haciendo que los esfuerzos económicos y políticos empiecen a apuntar hacia la expansión de las TIC.

A día de hoy, el término cubre cualquier producto que almacene, recupere, manipule, transmita o reciba información electrónicamente en forma digital (como pueden ser computadoras personales, televisión digital, teléfonos inteligentes o robots).

La humanidad atraviesa por graves problemas en torno a la situación ambiental, caracterizado por el continuo crecimiento de la producción, el irracional consumo de bienes y servicios, y la desigual distribución de riquezas y conocimientos, elementos que contribuyen sustancialmente al total y progresivo agotamiento de los recursos naturales planetarios y a su contaminación, además de crear desigualdades alarmantes entre seres humanos y naciones.

De este problema surge la necesidad de un desarrollo sostenible tanto de hardware como software, definido como el desarrollo que es capaz de satisfacer las necesidades actuales sin comprometer los recursos y posibilidades de las futuras generaciones [1]. Se puede decir entonces que el desarrollo sostenible busca garantizar el equilibrio entre el crecimiento económico, el cuidado del medio ambiente y el bienestar social en el presente y el futuro.

El término de “desarrollo sostenible” fue utilizado por primera vez en un documento público en 1980, en la Estrategia Mundial de Conservación, pero no es hasta 1987 que la expresión gana popularidad, en un Reporte de la Comisión

Mundial sobre Medioambiente y Desarrollo, denominado “Nuestro Futuro Común” [1].

Las investigaciones relacionadas a la sostenibilidad y desarrollo sostenible han ido en crecimiento desde su surgimiento en 1987 hasta la actualidad (2022), pues este no es un concepto estático, ha ido evolucionando paralelamente al desarrollo científico tecnológico y humano, admitiendo múltiples interpretaciones. La sostenibilidad tiene que entenderse como una disciplina articulada del conocimiento y como una nueva manera de repensar la relación de los hombres con la naturaleza, a partir de la integralidad de las dimensiones económicas, sociales, ambientales y de valores, que conlleve a una revolución global de supervivencia con el planeta [2].

Investigadores internacionales han dedicado sus esfuerzos a este tema, H. Komiyama [3] y M. Yarime [4] consideran que es toda una ciencia, ya que en su evolución apareció la necesidad de su formulación como la ciencia de la sostenibilidad la cual es integrativa, interdisciplinaria y transdisciplinaria. También coinciden en que es imprescindible la adopción de patrones racionales para la producción, el consumo y la distribución de todo tipo de bienes y servicios y una responsabilidad por parte de los gobiernos, materializado en proyectos sostenibles dirigidos a la conservación de los recursos económicos, sociales y ambientales.

Modificar actitudes y prácticas personales, entre ellas, nuestra forma de producir y consumir es una premisa del desarrollo sostenible. En la actualidad se reclama que el modelo de desarrollo de un país debe ser sostenible, lo que significa que sea compatible con los recursos disponibles y con la conservación del medio ambiente.

La formación curricular del ingeniero informático no puede desconocer que la tecnología es un fenómeno social que forma parte de la cultura. Si la tecnología es un hecho cultural, su práctica requiere de transformación social en la elección de alternativas para dar respuesta a las necesidades de desarrollo en un contexto dado.

Las TIC han traído consigo innumerables beneficios y con el pasar del tiempo han ayudado a mejorar y a facilitar la vida de las personas. Sin embargo, un factor poco considerado es la contaminación que causan los residuos tecnológicos cuando terminan su ciclo de vida útil. Los aparatos eléctricos y electrónicos (AEE) son aquellos que necesitan corriente eléctrica para poder funcionar, y pasan a ser residuos de aparatos eléctricos y electrónicos (RAEE) cuando han terminado su ciclo de vida útil o cuando su poseedor los considera obsoletos. Estos RAEE representan un peligro para el medio ambiente pues están compuestos por cientos de materiales tanto valiosos como potencialmente peligrosos, que pueden poner en peligro la salud de las personas y el medio ambiente si no se manejan de manera adecuada.

En los últimos años, se ha incrementado notablemente el interés de la comunidad científica internacional por el estudio de la sostenibilidad de los sistemas que componen la sociedad, debido en gran medida, a que cada vez sé es más consciente de la necesidad de un desarrollo que no comprometa la existencia de las futuras generaciones. Las organizaciones no se han quedado al margen de esta tendencia; y desde el 2013 se ha incrementado el número de instrumentos de reportes y evaluación de la sostenibilidad, especialmente en Europa, Asia y Latinoamérica [5]; además la mayoría de las más grandes compañías a nivel mundial integran datos financieros y no financieros en sus balances anuales como parte de sus estrategias para el logro de la sostenibilidad. Este incremento también se ha manifestado en los organismos internacionales dirigidos a promover el desarrollo sostenible en las empresas, tales como: los Negocios por la Responsabilidad Social, en inglés Business for Social Responsibility (BSR), el Consejo Empresarial Mundial para el Desarrollo Sostenible, en inglés World Business Council for Sustainable Development (WBCSD), el Pacto Mundial de las Naciones Unidas y la Carta de la Tierra de la Haya, por sólo citar las más importantes.

En el año 2015 sale a la luz un manifiesto que recoge un conjunto de principios y compromisos vinculados con el diseño sostenible. Sus suscriptores se refieren a que: la sostenibilidad tiene múltiples dimensiones (social, medioambiental, económica, técnica y humana) y que todas hay que incluirlas en el análisis; la sostenibilidad es sistémica y nunca es una propiedad aislada, la sostenibilidad

trasciende a múltiples disciplinas por lo que las personas de diferentes disciplinas deben trabajar juntas. La sostenibilidad se aplica al sistema y al contexto más amplio al que forma parte, la visibilidad del sistema es una condición previa necesaria y disponible para el plan de sostenibilidad; la sostenibilidad requiere de la acción de múltiples niveles. La sostenibilidad requiere de pensamiento a largo plazo y es posible satisfacer las necesidades de generaciones futuras sin sacrificar a la generación actual.

El término de computación verde, se refiere a un modelo que incluye el estudio y la práctica de las tecnologías de la información relacionadas con la arquitectura hardware, software, redes, sistemas y procesos, con el fin de gobernar la huella de carbono del medio ambiente mediante el uso de los recursos informáticos de manera eficaz y eficiente con un mínimo de efectos negativos para el medio ambiente y sin comprometer la productividad económica y la responsabilidad social a largo plazo. En la actualidad se utilizan dos términos diferentes para abarcar la multiplicidad de aristas asociadas con la computación y la sostenibilidad: Green IT (Green InformationTechonlogies o Tecnologías de la Información Verdes) y Green IS (Green Information Systems o Sistemas de Información Verdes).

Las tecnologías de la informática y comunicación (TIC) son una pieza fundamental y pueden convertirse en aliadas en la lucha contra el cambio climático y sus impactos negativos, para el logro del desarrollo sostenible a través las tecnologías verdes, Green IT, Green Computing, referidas al uso eficiente de los recursos computacionales, minimizando el impacto ambiental, maximizando su viabilidad económica tal como lo propugna el documento final de Río + 20.

En el futuro la sociedad en general y la industria, como gestora de gran parte de las actividades humanas relacionadas con el ambiente y las materias primas, además de los parámetros de calidad, economía y productividad, deberá obtener su beneficio o sus menores costes teniendo en consideración que sus efectos sobre el medio natural y sobre la población sean mínimos o nulos.

El software sostenible es aquel cuyo desarrollo, despliegue y uso tiene un impacto mínimo, directo o indirecto, negativo o incluso positivo en la economía,

sociedad, seres humanos y medio ambiente. Se puede decir que la Ingeniería del Software Sostenible es el arte de desarrollar software sostenible a través de procesos de ingeniería del software sostenible.

El software sostenible debe cumplir 3 condiciones:

1. Debe ser producido de forma que cumpla los objetivos de sostenibilidad.
2. Durante su uso debe tener un mínimo impacto negativo social y ambiental.
3. Sus funcionalidades deben reforzar el desarrollo sostenible o al menos no tener impactos negativos en la sociedad o el medio ambiente.

Se considera que los aspectos de la sostenibilidad en la ingeniería de software son: el proceso de desarrollo de software con el uso responsable de recursos ecológicos, humanos y financieros, el proceso de mantenimiento del software hasta que se reemplace por uno nuevo, la producción del sistema y el uso del sistema.

Teniendo en cuenta lo planteado anteriormente se identifica como **problema a resolver**: ¿Cómo medir la sostenibilidad de un software antes de su desarrollo?

Se presenta como **objeto de estudio**: el desarrollo de software sostenible y como **campo de acción**: indicadores para la medición de la sostenibilidad de un software.

Se plantea como **idea a defender** que: la propuesta de un método para medir la sostenibilidad del software antes de desarrollarse va a contribuir a las buenas prácticas de sostenibilidad y la toma de decisiones de clientes de software.

Se define como **objetivo general** de la investigación: proponer un método para medición de sostenibilidad de un software antes de su desarrollo.

La **justificación de esta investigación** está dada por la necesidad de que los estudiantes tengan una guía para tener en cuenta la sostenibilidad a la hora de desarrollar un software, acorde con las exigencias ambientales actuales.

Como métodos de investigación se emplearon fundamentalmente los siguientes:

De los Métodos Teóricos se utilizaron:

Método Histórico Lógico: Se utiliza con el objetivo de profundizar en los antecedentes de las teorías correspondientes al desarrollo del software sostenible, considerando los elementos más significativos de este proceso y revelando las características y métricas.

Método Análisis- Síntesis: Se utilizará para captar y resumir varios documentos y procedimientos por los cuales se rige el desarrollo del software sostenible, de ellos se extraerán las ideas fundamentales y al mismo tiempo se detallará la información necesaria para la propuesta de la guía.

La tesis está estructurada, en Resumen, Introducción, 3 capítulos, Conclusiones, Recomendaciones y Bibliografía.

Capítulo 1: Fundamentos teóricos del desarrollo de software sostenible.

Se hace un análisis de los antecedentes científicos en Cuba y el mundo sobre los principales conceptos asociados al desarrollo sostenible enfocándose en el desarrollo de software. También se realiza una revisión de los principales teóricos del tema e indicadores.

Capítulo 2: Análisis de métricas existentes para medir la sostenibilidad de un software.

Se realiza un análisis de las diferentes métricas existentes para medir el grado de sostenibilidad de un software.

Capítulo 3: Aplicación de la guía.

Para comprobar la validez de la propuesta se decide aplicar la misma a softwares, que, aunque ya fueron desarrollados, nos permiten comprobar si su desarrollo está en correspondencia con las métricas de sostenibilidad de sistemas informáticos. Se seleccionan 4 tesis desarrollas en el departamento.

1 - Fundamentos teóricos del desarrollo de software sostenible

1.1 - Introducción

En este capítulo se hace un análisis de los antecedentes científicos en Cuba y el mundo sobre los principales conceptos asociados al desarrollo sostenible enfocándose en el desarrollo de software. También se realiza una revisión de los principales teóricos del tema e indicadores de sostenibilidad del software.

1.2 - Principales conceptos asociados al dominio del problema

1.2.1 - Sostenibilidad

El origen del concepto sostenibilidad se sitúa a principios de la década de los años 80, a partir de perspectivas científicas sobre la relación entre el medioambiente y la sociedad y la publicación de varios documentos relevantes, principalmente la Estrategia Mundial para la Conservación [6] y el conocido como Informe Brundtland [1].

En el Informe Brundtland, que sería el primer intento de introducir el concepto de Sostenibilidad se describe como: "El Desarrollo Sostenible es el desarrollo que satisface las necesidades de la generación presente sin comprometer la capacidad de las generaciones futuras para satisfacer sus propias necesidades".

Surge como resultado de los análisis de la situación del mundo, que puede describirse como una “emergencia planetaria” y de larga duración como una

situación insostenible, fruto de las actividades humanas, que amenaza gravemente el futuro (y ya el presente) de la humanidad. Según Bybee, se trata de "la idea central unificadora más necesaria en este momento de la historia de la humanidad" [7].

Luego del informe Brundtland se alcanza un consenso sobre el desarrollo de una consciencia sobre los límites del planeta y la importancia de trabajar de una manera sostenible. El concepto de sostenibilidad es aparentemente confuso y está lleno de contradicciones, con la indefinición de lo que se debe o no sostener, o con las listas de necesidades que se han de satisfacer.

Norton [8] plantea dos tipos de sostenibilidad: sostenibilidad débil y sostenibilidad fuerte. La sostenibilidad débil se ubica dentro de la economía estándar, mientras que la segunda esta vincula a la termodinámica y a la ecología.

Sostenibilidad débil: La sostenibilidad débil es un concepto genérico que puede definirse como «la viabilidad de un sistema socioeconómico en el tiempo». Esta viabilidad se consigue manteniendo el capital global (las capacidades en términos del informe Brundtland), generación tras generación, siendo este capital global el resultado de otros dos: el capital natural y el capital de formación humana. El capital de formación humana hace referencia a la disponibilidad de capital monetario, tecnología, personal formado, etc.

Sostenibilidad fuerte: Es definida como la viabilidad de la relación que mantiene un sistema socioeconómico con un ecosistema [9]. En esta definición, el énfasis se pone en la interacción entre estos dos sistemas dinámicos, teniendo en cuenta que el sistema socioeconómico es dependiente del ecosistema en el sentido de que éste podría funcionar autónomamente. La interacción consiste en una permanente co-adaptación.

Mientras las sociedades se abastecen de los recursos naturales y expulsan sus desechos, los ecosistemas sufren cambios y se reajustan; a menudo estos cambios provocan modificaciones tecnológicas, económicas y sociales.

En el año 2015 en un manifiesto publicado, se acopian los principales principios y compromisos de la sostenibilidad, estos son: [10]

- la sostenibilidad tiene múltiples dimensiones (social, medioambiental, económica, técnica y humana) y que todas deben incluirse en el análisis
- la sostenibilidad se aplica al sistema y al contexto más amplio al que forma parte
- la visibilidad del sistema es una condición previa necesaria y disponible para el plan de sostenibilidad
- la sostenibilidad requiere de la acción de múltiples niveles
- la sostenibilidad requiere de pensamiento a largo plazo y es posible satisfacer las necesidades de generaciones futuras sin sacrificar a la generación actual.
- la sostenibilidad es sistémica y nunca es una propiedad aislada.
- la sostenibilidad trasciende a múltiples disciplinas por lo que las personas de diferentes disciplinas deben trabajar juntas.

1.2.2 - Desarrollo sostenible

El desarrollo sostenible es un término asociado a la posibilidad de lograr que una región crezca a partir de la explotación de sus recursos, sin que dicha explotación ponga en riesgo la existencia futura de los recursos en cuestión. El desarrollo sostenible también contempla que el crecimiento se consiga sin injerencia del exterior [11]. Si un determinado país se desarrolla bajo la sobreexplotación de sus recursos naturales no renovables, tarde o temprano, el crecimiento será interrumpido, por eso el desarrollo sostenible se logra sin perjudicar al medio ambiente ni en el presente ni en el futuro.

El desarrollo sostenible como concepto, es una alternativa al concepto de desarrollo habitual o social, que pretende una homogeneidad y coherencia entre

el crecimiento económico de la población en todos sus estratos, los recursos naturales y la sociedad.

El ámbito del desarrollo sustentable puede dividirse conceptualmente en tres partes: ecológico, económico y social. Se considera el aspecto social por la relación entre el bienestar social con el medio ambiente y la bonanza económica. El triple resultado es un conjunto de indicadores de desempeño de una organización en las tres áreas [12].

1.2.3 - Software sostenible

Debido al creciente uso de los servicios de Informatización y Comunicación, se ha incrementado la demanda de energía, y esta demanda trae consigo un impacto negativo en el medio ambiente. Las emisiones globales de dióxido de carbono anuales han alcanzado recientemente los 9.100 millones de toneladas, donde al menos el 2% de las mismas pueden ser atribuidas a los sistemas de TIC [13].

Muchos de los estudios y regulaciones se centran en mediciones, análisis y control del consumo de energía asociado con el hardware, sin considerar el software. El software no consume energía directamente, pero sí afecta la utilización del hardware, esto se puede traducir en un consumo de energía indirecto. El dominio de investigación en ingeniería de software ha comenzado a prestar atención a la sustentabilidad, tal como lo demuestra el creciente número de publicaciones y estudios empíricos [14]. De la misma forma que las TIC sostenible, el software sostenible busca reducir el impacto ambiental del software. Las empresas de software han comenzado a confrontar la necesidad de ser amigables con el ambiente y las necesidades de los clientes por requerimientos cada vez mayores.

Este impacto del software en el consumo de energía no es precisamente despreciable, especialmente si pensamos en los millones de dispositivos y de

veces que se ejecuta cierto código. Las tecnologías de la información “verdes” (Green IT) tienen como objetivo procurar el uso de recursos hardware de forma sostenible para minimizar el impacto medioambiental y alargar la vida útil de las baterías de los dispositivos inteligentes. Esto significa que las metodologías de desarrollo de software actuales deben revisarse para centrarse en producir software más sostenible. En este contexto el rol del Ingeniero del Software es fundamental, ya que debe ser consciente de las implicaciones que cada decisión de diseño e implementación tienen en el consumo de energía del sistema final [15].

Muchas son las empresas que se han comprometido por completo al desarrollo del software verde. Como parte de su compromiso, Globant, una institución global dedicada a crear mejores prácticas para desarrollar software sostenible y así reducir las emisiones de carbono, lidera otras iniciativas en el ámbito de la sostenibilidad. En el frente de cero emisiones, la compañía empezó su ruta hacia las emisiones Net Zero en 2020, pasó a operar con recursos de energía 100% renovable y está comprometida a ser completamente neutra en carbono para finales de este año [16].

En el campo de la ingeniería de software no hay un consenso sobre la definición de sostenibilidad. Cada investigador tiene su comprensión de lo que significa este concepto. Se puede considerar que el software es sostenible cuando los impactos negativos, directos e indirectos en la economía, la sociedad, los seres humanos y el medio ambiente que resultan del desarrollo, implementación y uso del software son mínimos y/o tienen un efecto positivo en el desarrollo sostenible. [14].

Para caracterizar la sostenibilidad de software se han utilizado los modelos de calidad de los programas informáticos estándar, como la norma ISO/IEC 25010. Hay varios trabajos que proponen diferentes características de sostenibilidad como la fiabilidad, la mantenibilidad, la portabilidad y la modificabilidad, el rendimiento, la usabilidad, la interoperabilidad y la adaptabilidad. La evaluación de cada uno de estos parámetros o métricas puede darnos una idea del grado de sostenibilidad del software, estas se miden en función de atributos de calidad,

al mejorar estas métricas aportan mayores beneficios a las otras dimensiones como la económica, social y ambiental [17].

1.3 - Estudios existentes sobre modelos de procesos de desarrollo de software sostenible

- En 2010, Markus Dick y Stefan Naumann, en su publicación “Mejorar los procesos de Ingeniería de Software para el diseño de un producto de software sostenible” proponen una extensión genérica para procesos de desarrollo de software cuya meta es una integración entre cuestiones de sustentabilidad en procesos de desarrollo de software arbitrarios y guiar a un mejor producto de software sostenible. Las herramientas, listas de chequeo y líneas guías de asistencia profesional, con la aplicación de su modelo, se enfocan en el pilar medioambiental de la sostenibilidad y el consumo de energía [18].

- En 2012 Lami define un grupo de procesos de alto nivel que añaden los conceptos de sostenibilidad a los procesos de referencia de software definidos en ISO/IEC 12207. Los procesos de referencia estándar describen todas actividades directamente e indirectamente relacionadas con el desarrollo, mantenimiento y operación del software. Además, señala que los procesos respecto a la sostenibilidad están faltantes en la ISO/IEC12207, por lo tanto, define un grupo de tres procesos de sostenibilidad complementarios: el Proceso de Dirección Sostenible, el Proceso de Ingeniería Sostenible y el Proceso de Capacitación Sostenible [19].

- En 2013 la Universidad Politécnica de Cataluña, presenta un estudio realizado para incluir los conceptos de sostenibilidad en un proyecto de ingeniería, en la guía del Trabajo de Final de Grado (TFG) de Ingeniería Informática de la Facultad de Informática de Barcelona, donde siguen la idea de establecer una serie de preguntas que inviten al estudiante a

reflexionar sobre la forma de afrontar su trabajo. Sin embargo, las ideas y métodos discutidos sirven en general para cualquier proyecto TIC y, por extensión y con ciertas modificaciones, para cualquier ingeniería [11].

- En el 2018, en el departamento de Ingeniería Informática de la Universidad de Cienfuegos, se presenta la tesis de Deisy González Bilbao, titulada “Propuesta de un Modelo de Procesos como base para la formación de informáticos del territorio hacia un desarrollo de software sostenible”. En la misma se realizó un estudio de los aspectos teóricos sobre el desarrollo sostenible, profundizando en el desarrollo de software sostenible y los modelos de procesos de ingeniería de software sostenible. Como resultado, basándose en la propuesta el Modelo de Proceso para una Ingeniería de Software Ágil y Sostenible y adaptándose al contexto cubano considerando las características de las empresas del territorio, se propone un sistema de conocimientos como base para la formación de los ingenieros informáticos del territorio hacia el desarrollo de un software sostenible [20].

- En 2018 Achim Guldner, Marcel Garling, Marlies Morgen y Stefan Naumann, publican un artículo en el cual presentan un método para medir el consumo de energía que el software induce al hardware, aplicándolo a dos grupos de software: los procesadores de texto (WP) y sistemas de gestión de contenido (CMS) [21]. Aunque los dos grupos son muy diferentes en relación a sus requisitos, fueron exitosos en la creación de un ambiente de medición que soporta la producción de un resultado confiable y verificable, admitiendo la comparación del consumo de energía producido por sistemas de software con una funcionalidad similar. El método muestra los resultados viables para sistemas de escritorio y servidores, preparando el camino para configuraciones adicionales como por ejemplo dispositivos movibles e implantados.
  El artículo se concentra en la medición de la utilización de equipo físico y presentara un método de medición para el consumo de electricidad. Para asegurar la equivalencia entre los diferentes tipos de productos de

software, WP y CMS, se define un guion de uso estándar para cada categoría [21].

Hay algunos desafíos involucrados en la creación de tal guion. Primero se analiza las tareas que los usuarios generalmente realizan con el software y se consideran las funciones del software y módulos que son de uso frecuente. También se estudian los potenciales de optimización dentro del software. Por lo tanto, se tiene cuidado con las funcionalidades que pueden causar mayor consumo de energía en el hardware [21].

Finalmente, se debe conseguir un guion que sea tan realista como posible, considerando el comportamiento del usuario. Para esto graban a varios usuarios llevando a cabo un escenario, por ejemplo, para un CMS, usando otros medios como una herramienta generadora de carga que llama los sitios web generados por el CMS muchas veces por minuto, simulando muchos usuarios. Se llevan las mediciones verdaderas del consumo de energía y la utilización del hardware a un test que se crearon después que la ISO/IEC 14756.

En este, el producto de software es instalado sobre un sistema bajo prueba (SUT), que consiste en el hardware (computadora de escritorio, servidor, smartphone, etcétera), sistema operativo y si fuera necesario, promover ambientes de tiempo de ejecución. El SUT no sólo opera el producto de software sino también graba su utilización durante las mediciones. Adicionalmente, se usa un medidor de vatios industrial con una frecuencia de muestreo de 20 kHz por fase y resoluciones de 0.01 V y 1 mA, respectivamente, para medir el consumo de energía del SUT. Los datos recolectados son promediados por segundo y luego guardados y analizados en un agregador y evaluador central de datos.

Otro factor a considerar es la sobrecarga sobre el SUT, resultando de grabar la utilización del hardware. En las mediciones, se encontró que la sobrecarga de las herramientas de grabación de datos es insignificante. Sin embargo, si las mediciones más precisas son necesarias, podría ser aconsejable medir el software de grabación primero y luego restar este punto de partida de la medición verdadera. Cuando la configuración está completa, usamos un generador de carga para ejecutar los guiones

automática y repetidamente sobre el SUT, para asegurar unos resultados estadísticamente robustos.

A continuación, se describe el método para analizar la medición de los resultados de la evaluación del consumo de energía medio y la utilización media del hardware de los productos de software mientras efectuamos el guion de uso estándar.

El consumo de energía es calculado como un valor aritmético medio sobre cada medición. La energía para cada muestra de medición es calculada como la suma sobre el wataje medio por segundo (Pi) que fue medido para el SUT. Se divide el resultado entre 3600 s/h para recibir el resultado por el indicador en watt-horas (Wh). Por lo tanto, la fórmula para calcular el indicador para el consumo de energía es: [21]

$$E = \frac{1}{3600 \cdot n} \sum_{k=1}^{n} \sum_{i=1}^{m} p_i,$$

Donde la duración de k-th es m segundos y ahí está un total de n mediciones.

La utilización del procesador es calculada como el valor de media aritmética sobre cada muestra de medición en por ciento (%) del rendimiento de procesador máximo del SUT. Esto quiere decir que, si hay un total de n mediciones, calculamos el indicador para la utilización de procesador como:

$$U_P = \frac{1}{n} \sum_{k=1}^{n} \overline{x_k},$$

Donde $\overline{x_k}$ es la media aritmética de la muestra de k - th en %. Se calculan los indicadores para otra calidad de hardware como la utilización de RAM en una manera análoga.

Para el grupo de procesadores de texto, se seleccionaron dos procesadores de texto generalizados para compararlos, WP1 y WP2. La decisión de usar estos productos ha sido tomada por el hecho de que son ampliamente usados y uno de ellos es un producto de software comercial mientras que otro es de código abierto. Se implementaron las acciones como editar el contenido y el diseño del

documento, además de salvarlo, varias veces durante el guion. Se usó el documento ¨Anatomía quirúrgica¨ de José Maclise, el cual contiene más de 300 páginas. Al final se crearon dos documentos pdf idénticos, uno con cada procesador de texto.

Para tomar una muestra representativa los dos productos fueron instalados secuencialmente en el SUT e iniciados en la configuración estándar. Adicionalmente, prestamos atención a una visualización similar de la aplicación que comprende, por ejemplo, abrir el software en la pantalla completa, ajustar el nivel de desplazamiento de foco y activar los caracteres de control.

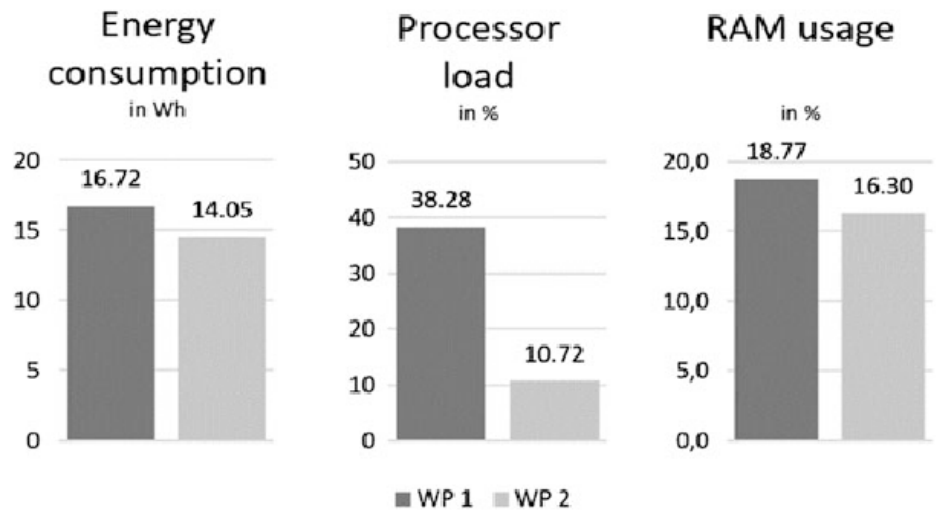

Figura 1: Resultados de las mediciones de los procesadores de texto WP1 y WP2 [21].

Al comparar WP1 con WP2 se hayan diferencias significativas en cuanto a consumo de energía, carga del procesamiento y uso de la RAM (Fig. 1). Ambas aplicaciones tenían un máximo apogeo en el principio mientras ponían en marcha el software, abrían el documento y cambiaban el diseño del contenido [21].

Además de eso, las mediciones de WP2 indican los máximos apogeos mientras guardaba el documento. Las otras acciones tienen solamente influencias pequeñas sobre el nivel general del consumo de energía. Sin embargo, WP1

presentaba los giros mucho más grandes mientras llevar a cabo las acciones de todas clases. Eso es parte de la razón por la cual también se analiza la transición al modo de reposo para ambos productos de software.

Para este guion, se abre un nuevo documento sin ningún contenido y se da una mirada en la reacción de software. WP1 espera 26 s hasta que el cursor deja de destellar. Después, el consumo de energía y utilización de equipo físico disminuyen a un nivel insustancial de la lectura. WP2 se queda en modo activo por 18 s sin la entrada del usuario antes de que cambie al modo de reposo y la utilización del hardware y el consumo de energía disminuyen a un mínimo.

Los resultados indican que hay potencial de optimización para WP1. Especialmente para el cursor destellando. Además, en WP2 la mayor parte del editar, como añadir el contenido o cambiar el diseño del documento, tenía que un inferior impacto en el consumo de energía en WP1.

Para los grupos de sistemas de contenidos, se escogieron dos CMS (CMS1 y CMS2) debido a su larga participación en el mercado. Para poder llevar a cabo las mediciones necesarias con el CMS, se crea el contenido en forma de un sitio web. Para desempeñar la equivalencia de los resultados, este contenido tiene que ser igual para todos productos. La demanda para la igualdad cubre el contenido y la estructura, tanto como el diseño del sitio web. Se cumplen estos requisitos desarrollando una plantilla básica, implementarlo para cada CMS y luego compilar la misma estructura y el contenido en cada uno. En relación con el tamaño, el sitio web representa un sitio de información pequeño, comprimible en texto e imágenes, como esas a menudo operadas por pequeñas empresas.

El guion de uso sobre la base del sitio web cubre las típicas interacciones usuario de un editor que trabaja en el panel de administrador. Después de que el usuario entra al sistema, algunos comentarios son dirigidos, entonces una nueva página es creada, editada, y publicada. El usuario carga archivos de PDF y los vincula. Después, son retirados otra vez con la herramienta de revisión incorporado. Definitivamente, todos cambios son revocados, permitiendo que la próxima corrida de medición empiece con las condiciones idénticas.

Se crea este guion para correr sobre un cliente, que sirve de generador de carga, mientras que el CMS fue instalado en el servidor (SUT). Se miden ambos

sistemas simultáneamente durante las corridas de prueba. El procedimiento es aplicado repetidamente para medir el impacto de diferentes niveles de tensión: en la primera configuración, solamente el usuario autómata del guion de uso ejecuta las interacciones con el sistema. En una segunda configuración, sin embargo, se usa la herramienta de generador de carga mencionada anteriormente para llevar a cabo 14,000 solicitudes del sitio web adicionales por minuto, simulando algunos clientes de computadoras automáticamente.

Es extraordinario que ejecutar el guion que usa CMS1, el servidor envía por promedio más datos que cuando corre CMS2 (CMS1: 0.09 megabytes/ s; CMS2: 0.03 megabytes/ s). Aunque la actividad de la red en el lado del cliente es más alta para CMS1, CMS2 es el que muestra la actividad de escribir en el disco duro (CMS1: 0.08 megabytes/ s; CMS2: 0.13 megabytes/ s).

Los resultados de la segunda configuración (una con la herramienta de generador de carga en uso) brinda una nueva percepción mucho más diferenciada. El consumo de energía y la utilización de hardware son ahora elevadas evidentemente. Comparando los dos sistemas bajo tensión, CMS2 resulta ser más absorbente que CMS1 en muchos aspectos. Tiene un consumo de energía más alto, un uso de CPU más alto y ahora también una actividad de transferencia de datos más alta. (Fig. 2)

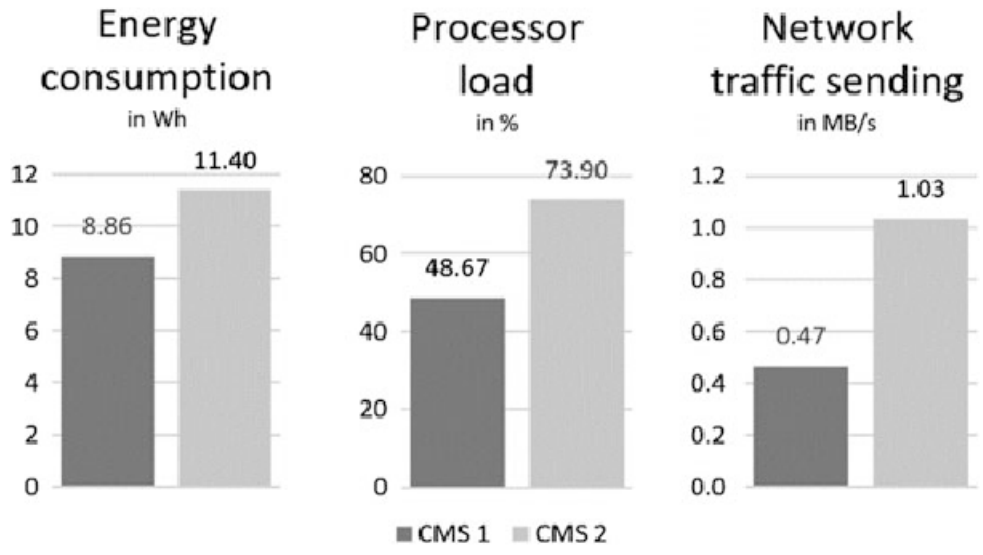

Figura 2: Resultados de las mediciones de CMS1 y CMS2 [21].

Sobre la base de estos resultados, los desarrolladores pueden mejorar su código, y los compradores de software pueden tomar las decisiones, sobre la base de la sostenibilidad de los productos [21].

1.3.1 - Modelo McCall

Este modelo fue creado por Jim McCall en 1977. Establece 3 perspectivas para el análisis de la calidad de software, define 11 factores y 23 criterios relacionados a estos. Las métricas que propone son preguntas que ponderan numéricamente un determinado atributo del producto de software. Después de obtener los valores para todas las métricas de un criterio específico, el promedio de todas ellas es el valor para ese criterio. En la Fig. 3, se presentan los criterios de calidad asociados a los factores de calidad en el modelo de McCall [22].

Perspectivas	Factores	Criterios
Operatividad del Producto: factores de calidad que influyen en el grado en que el software cumple con su especificación.	Usabilidad: La facilidad de uso del software.	Operatividad Entrenamiento Comunicación
	Integridad: La protección de programa del acceso no autorizado.	Control de Acceso Auditoría de Acceso
	Corrección: El grado en que una funcionalidad coincide con su especificación.	Rastreabilidad Completitud Consistencia
	Fiabilidad – confiabilidad: La capacidad de los sistemas de no fallar / la medida en que falla el sistema.	Consistencia Exactitud Tolerancia a fallos
	Eficiencia: Además clasificado en la eficiencia de la ejecución y la eficiencia de almacenamiento y por lo general significa que el uso de los recursos del sistema, ejemplo: tiempo de procesador, memoria.	Eficiencia en Ejecución Eficiencia en Almacenamiento
Revisión del Producto: factores de calidad que influyen en la capacidad de cambiar el producto de software.	Mantenibilidad: Esfuerzo requerido para localizar y arreglar un fallo en el programa dentro de su entorno operativo.	Simplicidad Concreción
	Facilidad de Prueba: La facilidad del programa de realizar pruebas para asegurarse de que está libre de errores y cumple con su especificación.	Simplicidad Instrumentación Auto-descripción Modularidad
	Flexibilidad: La facilidad de hacer los cambios necesarios según lo solicitado en el entorno operativo	Auto-descripción Capacidad de expansión Generalidad Modularidad
Transición del Producto: factores de calidad que influyen en la capacidad de adaptar el software a los nuevos entornos.	Reusabilidad: La facilidad de reutilización de software en un contexto diferente.	Auto descripción Generalidad Modularidad
	Interoperabilidad: El esfuerzo requerido para acoplar el sistema a otro sistema.	Modularidad Similitud de comunicación Similitud de datos Independencia del sistema Independencia de la máquina
	Portabilidad: El esfuerzo requerido para transferir un programa desde un entorno a otro.	Auto-descripción Independencia del sistema Independencia de la máquina

Fig. 3 – Diagrama de McCall – Criterios asociados a factores de calidad [22].

Si bien el modelo de McCall es uno de los primeros desarrollados, la mayoría de los factores definidos conservan su vigencia en la actualidad, y muchos otros modelos de calidad desarrollados y adaptados posteriormente se basen en él, incluso la Norma ISO 9126 es una estandarización de este modelo [22].

Para emplear el modelo de McCall hay que seguir las siguientes pautas: [23]

1. Se aceptan los factores, criterios y métricas que propone el modelo.
2. Se aceptan las relaciones entre factores y criterios, y entre criterios y métricas.
3. Se selecciona un subconjunto de factores de calidad sobre los que aplicar los requisitos de calidad establecidos para el proyecto.

Al comienzo del proyecto habrá que especificar los requisitos de calidad del producto software, para lo cual se seleccionarán los aspectos inherentes a la calidad deseada del producto, teniendo que considerarse para ello: [23]

- Las características particulares del propio producto que se está diseñando: por ejemplo, su ciclo de vida que si se espera que sea largo implicará un mayor énfasis en la facilidad de mantenimiento y la flexibilidad, o bien si el sistema en desarrollo está destinado a un entorno donde el hardware evoluciona rápidamente implicará como requisito su portabilidad, …
- La relación calidad-precio, que puede evaluarse a través del coste de cada factor de calidad frente al beneficio que proporciona. La siguiente tabla muestra la relación calidad-precio para cada factor considerado:

Factor	Beneficio / coste
Corrección	alto
Fiabilidad	alto
Eficiencia	bajo
Integridad	bajo
Facilidad de uso	medio
Facilidad de mantenimiento	alto
Facilidad de prueba	alto
Flexibilidad	medio
Portabilidad	medio
Reusabilidad	medio
Interoperabilidad	bajo

Tabla 1: Relación beneficio/coste de los factores de calidad [23].

- La determinación de las etapas del ciclo de vida donde es necesario evaluar cada factor de calidad para conocer en cuales se dejan sentir más los efectos de una calidad pobre con respecto a cada uno de los factores.
- Las propias interrelaciones entre los factores debido a que algunos factores pueden entrar en conflicto entre sí: por ejemplo, la eficiencia plantea conflictos prácticamente con todos los demás factores de calidad.

La interacción entre los diversos factores a evaluar queda reflejada en la tabla 1 que indica la dependencia entre los factores de McCall.

- También habrá que establecer valores deseables para los criterios, para lo cual se emplearán datos históricos, el promedio en la industria, …. y con ellos se concretarán los valores finales y otros intermedios o predictivos en cada período de medición durante el desarrollo, así como unos valores mínimos aceptables. La explicación para cualquier selección o decisión deberá ser adecuadamente documentada.

En la fase de desarrollo será necesario implementar las métricas elegidas, analizar sus resultados y tomar medidas correctivas cuando los valores obtenidos estén por debajo de los mínimos aceptables.

Una vez finalizado el proyecto será necesario contrastar las medidas predictivas utilizadas y comprobar si, en efecto, se pueden tomar como indicadores de los valores finales.

1.3.2 - Modelo FURPS

Este modelo fue desarrollado por Hewlett-Packard en el año 1987. En él se desarrollan un conjunto de factores de calidad de software, bajo el acrónimo de FURPS: funcionalidad (Functionality), usabilidad (Usability), confiabilidad (Reliability), desempeño (Performance) y capacidad de soporte (Supportability). En la Fig. 4 se muestra el diagrama de FURPS y los criterios de calidad y factores asociados [22].

Factores	Criterios
Funcionalidad	Características y capacidades del programa Generalidad de las funciones Seguridad del Sistema
Usabilidad	Factores Humanos Factores Estéticos Consistencia de la interfaz Documentación
Confiabilidad	Frecuencia y severidad de fallos Exactitud de las salidas Tiempo medio de fallos Capacidad de recuperación ante fallos Capacidad de predicción
Rendimiento	Velocidad de procesamiento Tiempo de respuesta Consumo de recursos Rendimiento efectivo total Eficacia
Capacidad de Soporte	Extensibilidad Adaptabilidad Capacidad de Prueba Capacidad de configuración Compatibilidad Requisitos de instalación

Fig. 4 – Diagrama de FURPS – Criterios asociados a factores de calidad [22].

Este modelo tiene como ventaja que los criterios son claramente entendibles, esto implica su fácil utilización. Tiene en cuenta las fallas en el producto y en el proceso esto permite una mayor corrección. Se podría utilizar no para una sino para varios proyectos. Como inconveniente está que al igual que el modelo McCall, se necesita de muestras métricas lo que implica un mayor esfuerzo de tiempo y costo.

1.3.3 - Modelo BOEHM

Este modelo propone una jerarquía de niveles, en forma de un árbol con tres ramas principales, que permiten que el software sea de utilidad: Portabilidad, Facilidad de Uso y Facilidad de Mantenimiento. Se estructura en tres niveles:

Aplicaciones primarias, Construcciones Intermedias (factores) y Construcciones Primitivas, y finalmente las Métricas que determinan los valores para los criterios (construcciones primitivas). El diagrama de Boehm se presenta en la Fig. 5 [22].

Factores	Criterios
Portabilidad	Independencia dispositivos Completitud
Fiabilidad	Completitud Exactitud Consistencia
Eficiencia	Eficiencia dispositivo Accesibilidad
Ingeniería humana	Accesibilidad Comunicatividad Estructuración Auto-descripción
Comprensibilidad	Consistencia Estructuración Auto-descripción Concisión Legibilidad Expansibilidad
Modificabilidad	Estructuración

Fig. 5– Diagrama de Boehm – Criterios asociados a factores de calidad [22].

Boehm, ideó y promulgó un modelo desde un enfoque distinto al tradicional en Cascada: El Modelo Evolutivo Espiral. Su Modelo de Ciclo de Vida en Espiral tiene en cuenta fuertemente el riesgo que aparece a la hora de desarrollar software. Para ello, se comienza mirando las posibles alternativas de desarrollo, se opta por la de riesgo más asumible y se hace un ciclo de la espiral. Si el cliente quiere seguir haciendo mejoras en el software, se vuelve a evaluar las distintas nuevas alternativas y riesgos y se realiza otra vuelta de la espiral, así hasta que llegue un momento en el que el producto software desarrollado sea aceptado y no necesite seguir mejorándose con otro nuevo ciclo [24].

1. Es un modelo de proceso de software evolutivo donde se conjuga la naturaleza de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal y secuencial. Proporciona el potencial para el

desarrollo rápido de versiones incrementales del software que no se basa en fases claramente definidas y separadas para crear un sistema. El modelo en espiral se divide en un número de actividades de marco de trabajo, también llamadas regiones de tareas, cada una de las regiones están compuestas por un conjunto de tareas del trabajo llamado conjunto de tareas que se adaptan a las características del proyecto que va a emprenderse en todos los casos se aplican actividades de protección [24].

2. El modelo espiral tuvo varias modificaciones que son:

• Modelo Original de Boehm. Modelo Típico de Seis Regiones.

• Modelo WINWIN. Modelo original de Boehm. No hay un número definido de iteraciones.

Las iteraciones debe decidirlas el equipo de gestión de proyecto. Cada vuelta se divide en 4 sectores:

• Planeación: determinación de los objetivos, alternativas y restricciones

• Análisis de riesgo: análisis de alternativas e identificación/resolución de riesgos
• Ingeniería: desarrollo del producto hasta "el siguiente nivel".

• Evaluación: valoración por parte del cliente de los resultados obtenidos.

El movimiento de la espiral, ampliando con cada iteración su amplitud radial, indica que cada vez se van construyendo versiones sucesivas del software, cada vez más completas. Uno de los puntos más interesantes del modelo, es la introducción al proceso de desarrollo a las actividades de análisis de los riesgos asociados al desarrollo y a la evaluación por parte del cliente de los resultados del software.

• Modelo típico de seis regiones:

A diferencia del modelo de proceso clásico que termina cuando se entrega el software, el modelo en espiral puede adaptarse y aplicarse a lo largo de la vida del software de computadora. Una visión alternativa del modelo en espiral puede ser considerada examinando el eje de punto de entrada en el proyecto. Las regiones de tareas que componen este modelo son: [24]

• Comunicación con el cliente: las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.

• Planificación: las tareas requeridas para definir recursos, el tiempo y otras informaciones relacionadas con el proyecto. Son todos los requerimientos.

• Análisis de riesgos: las tareas requeridas para evaluar riesgos técnicos y otras informaciones relacionadas con el proyecto.

• Ingeniería: las tareas requeridas para construir una o más representaciones de la aplicación.

1.3.4 - ISO/IEC 9126

El Estándar Internacional (ISO), aplicable a todo tipo de software, está basado en un modelo jerárquico con tres niveles: Características, Subcaracterísticas y Métricas. En el primer nivel tiene seis características principales: Funcionalidad, Fiabilidad, Eficiencia, Facilidad de Mantenimiento, Portabilidad y Facilidad de Uso. Estas características (factores) están compuestas a su vez por 27 subcaracterísticas (subfactores) relacionadas con la calidad externa, y 21 subcaracterísticas relacionadas con la calidad interna, en la Fig. 6 se presentan los factores y criterios asociados al modelo [22].

Factores	Criterios
Funcionalidad	Adaptabilidad Exactitud Interoperabilidad Seguridad
Usabilidad	Comprensibilidad Aprendizaje Operatividad Atractivo
Mantenibilidad	Análisis Cambio Estabilidad Prueba
Fiabilidad	Madurez Tolerancia a fallos Recuperabilidad
Eficiencia	Comportamiento del tiempo Uso de los recursos
Portabilidad	Adaptabilidad Instalación Coexistencia Reemplazo

Fig. 6 – Diagrama de ISO/IEC 9126 – Criterios asociados a factores de calidad [22].

Estas normas no solo son requisitos que se deben cumplir para estar alineado con los requerimientos de la industria, sino que son herramientas que ofrecen importantes beneficios para los desarrolladores y el cliente final. En este sentido, garantizar todas las premisas basadas en las normativas internacionales y nacionales del desarrollo de productos digitales puede hacer parte de las prácticas de aseguramiento de la calidad del software [22].

1.3.5 - Modelo de Calidad para Software Sostenible de Naumann

El Modelo de Calidad para el Software Sostenible, en la Fig. 7, muestra los criterios para determinar si un software es sostenible o no. En general, los diferentes aspectos pueden ser agrupados en los criterios comunes, los criterios

directamente relacionados, y los criterios indirectamente relacionados y métricas, como se describe en el modelo [14].

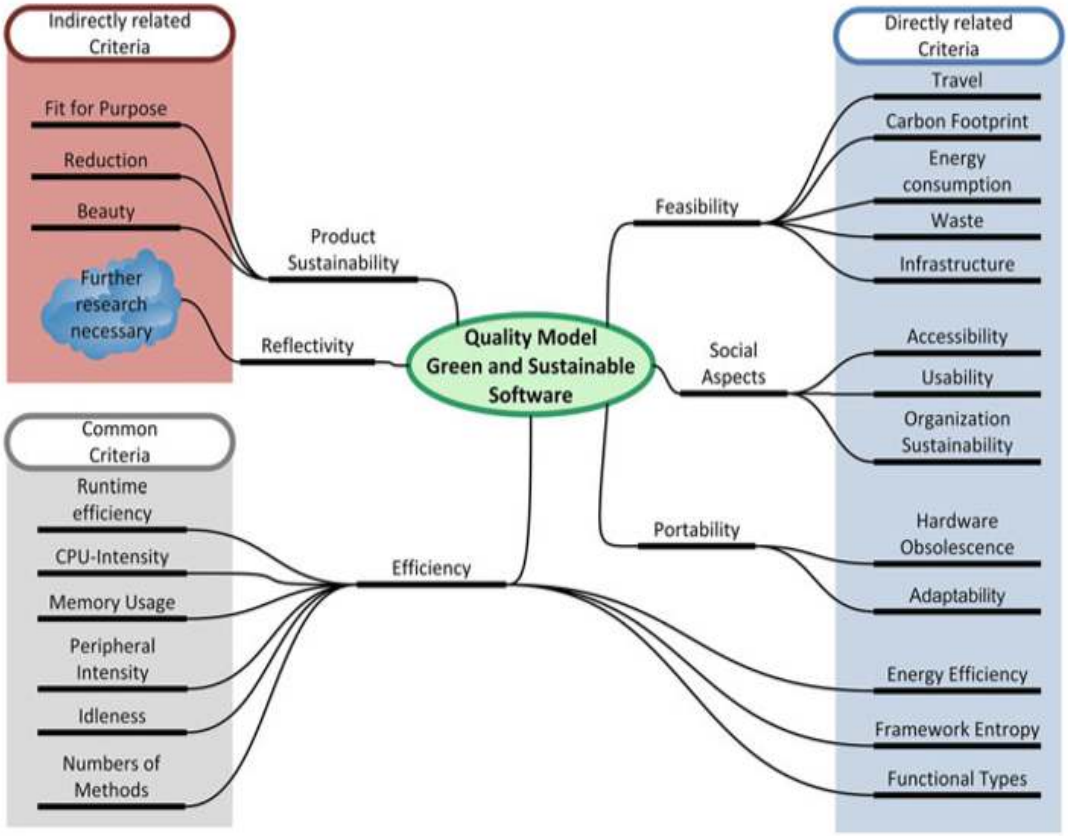

Fig. 7 - Modelo de Calidad para Software Sostenible [14].

Criterios comunes de calidad (*Common Quality Criteria*): Los criterios comunes surgen de los conocidos y estandarizados aspectos de calidad de software, emitidos por la Organización Nacional de Estandarización. El modelo toma aspectos como: Eficiencia, Reusabilidad, Escalabilidad y Usabilidad. Los siguientes indicadores pertenecen al criterio Eficiencia: [14]

- Eficiencia en el tiempo de corrida (*Runtime Efficiency*): se considera el tiempo necesario para terminar la ejecución del software dependiendo de su implementación.
- Intensidad de la CPU (*CPU Intensity*), Uso de la Memoria (*Memory Usage*) e Intensidad de los periféricos (*Peripheral Intensity*): Comprenden los recursos utilizados por la ejecución del software.

- Inactividad (*Idleness*): Describe cuan a menudo el sistema está inactivo (usualmente utilizado en los servidores).
- Número de métodos (*Number of Methods*): Expresa el tamaño de la aplicación.

Criterios Directamente Relacionados (*Directly Related Criteria*): En contraste con los criterios comunes de eficiencia, la eficiencia energética del sistema en si puede ser evaluado en un sistema de referencia. Esto significa que su meta es optimizar el consumo de energía en relación con los servicios del sistema. Por tanto, el indicador Eficiencia Energética (*Energy Efficiency*), que pertenece al criterio Eficiencia, es asignado a los Criterios Directamente Relacionados, junto con la Entropía del Framework (*Framework Entropy*) y Tipos Funcionales (Functional *Types*) [14].

- Entropía del Framework (*Framework Entropy*): representa el grado de uso de librerías externas y peticiones de alto nivel.
- Tipos Funcionales (*Functional Types*): Indican los diferentes tipos de aplicaciones con sus respectivos tipos de niveles de eficiencia energética.

Obsolescencia del Hardware: considera la cantidad de hardware que necesita ser reemplazado antes de que se termine la sustentabilidad de la vida útil, considerando los altos requerimientos del sistema. Este indicador, junto con Adaptabilidad, pertenece al criterio Portabilidad. Adaptabilidad describe las posibilidades del software de adaptarse a las necesidades específicas del consumidor [14].

El criterio de calidad Factibilidad (*Feasibility*) evalúa los efectos del desarrollo del producto, o más específicamente el proceso de ingeniería de software en el desarrollo sostenible. En este contexto se tiene: [14]

- Viajes (*Travel*): Incluye viajes de negocios que corresponden al desarrollo del software o incluso los viajes al centro de trabajo diariamente.
- Huella de carbón (*Carbon footprint*): especifica la cantidad de CO2 emitido a causa del desarrollo del software en el curso del ciclo de vida de este.
- Consumición de energía (*Energy Consumption*): indica la cantidad de energía consumida a causa del desarrollo del software.

- Desperdicios (*Waste*): Cuenta los recursos consumidos durante el desarrollo del software por actividades que no tienen valor adicional para consumidores o usuarios finales.
- Infraestructura (*Infrastructure*): condiciones necesarias para que el software pueda ser utilizado.

Desde la perspectiva de los Aspectos Sociales del desarrollo sostenible se tiene los siguientes indicadores: [14]

- Accesibilidad (*Accessibility*): Indica si el software puede ser usado por la mayor cantidad de personas posible sin restricciones.
- Usabilidad (*Usability*): Cubre la entendibilidad, operabilidad y facilidad de aprendizaje del software por parte de los usuarios.
- Sostenibilidad de la organización (*Organization Sustainability*): Cubre la situación social de la compañía.

Criterios Indirectamente Relacionados (*Indirectly Related Criteria*): Describe como por el uso del software, el uso de recursos y energía pueden ser reducido en otros sectores. Por eso, los criterios indirectamente relacionados con el producto de software deben ser también considerados. [14]

El criterio Sostenibilidad del Producto (*Product Sustainability*) hace un sumario de los efectos del software en otros productos o servicios.

- Ajusta al propósito (*Fit for Purpose*): considera que como se ajusta el software al propósito para el que fue desarrollado.
- Reducción (*Reduction*): Es la contribución del software dentro de su dominio de aplicación para reducir emisiones, desperdicios.
- Belleza (*Beauty*): Contribución del software al desarrollo sostenible.
- Reflectividad (*Reflectivity*): se refiere a la calidad como un aspecto de eficiencia. Especifica la manera en la que el uso del software influye en los recursos que son usados indirectamente por otras personas o sistemas. Este aspecto no pertenece a los Efectos de Primer Orden en el sentido propio del término, pero cuenta como un efecto del uso e indica un estándar de calidad de los Criterios Indirectamente Relacionados. Sin

embargo, se requiere más investigación en este campo con el objetivo de considerar el comportamiento del usuario y los escenarios de uso [14].

1.3.6 - Modelo GQM O Goal Question Metric

Goal Question Metric busca formas por medio de las cuales se puedan definir las métricas para medir los avances que ocurren aplicando preguntas que tengan que ver con el tema del proyecto [25].

En este modelo se presentan tres niveles:

- Nivel Conceptual:

aquí se define un objetivo para un objeto, se tienen en cuenta varios puntos de vista con relación a un entorno particular.

- Nivel Operacional: se utilizan varias preguntas que permitan definir

los modelos del objeto de estudio, con el fin de establecer cuál es el logro de un objetivo específico.

- Nivel Cuantitativo: Un grupo de métricas se asocian con cada una de las preguntas para responderlas utilizando medidas.

En el modelo GQM encontramos objetivos comerciales que se utilizan en la identificación de las mediciones correctas. También se realizan otros pasos para almacenar los datos de medición; luego esta información servirá para tomar decisiones y realizar mejoras. Se enfoca a proporcionar una forma que permita definir métricas para medir el avance como los resultados de algún proyecto, a partir de la aplicación de unas preguntas relacionadas con el proyecto, que permitan alcanzar unas metas previamente planteadas, el modelo trabaja sobre metas, preguntas y métricas [25].

En Cuba se aplica el modelo GQM en contextos académicos, en donde se ha identificado una pequeña brecha entre los resultados de la evaluación de

software y la comprensión del equipo de desarrollo, de esta manera en la aplicación de las tres etapas del modelo en conjunto con UML se describe la estructura a partir del proceso de pruebas, teniendo en cuenta la arquitectura y el comportamiento de los datos en el momento de ser probados. Entre las experiencias de aplicación de GQM en el campo empresarial se puede mencionar el caso de la implantación del modelo en un core bancario, a partir de la definición de métricas, alineadas con los objetivos y metas del negocio, se orienta al mejoramiento en el proceso del desarrollo de software, conducente a la obtención de un producto de calidad para la empresa. En la Tabla 2, se presenta un listado de empresas que realizaron la implementación de GQM [25].

Empresa	**Área**	**País**
TERABANK	Desarrollo de software bancario	Georgia
Universidades europeas	Competencias en ingeniería académica	Europa
Reservado	Evaluación de plataforma Joomla	Argentina
Universidad de Ciencias Informáticas	Modelo de desarrollo de software	Cuba

Tabla 2 - Implementación del modelo GQM [25].

1.3.7 - Modelo de calidad GILB

Tom Gilb nació en Pasadena, en 1940. Este modelo presenta como aspecto fundamental la definición de los atributos de la calidad que realmente interesan

al usuario y el nivel de calidad que debe tener cada uno de ellos para satisfacerlo [25].

La estructura del modelo Gilb pertenece a un tipo fijo el cual se compone de 4 dimensiones de calidad [26].

· **Capacidad de trabajo**. Evalúa la capacidad natural del sistema para realizar su trabajo.

O subatributos: Capacidad del proceso, capacidad de respuesta, capacidad de almacenamiento.

· **Disponibilidad**. Refleja la medida de la disponibilidad del sistema para realizar de forma útil el trabajo para el que fue diseñado.

O subatributos. Fiabilidad, Mantenibilidad e integridad

· **Adaptabilidad.** Es la medida de la capacidad de un sistema para ser modificado de manera adecuada.

O subatributos: Improbabilidad, extensibilidad y transportabilidad.

· **Usabilidad**. Es la medida de la facilidad con que la gente será capaz y estará motivada para utilizar el sistema en la práctica.

O subatributos: Requisitos de entrada, requisitos de aprendizaje y habilidad de manejo.

De acuerdo a las características del modelo Gilb puede presentar las siguientes métricas donde también podemos observar sus definiciones: [26]

Métricas externas: son aquellas que miden el comportamiento de todo software que parte de él, a través de testeos, operaciones y observaciones del software

ejecutable en el sistema. Proporcionando a todos los involucrados el beneficio de conocer la calidad del producto software durante las pruebas.

Métricas internas: Las métricas internas pueden ser aplicadas durante el diseño y la codificación del producto software no ejecutable (por ejemplo, código fuente) y proporciona a todos los involucrados el beneficio de conocer la calidad del producto durante su construcción y tomar decisiones cobre esa base para conseguir el producto con la calidad esperada.

Métricas de uso: Mide como un producto cumple con las necesidades de los usuarios para alcanzar sus objetivos. La evaluación de la calidad en el uso valida la calidad del producto software en escenarios específicos de uso.

Este modelo es que propone características como la corrección, la integridad, la facilidad de mantenimiento y la facilidad de uso, como base para proporcionar indicadores útiles para los equipos de trabajo [26].

1.4 - Conclusiones

Durante los últimos años, las investigaciones científicas sobre la sostenibilidad han tomado auge. Han surgido a lo largo de la historia estos modelos que tienen como objetivo mejorar la sostenibilidad del software y a su vez, ayudar a la reducción de la huella ecológica que dejan este tipo de productos.

2 - Análisis de métricas existentes para medir la sostenibilidad de un software

2.1 - Introducción

En este capitulo se realiza un análisis de las diferentes métricas existentes para medir el grado de sostenibilidad de un software. También se propone un grupo de métricas para ser usadas antes de desarrollar el software y tener un estimado del grado de sostenibilidad que va a presentar el mismo.

2.2 - Concepto de métricas

Se define Métricas de Software como la forma eficaz de proporcionar evidencia empírica que puede mejorar la comprensión de las diferentes dimensiones de la complejidad del software [27]. Estas métricas son aplicadas constantemente durante todo el proceso de desarrollo del software y también en el producto obtenido y brindan información importante que permite a los desarrolladores mejorar los procesos y productos.

Las métricas e índices sostenibles son medidas de sostenibilidad e intentan cuantificar más allá del concepto genérico. Aunque existen desacuerdos entre personas de diferentes disciplinas (e influenciadas por diferentes creencias políticas sobre la naturaleza de la buena sociedad), estas disciplinas y organizaciones internacionales han ofrecido medidas o indicadores de cómo medir el concepto. Para poder medir el grado en que un sistema o proceso posee un atributo dado se definen y utilizan estas métricas [28].

Las Métricas son propias de cada modelo de calidad y se crean para medir los criterios definidos en él. El proceso de recopilación de las métricas de software se realiza mediante la obtención de datos de los procesos de Ingeniería de software y de los proyectos y productos de software. Estas medidas son utilizadas para poder llevar a cabo el cálculo de las métricas, de cuya aplicación

y posterior evaluación se plantean los indicadores para las mismas. Las Métricas pueden ser directas o indirectas, las primeras son aquellas que no necesitan ningún otro atributo o entidad, mientras que las segundas se forman por la combinación de una o más métricas directas [29]. Por ejemplo, métricas directas serian LOC: líneas de código fuente escritas o HPD: horas-programador diarias; y métricas indirectas HPT: horas-programador totales (Σ HPD) o LCFH: líneas de código fuente por hora de programador (LOC/HPT).

2.2 - Análisis de métricas existentes

Existen un gran número de métricas que pueden ser utilizadas en la medición de la sostenibilidad de un software, adaptándolas a las propias necesidades del equipo que está en envuelto en el desarrollo. Algunas otras métricas que pueden ser utilizadas para medir la sostenibilidad de un software son:

- **Consumo total de energía**: es la energía total consumida para desarrollar el software. Esta es típicamente la energía eléctrica extraída de la red de servicios públicos, pero también incluiría cualquier producción de energía en el sitio a partir de generadores, solares o eólicos. También se debe contar la energía importada en forma de gas natural, vapor o agua refrigerada. En muchos casos, una parte significativa de las emisiones de carbono proviene del consumo de energía. Comprender el consumo total de energía es necesario para realizar un seguimiento de la mejora de la eficiencia y reducir la mezcla de carbono en el suministro [30].
- **Adecuación de suministro y consumo hora por hora**: esta métrica mide la medida en que la generación de energía renovable coincide con el consumo de energía dentro de una organización. Gigantes de Internet como Microsoft y Google todavía están probando este concepto. Esto puede proporcionar un mayor nivel de transparencia sobre cómo la producción de energía renovable coincide con el consumo en tiempo real. El objetivo es alcanzar el 100% de la producción y el consumo renovables hora a hora [30].

- **Cantidad de código**: los equipos de desarrollo pueden mirar esta métrica de software, también llamada miles de líneas de código (KLOC), para determinar el tamaño de una aplicación. Si este valor es alto, podría indicar que los desarrolladores fueron productivos en sus esfuerzos de programación. Sin embargo, esta métrica no es útil cuando un equipo de desarrollo intenta comparar dos proyectos escritos en diferentes lenguajes de programación. Además, tenga en cuenta que más código no siempre hace que el código sea eficiente o efectivo, lo que puede significar más trabajo de refactorización más adelante [31].
- **Tiempo de entrega**: es el tiempo de entrega es el tiempo que tarda algo de principio a fin. En el desarrollo de software, por ejemplo, el tiempo de entrega de un proyecto comienza con la propuesta y termina con la entrega [31].
- **Mantenibilidad**: se define la como el grado de efectividad o eficiencia con la que un producto o sistema puede ser modificado. Es el esfuerzo requerido para localizar y arreglar un fallo en el programa dentro de su entorno operativo. Estudios previos han demostrado que una de las causas de los problemas en el mantenimiento del software es que a menudo, la mantenibilidad no es una consideración importante durante las etapas de diseño e implementación de software. Realizar más esfuerzo durante el ciclo de vida de desarrollo para hacer que el software sea mantenible puede reducir significativamente el total de los costos del software. Factores que afectan la mantenibilidad son: la documentación del software, simplicidad en el código, capacidad de expansión, lenguaje de programación usado [32].
- **Facilidad de mantenimiento:** es el esfuerzo requerido para localizar y reparar errores. Cuando se desarrolla un software, la tarea no termina cuando se instala y pone en funcionamiento el software en las instalaciones del cliente. Un producto software envuelve muchos aspectos y características que provocan que sea totalmente necesario supervisar su funcionamiento correcto durante un tiempo después de la entrega del mismo. Los factores que influyen en la mejora de la mantenibilidad son una estructura del software fácil de comprender, facilidad de uso del

sistema, emplear lenguajes de programación y sistemas operativos estandarizados y crear una estructura estandarizada de la documentación [31].

- **Portabilidad**: se define como la característica que posee un software para ejecutarse en diferentes plataformas, el código fuente del software es capaz de reutilizarse en vez de crearse un nuevo código cuando el software pasa de una plataforma a otra. A mayor portabilidad menor es la dependencia del software con respecto a la plataforma. La capacidad del software para ser trasladado de un entorno a otro. Este es el núcleo del concepto de portabilidad [33].
- **Facilidad de uso**: la facilidad de uso es una medida de lo fácil que es usar un producto para realizar tareas prescritas. Un factor crítico en el desarrollo de software es el diseño de la interfaz entre el usuario y la máquina. De esta manera, un programa o un sitio web deben estar bien estructurados, que el uso sea intuitivo y sea fácil de navegar para el usuario [33].
- **Fiabilidad**: es la capacidad de un sistema para desempeñar sus funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados [33].
- **Uso total de agua:** es el consumo total de agua en el lugar donde se está desarrollando el software. El uso del agua es el valor neto que cubre la extracción, evaporación y descarga de agua. Incluye el uso de agua dulce y agua regenerada. El agua regenerada se puede utilizar para torres de enfriamiento de centros de datos para ahorrar agua potable/dulce. Predecir el uso de agua en la fase de diseño dará como resultado una tecnología de enfriamiento mejorada que reducirá el uso de agua en sitio [30].
- **Reusabilidad**: se puede definir como una medida de la facilidad con la que pueden utilizarse conceptos o elementos software en nuevas situaciones [33].
- **Eficiencia**: se refiere a la capacidad del producto de software para proporcionar un desempeño apropiado, en relación con la cantidad de recurso utilizado, bajo condiciones establecidas en determinado momento

del tiempo. Para determinar qué tan eficiente es un producto es necesario tener en cuenta el comportamiento en el tiempo, consumo de recursos y conformidad en la eficiencia. Se puede decir que un producto software es eficiente si: [30]

- La utilización de recursos del sistema es adecuada, estos pueden incluir otros productos de software con los cuales debe interactuar la aplicación en un momento determinado, la configuración del software y hardware necesaria para el sistema, materiales requeridos.
- El desempeño del software se considera que es el esperado, de acuerdo con los niveles de servicio pactados para el mismo, teniendo en cuenta las necesidades de los usuarios.

- **Huella de carbono**: la Huella de Carbono de software es una ecoetiqueta identificativa de la totalidad de gases de efecto invernadero (GEI) emitidos por efecto directo o indirecto de este producto. Las empresas que están interesadas en corregir el impacto ambiental negativo de sus acciones miden su huella de carbono con el afán de poder tomar acciones e implementar procesos que contribuyan a mitigar los efectos adversos. El cálculo de la huella de carbono se obtiene a partir de la multiplicación de la cifra de consumo de energía (dato de actividad) por el factor de emisión. Es el parámetro que define el nivel de la actividad que generan las emisiones de gases de efecto invernadero. Por ejemplo: el volumen de gas natural que se utiliza para la calefacción (medido en kWh). El factor de emisión es la cantidad de gases de efecto invernadero emitido por cada unidad del dato de actividad [34]. La industria del software busca contribuir con un futuro más sostenible. Por eso la institución Green Software Foundation (GSF) lanzó la versión alfa de Software Carbon Intensity (SCI) Specification, un mecanismo de puntuación para sistemas de software sobre la base de las emisiones de carbono que producen. “Como compañía nativa digital, creemos que es fundamental reconocer y mitigar el impacto de la tecnología en el medioambiente, así como reforzar de manera constante nuestro compromiso para hacer que el desarrollo de

software sea más ecológico y sostenible, tanto en el caso de nuestros propios productos como los de los clientes [34]. Contribuimos a los objetivos sostenibles globales mejorando la sustentabilidad de nuestra cadena de valor tecnológica y la de nuestros clientes", explicaron en un comunicado de prensa miembros de Green Software Foundation. El mecanismo de Software Carbon Intensity Specification (SCI) funciona como una guía para la reducción de la huella de carbono total del software. En lugar de crear una metodología para calcular la huella de carbono total de un sistema de software, "este mecanismo brinda información sobre cómo las empresas pueden reducir su propia huella. No se trata de una solución integral, sino que determina la tasa de emisiones de carbono de los diferentes sistemas de software y funciona como guía para garantizar que las empresas estén yendo por el camino correcto" [34].

Un grupo de expertos en Inteligencia Artificial (IA) ha desarrollado una innovadora herramienta de código abierto llamada CodeCarbon que puede calcular la huella de carbono que generan los sistemas informáticos en función de su ubicación. Se trata de un paquete de software ligero integrado en el código base de Python que es capaz de rastrear las emisiones de CO2 que producen los recursos informáticos para ejecutar código. "El rastreador registra la cantidad de energía que utiliza la infraestructura de los principales proveedores de nube y centros de datos locales", explican los creadores. "Basado en fuentes de datos disponibles públicamente, estima la cantidad de emisiones producidas teniendo en cuenta la red eléctrica a la que está conectado el hardware". El objetivo de *CodeCarbon* es incentivar a los programadores a optimizar la eficiencia de su trabajo y también les aconseja sobre cómo pueden reducir las emisiones seleccionando infraestructuras en la nube en regiones que utilicen fuentes de energía con menor huella de carbono [35].

- **RAEE (Residuos de Aparatos Eléctricos y Electrónicos)**: los dispositivos, cuando dejan de ser funcionales, se consideran basura electrónica y en sí mismos no causan daño, sino hasta que se destruyen y/o incineran, al liberar una serie de residuos peligrosos que afectan la

salud de las personas y contaminan el medio ambiente. Se han propuesto medidas para que el efecto adverso de estos residuos sea menor o nula, algunas de estas son: [36]

- **Re-uso:** Extensión de la vida útil del equipo o sus componentes para ser usado para el mismo propósito para el cual fue conceptualizado inicialmente; puede o no incluir un cambio en la propiedad del equipo. Este proceso pretende promover un uso óptimo de los recursos disponibles, sin embargo, hay que tener cuidado con riesgos sociales o ambientales asociados a una mala gestión [36].
- **Desemsamblaje y segregación:** Consiste en la separación manual y cuidadosa de las partes y componentes de un equipo en desuso. Se sugiere que esta actividad se realice por recicladores autorizados, compañías especializadas en reacondicionamiento [36].
- **Reciclaje y recuperación:** Consiste en la recuperación del equipo, sus componentes y materiales. El desmantelamiento puede ser manual o semi-manual. La recuperación de materiales forma parte del proceso de reciclaje de RAEE sobre todo para la recuperación de metales que requiere instalaciones especializadas e inversiones importantes [36].
- **Reacondicionamiento:** Toda operación que permite que el AEE considerado como RAEE pueda funcionar nuevamente. Incluye operaciones de hardware y software [36].
- **Disposición final:** consiste en el proceso de eliminación final de residuos o materiales no recuperables bajo técnicas controladas por ejemplo rellenos sanitarios (vertederos) o incineración [36].

- **Desperdicio total generado:** es el peso total de los residuos de material generados. La medición debe comenzar desde la construcción y continuar hasta el final de la vida útil del software. La cantidad de residuos se puede clasificar por fase, por ejemplo, durante la construcción, pero también por marco de tiempo durante las operaciones regulares. Al igual que las emisiones de carbono, los residuos se pueden medir como residuos directos, pero también como

residuos indirectos. Esta métrica se puede utilizar para cuantificar los impactos relacionados con los residuos de la organización en el medio ambiente y el objetivo es minimizar los residuos generales generados. Los residuos directos deben ser el foco de los informes, y a medida que los informes mejoran en toda la industria, se puede agregar la generación indirecta de residuos para rastrear la cadena de suministro [30].

A pesar de que existe un sinnúmero de métricas para medir la sostenibilidad de un software, en esta investigación se propone tener en cuenta la mantenibilidad, la fiabilidad, la portabilidad, la facilidad de uso y el consumo total de energía. Se seleccionan estas métricas porque son medibles en el momento del desarrollo del software y no necesitan de un sistema en operación para ser analizadas, dando así una guía para que los desarrolladores se enfoquen en la sostenibilidad de su producto software.

2.3 - Guía propuesta para el desarrollo de software sostenible

Ya analizadas las anteriores métricas, se seleccionan la **mantenibilidad**, la **fiabilidad**, la **portabilidad**, la **facilidad de uso** y el **consumo total de energía**, ya que es importante tener estas métricas en cuenta antes de desarrollar el software. También se darán consejos para contribuir a reducir la emisión de gases de efecto invernadero y reducir la huella de carbono. Con esta guía el estudiante va a tener en sus manos un “medidor” de sostenibilidad para su producto.

La guía se centra en estas cinco métricas y propone un rango de clasificación para cada una:

- **Mantenibilidad:** la mayoría de los softwares con el tiempo es necesario hacer cambios en el código, ya sea por un cambio en la estructura de una

base de datos, cambios en el tipo o configuración de red o cualquier otro cambio que entre en conflicto con el programa. De ahí surge la necesidad de hacer cambios en este y modificar su código. Un buen trabajo en la etapa de desarrollo significaría un ahorro en costos de mantenimiento en el futuro.

  - **Clasificación:** un software con una documentación, un código simple y un lenguaje de programación que sea bastante usado y común (como Java, Python, C++), se puede clasificar esta métrica en **sostenible**. La documentación es esencial para poder comprender un código, por tanto, si no presenta documentación, a pesar de presentar un código simple o estar desarrollado en un lenguaje fácil de comprender, se clasifica como **poco sostenible**. Si no presenta ninguno de los parámetros anteriores, podemos decir que la métrica **no es sostenible**.

- **Consumo total de energía:** un producto software no consume energía, pero si condiciona el uso del procesador que si consume energía. También hay que tener presente que en el desarrollo del software se consume energía eléctrica, ya sea en iluminación o los mismos recursos para desarrollar el software. Esta energía eléctrica es la extraída de la red de servicios públicos, aunque también puede provenir de fuentes renovables. En la mayoría de los casos, una gran parte de las emisiones de carbono proviene del consumo de energía. Comprender el consumo total de energía es necesario para realizar un seguimiento de la mejora de la eficiencia y reducir la mezcla de carbono en el suministro.

  - **Clasificación:** un producto software puede ser ambicioso en sus requerimientos o ser un poco más simple, de la complejidad de sus funciones depende la complejidad de su código. Aunque estos aspectos sean proporcionales, siempre los programadores deben de tener en cuenta la optimización del código para así reducir el consumo energético por parte del procesador. Un software que este bien optimizado, a pesar de presentar un código complejo, va a ser un producto que va a consumir poca energía eléctrica. Implementar modos de bajo consumo cuando el programa no esté

en uso y tener en cuenta el consumo de energía en el momento de su desarrollo, ya sea apagando luces innecesarias o aprovechando la luz natural, son prácticas que harán que nuestro software sea **sostenible**. Si nada de esto se tiene en cuenta, tendremos un software que genera un consumo elevado de electricidad y hoy en día ninguna empresa quiere tener un producto que no sea amigable con el medio ambiente, ya que sería un software **no sostenible**.

- **Facilidad de uso:** en su mayoría, un software se desarrolla para una o un grupo de personas que no necesariamente tiene un amplio conocimiento de la informática. En la gran mayoría de los softwares, la forma de lograr la interacción del programa con el usuario que lo va a utilizar es mediante una interfaz gráfica. Se acostumbra en los proyectos realizados por grandes empresas que esta tarea del diseño sea de los diseñadores, pero en proyectos pequeños es necesario que el mismo desarrollador tenga en cuenta el diseño de la interfaz gráfica, por eso es conveniente planificar una que sea agradable a la vista e intuitiva a la hora de usar.
    - **Clasificación:** lo primero que se debe de tener en cuenta en cuanto a la facilidad de uso es la accesibilidad. Cuanto más accesible sea el software, más sencillo será de usar. Hay usuarios para los que la accesibilidad supone un desafío. Se debe tener en cuenta a los que tienen pérdida parcial o total de la audición, problemas visuales, problemas motores y discapacidades cognitivas. Ofrecer a los usuarios un sistema de búsqueda efectivo, que permita encontrar un comando, función o sección por su nombre parcial o un sinónimo de éste, hace mucho más accesible para personas que no son expertas. Un diseño minimalista es esencial a la hora de navegar por una aplicación o programa, muchos usuarios de tan solo ver un diseño plagado de opciones que en ocasiones son innecesarias se agobian y rechazan el software o ya tienen una mala impresión de este. Todas estas buenas prácticas a la hora de planificar como va a ser implementada la interfaz grafica conllevan a que tenga una

facilidad de uso **sostenible**. No tener en cuenta un diseño minimalista, colores agradables a la vista o no tener una buena organización son malas prácticas para el desarrollo, pero son esenciales a la hora de usar el producto, no cumplir con lo anterior conllevaría a un software **poco sostenible**. Algo que no puede suceder es no tener en cuenta la accesibilidad, por ejemplo, una persona con baja visión no puede leer texto con una fuente muy pequeña, para esta persona el software es inservible. Esta hace que el software no llegue a la totalidad de usuarios y por tanto, se considera **no sostenible**.

- **Portabilidad:** en el mercado existe una gran diversidad de hardware y de plataformas de software con distintas especificaciones, sistemas operativos e instrucciones. Esto es algo con que los desarrolladores han tenido que tener en cuenta con el paso del tiempo para que su producto software pueda ser ejecutado en diferentes plataformas y utilizando distinto hardware y no se quede obsoleto luego de que un usuario o empresa cambie de su hardware o software.
    - **Clasificación:** enfocarse en la portabilidad en la etapa del desarrollo es crucial para que más usuarios utilicen el software. Cada 4 años o menos, las organizaciones clientes compran un nuevo hardware, y todo su software a continuación, debe ser adaptado para funcionar con el nuevo hardware y la portabilidad facilita la migración a nuevas versiones de los sistemas y a nuevos entornos. Un producto software que se adapta en ambientes determinados sin realizar acciones o aplicar medidas, fácil de instalar y que pueda coexistir en el sistema sin tener conflictos con otro software, es lo que busca la portabilidad, haciendo de este un software **sostenible**. Un software que su instalación sea un problema para el usuario sería **poco sostenible**, ya que requeriría de un experto para poder instalarlo. Si al desarrollar el software no tenemos en cuenta todos los aspectos anteriores es probable que ese software, aproximadamente a los 4 años de su despliegue, ya se vuelva obsoleto por una actualización del sistema o el cambio del sistema operativo por parte del usuario o empresa, dándole un

ciclo de vida relativamente corto. Esto conllevaría a que el software sea **no sostenible**.

- **Fiabilidad:** la fiabilidad se basa en la capacidad que tiene el software para cumplir con todos los requisitos de cliente usándose en las condiciones y el periodo de tiempo determinado. Usar una aplicación que cumpla con todos los requisitos y que no presente fallas al momento de ser usada es la premisa de la fiabilidad. Siguiendo buenas prácticas se puede contribuir a mejorar la fiabilidad de nuestro producto software.
    - **Clasificación:** muchos proyectos fracasan o se prolongan en plazos por hacer estimaciones poco realistas. Una planificación razonable depende de fijar bien los tiempos, los recursos y los esfuerzos. Una buena elección garantizará una correcta utilización de los requisitos del sistema y menos problemas dados por limitaciones y restricciones del sistema en caso de haberlas. El uso de módulos pequeños que ya están probados y que se integren al proyecto actual es una buena práctica. Siguiendo buenas prácticas se puede lograr que el producto sea fiable, con menos errores y con un ciclo de vida amplio. Un software que se desarrolle empleando todas estas buenas practicas es un producto **sostenible**. Sin embargo, no utilizarlas puede resultar en que el software no cumpla con todos los requerimientos del cliente, sea propenso a errores o quede obsoleto en poco tiempo. Este escenario para una aplicación significaría más gastos de recursos y esfuerzos para reparar errores o actualizar módulos en el futuro y se debe considerar este producto como **no sostenible**.

2.3.1 - Propuestas para reducir la huella de carbono

La lucha contra el cambio climático es una prioridad para el mundo. Este cambio climático afecta a todos los ámbitos de la sociedad y se intensifica tanto con las actividades directamente emisoras de gases de efecto invernadero que con las denominadas actividades difusas o indirectas. La huella de carbono de un

producto software está dada por la cantidad de Gases de Efecto Invernadero (GEI) emitidos durante todo su ciclo de vida: desde la extracción de las materias primas, pasando por el procesado y fabricación y distribución, hasta la etapa de uso y final de la vida útil (depósito, reutilización o reciclado). Un software, aunque no consume energía eléctrica directamente, si condiciona con el procesamiento a la CPU que si consume energía eléctrica.

La reducción de la huella ecológica de las TIC debe abordarse con un enfoque global, en todo su ciclo de vida:

- Diseño y fabricación, sustituyendo o minimizando la presencia de materiales tóxicos.
- Distribución, aplicando métodos de logística optimizada y eficiente para el almacenamiento y distribución.
- Vida útil, mediante un uso responsable de estos productos en cualquiera de los sectores productivos en que se apliquen, tanto por los usuarios como por los gestores de las infraestructuras y servicios TIC.
- Fin de ciclo, garantizando el reciclado eficiente de los productos y servicios.

Algunas medidas que se deben tener en cuenta son:

- Aprovechamiento de luz natural.
- Sustitución de lámparas incandescentes por fluorescentes de bajo consumo.
- Sustitución de balastos electromagnéticos en luminarias.
- Utilizar iluminación con lámparas LED.
- Limpieza regular de ventanas y lámparas.
- Apagado de los equipos eléctricos y electrónicos cuando no se usan.
- Tener en cuenta a la hora de desarrollar el software que un uso elevado del procesador conlleva a más gasto de energía.
- Reducir la movilidad usando las tecnologías como medio de comunicación siempre que sea posible.
- Mejorar los suministros de energía.

Con estas medidas se puede contribuir a reducir la emisión de los GEI generados por un producto software, haciendo que el proceso de su desarrollo tenga una relación menos negativa con el medio ambiente.

2.3.2 - Clasificación para las métricas

En este epígrafe se propondrá un sistema de clasificación por puntos para cada métrica analizada y a su vez, para clasificar el software y dado los resultados, conocer si el software va a ser sostenible o no antes de su desarrollo. Esto no significa que la guía sea definitoria ni que no se pueda desarrollar el producto, pero si va a aportar una idea de que hay que mejorar o cambiar para que sea sostenible.

Se medirá cada métrica analizando una serie de factores a considerar con una puntuación dada, siendo 0 la puntuación más baja. Luego, aplicando la fórmula dada obtendremos un valor que se comparará con la tabla dada, dando así una clasificación a cada métrica por separado.

Mantenibilidad

Factores a considerar y su puntuación

- **Documentación del software.**

  - No presenta documentación: 0

  - Presenta una documentación pobre: 1

  - Presenta una documentación adecuada: 2

- **Simplicidad del código.**

  - Métodos sin comentar y variables no entendibles: 0

  - Métodos y variables poco entendibles: 1

- Métodos y variables entendibles: 2

- **Lenguaje de programación usado.**

  - C++, Ensamblador, Lisp: 0

  - Prolog, C: 1

  - Python, Java: 2

(Los lenguajes de programación están en constante evolución e incluso salen nuevas creaciones. Es recomendable ajustar esta lista teniendo en cuenta los lenguajes de programación más complejos en el momento de ser aplicada la guía)

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor
Sostenible	6-4
Poco Sostenible	3-2
No Sostenible	1-0

Facilidad de uso

Factores a considerar y su puntuación

- **Accesibilidad.**

  - No se tiene en cuenta: 0

  - Presenta un diseño accesible para la mayoría del público o clientes: 1

- Además de su interfaz predeterminada, tiene ajustes que hacen de la aplicación más accesible y así llegar a todos con la máxima comodidad: 2

- **Interfaz gráfica.**

  - Presenta una interfaz gráfica compleja y poco intuitiva: 0

  - Presenta una interfaz gráfica con opciones ventanas poco ordenadas: 1

  - Su interfaz gráfica en sencilla, intuitiva y fácil de usar: 2

- **Diseño minimalista.**

  - Presenta un diseño atiborrado de opciones, colores que dificultan la lectura y la información que brinda no es concisa: 0

  - Presenta un diseño cómodo pero el acceso a las distintas ventanas puede ser tedioso o poco intuitivo para su acceso: 1

  - Se opta por un diseño con colores agradables para la lectura, una buena organización y fácil acceso a las distintas ventanas: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor
Sostenible	6-4
Poco Sostenible	3-2
No Sostenible	1-0

Portabilidad

Factores a considerar y su puntuación

- **Cambio de hardware o software.**

- No se tiene en cuenta el cambio de software o hardware que puede hacer la empresa o cliente que va a consumir el software: 0

- Se desarrolla el software con compatibilidad para una actualización del sistema operativo o incluso una versión anterior: 1

- Se tiene en cuenta los cambios de hardware y de software para que, si en el futuro se produce un cambio de estos, el desempeño del software no se vea afectado: 2

- **Facilidad de instalación.**

- El proceso de instalación es engorroso, al punto de no poder ser instalado por cualquiera persona: 0

- Puede ser instalado por cualquier usuario, pero presenta un nivel de complejidad elevado: 1

- Presenta un proceso de instalación sencillo, capaz de ser instalado por cualquier usuario: 2

- **Adaptación a distintos sistemas.**

- No posee la capacidad de adaptarse a otros sistemas operativos ni a versiones distintas al sistema para el cual fue desarrollado: 0

- Se necesita un proceso complejo y una alta comprensión del código para ser llevado a un sistema diferente: 1

- Presenta gran facilidad para ser llevado de un sistema operativo a otro o ser adaptado a versiones distintas del sistema: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor
Sostenible	6-4
Poco Sostenible	3-2
No Sostenible	1-0

Fiabilidad

Factores a considerar y su puntuación

- **Estimaciones poco realistas.**

- Realizar estimaciones imposibles de lograr en el tiempo de entrega del software: 0

- Realizar estimaciones que, aunque halla posibilidad de lograrse por parte del equipo, comprometan directamente la fecha de entrega del software acordada previamente: 1

- Realizar las estimaciones, en cuanto a requerimientos, de forma tal que se puedan lograr por parte del equipo de programación sin ningún problema ni contratiempo: 2

- **Planificar bien el tiempo, recursos y esfuerzos.**

- Desaprovechar el tiempo y los horarios programados para el desarrollo del software y los recursos que están a la disposición: 0

- No tener un buen manejo del tiempo necesario para el desarrollo del producto o gastar recursos de forma innecesaria: 1

- Aprovechar al máximo el tiempo de trabajo planificado y usar de forma eficaz los recursos que están a disposición: 2

- **Aprovechar los recursos del sistema y conocer limitaciones y restricciones del mismo.**

- No tener idea del funcionamiento o las características del sistema para el cual va a ser desarrollado el software: 0

- Conocer vagamente el sistema en el cual va a ser implementado el software, sin conocer completamente sus limitaciones y características: 1

- Conocer el sistema en el cual va a ser desplegado el software para aprovechar los recursos del mismo, así como sus limitaciones: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor
Sostenible	6-4
Poco Sostenible	3-2
No Sostenible	1-0

Consumo de energía

Factores a considerar y su puntuación

- **Aprovechamiento energético en la etapa del desarrollo.**

- No se aprovecha la luz natural, se usan luminarias que suponen un gasto energético elevado y no se ahorra energía apagando los equipos encendidos innecesariamente: 0

- No se utiliza iluminación natural o se gasta más energía de la necesaria por parte de los equipos electrónicos usados: 1

- Se utiliza la luz natural en lugar de la artificial, se cambian las lámparas o bombillas de alto consumo y se utilizan solo los equipos necesarios: 2

- **Ahorro energético del software.**

- No optimizar el código del programa, tener el programa funcionando siempre de forma innecesaria, aunque no se esté utilizando en primer plano por el usuario: 0

- Hacer optimizaciones pequeñas y no implementar modos de funcionamiento dedicados al ahorro de energía: 1

- Tener en cuenta que el código del software condiciona el uso del hardware, usar líneas de código sencillas e implementar un modo de ahorro de energía, si fuese necesario, para que el programa no esté funcionando permanentemente: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor
Sostenible	4-3
Poco Sostenible	2-1
No Sostenible	0

2.3.3 - Clasificación del software

Teniendo en cuenta el calculo de cada una de las métricas anteriores se tiene que la mantenibilidad tendrá un valor mínimo de 0 y máximo 6, al igual que la fiabilidad, la facilidad de uso y la portabilidad, el consumo de energía tendrá un

mínimo de 0 y un máximo de 4. Estos valores se utilizarán en la siguiente fórmula y se compararán con los valores de la tabla.

Sostenibilidad $= V(1) + V(2) + V(3) + V(4) + V(5)$

(siendo $V(1)$ la puntuación obtenida de la mantenibilidad, $V(2)$ la puntuación obtenida de la fiabilidad, $V(3)$ la puntuación obtenida de la facilidad de uso, $V(4)$ la puntuación obtenida de la portabilidad y $V(5)$ la puntuación obtenida del consumo de energía)

Clasificación	Valor
Sostenible	28-21
Poco Sostenible	20-9
No Sostenible	8-0

2.4 - Conclusiones

A lo largo de este capitulo hemos podido identificar los factores que afectan la sostenibilidad de un software tanto antes de su desarrollo como en su despliegue. La guía propuesta puede medir el grado de sostenibilidad de un software antes de su desarrollo. También esta guía es una herramienta útil a la hora de tomar decisiones en un equipo de programación a la hora de construir un software.

3 - Validación de la Propuesta.

El presente capítulo está dirigido a presentar la validación de la Guía para medir la Sostenibilidad de un producto de software. En la ISO 17025 la validación se define como la “confirmación por examen y la provisión de evidencia objetiva de que se cumplen los requisitos particulares para un uso específico propuesto”. La versión más reciente de su definición se presenta en la norma ISO 9000:2000, en donde se establece que la validación es la “confirmación y provisión de evidencia objetiva de que se cumplen los requisitos para un uso o aplicación prevista”.

3.1 - Aplicación de la guía

La validación es una actividad indispensable para asegurar que un objeto satisfaga los requisitos necesarios para un cierto uso. Los elementos que constituyen una validación son: El uso propuesto o función, el objeto de la validación o herramienta, los requisitos para la función propuesta, el aporte de evidencias objetivas de desempeño de la herramienta, la actividad de examinar y la expresión del resultado. El resultado de una validación debe ser expresada como que la herramienta cumple los requisitos para la función prevista, con una mención explícita a los requisitos y a la función, uso o aplicación, prevista.
Para comprobar la validez de la propuesta se decide aplicar la misma a softwares, que, aunque ya fueron desarrollados, nos permiten comprobar si su desarrollo está en correspondencia con las métricas de sostenibilidad de sistemas informáticos.
Se seleccionan 4 tesis desarrollas en el departamento.

3.1.1 - Tesis 1: Atel, Agente de Telecomunicaciones.

El proyecto Atel se engloba dentro del área de desarrollo de aplicaciones móviles. Ofrece a la División Territorial Cienfuegos contar con una herramienta

de trabajo para el mejoramiento del servicio a la población, ofreciendo una ventaja competitiva en estos momentos en que las TICs toman el mando y el teléfono móvil es el dispositivo a través del que más se accede a Internet. Esta novedosa aplicación, única en el país propiciará el intercambio con productos y servicios, además de buscar en la Red de Ventas (localización), el punto de contacto más cercano al usuario, con sus horarios establecidos, dirección y número telefónico.

Mantenibilidad

Factores a considerar y su puntuación

- **Documentación del software. (2 puntos)**

  - No presenta documentación: 0

  - Presenta una documentación pobre: 1

  - Presenta una documentación adecuada: 2

- **Simplicidad del código. (2 puntos)**

  - Métodos sin comentar y variables no entendibles: 0

  - Métodos y variables poco entendibles: 1

  - Métodos y variables entendibles: 2

- **Lenguaje de programación usado. (2 puntos)**

  - C++, Ensamblador, Lisp: 0

  - Prolog, C: 1

  - Python, Java: 2

(Los lenguajes de programación están en constante evolución e incluso salen nuevas creaciones. Es recomendable ajustar esta lista teniendo en cuenta los lenguajes de programación más complejos en el momento de ser aplicada la guía)

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Facilidad de uso

Factores a considerar y su puntuación

- **Accesibilidad. (2 puntos)**

  - No se tiene en cuenta: 0

  - Presenta un diseño accesible para la mayoría del público o clientes: 1

  - Además de su interfaz predeterminada, tiene ajustes que hacen de la aplicación más accesible y así llegar a todos con la máxima comodidad: 2

- **Interfaz gráfica. (2 puntos)**

  - Presenta una interfaz gráfica compleja y poco intuitiva: 0

  - Presenta una interfaz gráfica con opciones ventanas poco ordenadas: 1

  - Su interfaz gráfica en sencilla, intuitiva y fácil de usar: 2

- **Diseño minimalista. (2 puntos)**

  - Presenta un diseño atiborrado de opciones, colores que dificultan la lectura y la información que brinda no es concisa: 0

- Presenta un diseño cómodo pero el acceso a las distintas ventanas puede ser tedioso o poco intuitivo para su acceso: 1

- Se opta por un diseño con colores agradables para la lectura, una buena organización y fácil acceso a las distintas ventanas: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Portabilidad

Factores a considerar y su puntuación

- **Cambio de hardware o software. (2 puntos)**

- No se tiene en cuenta el cambio de software o hardware que puede hacer la empresa o cliente que va a consumir el software: 0

- Se desarrolla el software con compatibilidad para una actualización del sistema operativo o incluso una versión anterior: 1

- Se tiene en cuenta los cambios de hardware y de software para que, si en el futuro se produce un cambio de estos, el desempeño del software no se vea afectado: 2

- **Facilidad de instalación. (2 puntos)**

- El proceso de instalación es engorroso, al punto de no poder ser instalado por cualquiera persona: 0

- Puede ser instalado por cualquier usuario, pero presenta un nivel de complejidad elevado: 1

- Presenta un proceso de instalación sencillo, capaz de ser instalado por cualquier usuario: 2

- **Adaptación a distintos sistemas. (2 puntos)**

- No posee la capacidad de adaptarse a otros sistemas operativos ni a versiones distintas al sistema para el cual fue desarrollado: 0

- Se necesita un proceso complejo y una alta comprensión del código para ser llevado a un sistema diferente: 1

- Presenta gran facilidad para ser llevado de un sistema operativo a otro o ser adaptado a versiones distintas del sistema: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Fiabilidad

Factores a considerar y su puntuación

- **Estimaciones poco realistas. (2 puntos)**

- Realizar estimaciones imposibles de lograr en el tiempo de entrega del software: 0

- Realizar estimaciones que, aunque halla posibilidad de lograrse por parte del equipo, comprometan directamente la fecha de entrega del software acordada previamente: 1

- Realizar las estimaciones, en cuanto a requerimientos, de forma tal que se puedan lograr por parte del equipo de programación sin ningún problema ni contratiempo: 2

- **Planificar bien el tiempo, recursos y esfuerzos. (2 puntos)**

- Desaprovechar el tiempo y los horarios programados para el desarrollo del software y los recursos que están a la disposición: 0

- No tener un buen manejo del tiempo necesario para el desarrollo del producto o gastar recursos de forma innecesaria: 1

- Aprovechar al máximo el tiempo de trabajo planificado y usar de forma eficaz los recursos que están a disposición: 2

- **Aprovechar los recursos del sistema y conocer limitaciones y restricciones del mismo. (2 puntos)**

- No tener idea del funcionamiento o las características del sistema para el cual va a ser desarrollado el software: 0

- Conocer vagamente el sistema en el cual va a ser implementado el software, sin conocer completamente sus limitaciones y características: 1

- Conocer el sistema en el cual va a ser desplegado el software para aprovechar los recursos del mismo, así como sus limitaciones: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Consumo de energía

Factores a considerar y su puntuación

- **Aprovechamiento energético en la etapa del desarrollo. (2 puntos)**

- No se aprovecha la luz natural, se usan luminarias que suponen un gasto energético elevado y no se ahorra energía apagando los equipos encendidos innecesariamente: 0

- No se utiliza iluminación natural o se gasta más energía de la necesaria por parte de los equipos electrónicos usados: 1

- Se utiliza la luz natural en lugar de la artificial, se cambian las lámparas o bombillas de alto consumo y se utilizan solo los equipos necesarios: 2

- **Ahorro energético del software. (2 puntos)**

- No optimizar el código del programa, tener el programa funcionando siempre de forma innecesaria, aunque no se esté utilizando en primer plano por el usuario: 0

- Hacer optimizaciones pequeñas y no implementar modos de funcionamiento dedicados al ahorro de energía: 1

- Tener en cuenta que el código del software condiciona el uso del hardware, usar líneas de código sencillas e implementar un modo de ahorro de energía, si fuese necesario, para que el programa no esté funcionando permanentemente: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	4-3	4 puntos
Poco Sostenible	2-1	
No Sostenible	0	

Luego de aplicar la guía a cada uno de los factores de las métricas, procedemos a calcular la sostenibilidad del software.

Sostenibilidad $= V(1) + V(2) + V(3) + V(4) + V(5)$

(siendo $V(1)$ la puntuación obtenida de la mantenibilidad, $V(2)$ la puntuación obtenida de la fiabilidad, $V(3)$ la puntuación obtenida de la facilidad de uso, $V(4)$ la puntuación obtenida de la portabilidad y $V(5)$ la puntuación obtenida del consumo de energía)

Clasificación	Valor	Puntuación
Sostenible	28-21	28 puntos
Poco Sostenible	20-9	
No Sostenible	8-0	

Al aplicar el método a Atel, da como resultado que cumple con los parámetros de sostenibilidad teniendo en cuenta las métricas de la guía, por tanto, el software es sostenible.

3.1.2 - Tesis 2: Sistema Informático para la Gestión de Procesos comercial en ETECSA Cienfuegos

El trabajo describe las diferentes etapas seguidas en el desarrollo de un sistema que permitió automatizar e integrar los procesos de la gestión de procesos que se desarrollan en el departamento Soporte y Comercial de la Empresa de Telecomunicaciones de Cuba (ETECSA), a partir de estudios realizados a dichos procesos del departamento. Los procesos fundamentales son la gestión y manipulación de nuevos servicios de cada municipio, así como la actualización de la información correspondiente a cada sector. Durante su desarrollo se utilizó PostgreSQL 9.3 como sistema gestor de bases de datos, Python como lenguaje de programación y Django como framework. Como resultado se implementó una aplicación web para automatizar estos procesos.

Mantenibilidad

Factores a considerar y su puntuación

- **Documentación del software. (2 puntos)**

  - No presenta documentación: 0

  - Presenta una documentación pobre: 1

  - Presenta una documentación adecuada: 2

- **Simplicidad del código. (1 punto)**

  - Métodos sin comentar y variables no entendibles: 0

  - Métodos y variables poco entendibles: 1

  - Métodos y variables entendibles: 2

- **Lenguaje de programación usado. (2 puntos)**

- C++, Ensamblador, Lisp: 0

- Prolog, C: 1

- Python, Java: 2

(Los lenguajes de programación están en constante evolución e incluso salen nuevas creaciones. Es recomendable ajustar esta lista teniendo en cuenta los lenguajes de programación más complejos en el momento de ser aplicada la guía)

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	5 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Facilidad de uso

Factores a considerar y su puntuación

- **Accesibilidad. (1 punto)**

  - No se tiene en cuenta: 0

  - Presenta un diseño accesible para la mayoría del público o clientes: 1

  - Además de su interfaz predeterminada, tiene ajustes que hacen de la aplicación más accesible y así llegar a todos con la máxima comodidad: 2

- **Interfaz gráfica. (2 puntos)**

  - Presenta una interfaz gráfica compleja y poco intuitiva: 0

  - Presenta una interfaz gráfica con opciones ventanas poco ordenadas: 1

  - Su interfaz gráfica en sencilla, intuitiva y fácil de usar: 2

- **Diseño minimalista. (1 punto)**

  - Presenta un diseño atiborrado de opciones, colores que dificultan la lectura y la información que brinda no es concisa: 0

  - Presenta un diseño cómodo pero el acceso a las distintas ventanas puede ser tedioso o poco intuitivo para su acceso: 1

  - Se opta por un diseño con colores agradables para la lectura, una buena organización y fácil acceso a las distintas ventanas: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	4 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Portabilidad

Factores a considerar y su puntuación

- **Cambio de hardware o software. (2 puntos)**

  - No se tiene en cuenta el cambio de software o hardware que puede hacer la empresa o cliente que va a consumir el software: 0

- Se desarrolla el software con compatibilidad para una actualización del sistema operativo o incluso una versión anterior: 1

- Se tiene en cuenta los cambios de hardware y de software para que, si en el futuro se produce un cambio de estos, el desempeño del software no se vea afectado: 2

- **Facilidad de instalación. (1 punto)**

- El proceso de instalación es engorroso, al punto de no poder ser instalado por cualquiera persona: 0

- Puede ser instalado por cualquier usuario, pero presenta un nivel de complejidad elevado: 1

- Presenta un proceso de instalación sencillo, capaz de ser instalado por cualquier usuario: 2

- **Adaptación a distintos sistemas. (2 puntos)**

- No posee la capacidad de adaptarse a otros sistemas operativos ni a versiones distintas al sistema para el cual fue desarrollado: 0

- Se necesita un proceso complejo y una alta comprensión del código para ser llevado a un sistema diferente: 1

- Presenta gran facilidad para ser llevado de un sistema operativo a otro o ser adaptado a versiones distintas del sistema: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	5 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Fiabilidad

Factores a considerar y su puntuación

- **Estimaciones poco realistas. (2 puntos)**

  - Realizar estimaciones imposibles de lograr en el tiempo de entrega del software: 0

  - Realizar estimaciones que, aunque halla posibilidad de lograrse por parte del equipo, comprometan directamente la fecha de entrega del software acordada previamente: 1

  - Realizar las estimaciones, en cuanto a requerimientos, de forma tal que se puedan lograr por parte del equipo de programación sin ningún problema ni contratiempo: 2

- **Planificar bien el tiempo, recursos y esfuerzos. (2 puntos)**

  - Desaprovechar el tiempo y los horarios programados para el desarrollo del software y los recursos que están a la disposición: 0

  - No tener un buen manejo del tiempo necesario para el desarrollo del producto o gastar recursos de forma innecesaria: 1

  - Aprovechar al máximo el tiempo de trabajo planificado y usar de forma eficaz los recursos que están a disposición: 2

- **Aprovechar los recursos del sistema y conocer limitaciones y restricciones del mismo. (2 puntos)**

  - No tener idea del funcionamiento o las características del sistema para el cual va a ser desarrollado el software: 0

  - Conocer vagamente el sistema en el cual va a ser implementado el software, sin conocer completamente sus limitaciones y características: 1

  - Conocer el sistema en el cual va a ser desplegado el software para aprovechar los recursos del mismo, así como sus limitaciones: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Consumo de energía

Factores a considerar y su puntuación

- **Aprovechamiento energético en la etapa del desarrollo. (2 puntos)**

  - No se aprovecha la luz natural, se usan luminarias que suponen un gasto energético elevado y no se ahorra energía apagando los equipos encendidos innecesariamente: 0

  - No se utiliza iluminación natural o se gasta más energía de la necesaria por parte de los equipos electrónicos usados: 1

  - Se utiliza la luz natural en lugar de la artificial, se cambian las lámparas o bombillas de alto consumo y se utilizan solo los equipos necesarios: 2

- **Ahorro energético del software. (2 puntos)**

  - No optimizar el código del programa, tener el programa funcionando siempre de forma innecesaria, aunque no se esté utilizando en primer plano por el usuario: 0

  - Hacer optimizaciones pequeñas y no implementar modos de funcionamiento dedicados al ahorro de energía: 1

- Tener en cuenta que el código del software condiciona el uso del hardware, usar líneas de código sencillas e implementar un modo de ahorro de energía, si fuese necesario, para que el programa no esté funcionando permanentemente: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	4-3	4 puntos
Poco Sostenible	2-1	
No Sostenible	0	

Luego de aplicar la guía a cada uno de los factores de las métricas, procedemos a calcular la sostenibilidad del software.

Sostenibilidad $= V(1) + V(2) + V(3) + V(4) + V(5)$

(siendo $V(1)$ la puntuación obtenida de la mantenibilidad, $V(2)$ la puntuación obtenida de la fiabilidad, $V(3)$ la puntuación obtenida de la facilidad de uso, $V(4)$ la puntuación obtenida de la portabilidad y $V(5)$ la puntuación obtenida del consumo de energía)

Clasificación	Valor	Puntuación
Sostenible	28-21	24 puntos
Poco Sostenible	20-9	
No Sostenible	8-0	

Aplicando la guía a este sistema para la Gestión de Procesos comercial en ETECSA Cienfuegos, tenemos que cumple con los parámetros de sostenibilidad propuestos en la guía.

3.1.3 - Tesis 3: Programa automatizado del Sistema de Gestión de Capital Humano en EQUIFA

El presente trabajo tiene como objetivo la creación de un sistema que permita la gestión de los procesos del departamento de Recursos Humanos en la Empresa Química de Cienfuegos, atendiendo específicamente los procesos de confección y actualización de los Anexos 14, 14A y 14B, así como los reportes estadísticos mensuales. En su desarrollo se utilizó el lenguaje de programación “PHP” para la realización del producto y la base de datos implementada en MySQL. Se logró la realización de todos los requerimientos que estaban previstos con un correcto funcionamiento y sin retraso en la entrega de la aplicación.

Mantenibilidad

Factores a considerar y su puntuación

- **Documentación del software. (1 punto)**

  - No presenta documentación: 0

  - Presenta una documentación pobre: 1

  - Presenta una documentación adecuada: 2

- **Simplicidad del código. (1 punto)**

  - Métodos sin comentar y variables no entendibles: 0

- Métodos y variables poco entendibles: 1

- Métodos y variables entendibles: 2

- **Lenguaje de programación usado. (2 puntos)**

  - C++, Ensamblador, Lisp: 0

  - Prolog, C: 1

  - Python, Java: 2

(Los lenguajes de programación están en constante evolución e incluso salen nuevas creaciones. Es recomendable ajustar esta lista teniendo en cuenta los lenguajes de programación más complejos en el momento de ser aplicada la guía)

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	4 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Facilidad de uso

Factores a considerar y su puntuación

- **Accesibilidad. (1 punto)**

  - No se tiene en cuenta: 0

- Presenta un diseño accesible para la mayoría del público o clientes: 1

- Además de su interfaz predeterminada, tiene ajustes que hacen de la aplicación más accesible y así llegar a todos con la máxima comodidad: 2

- **Interfaz gráfica. (1 punto)**

- Presenta una interfaz gráfica compleja y poco intuitiva: 0

- Presenta una interfaz gráfica con opciones ventanas poco ordenadas: 1

- Su interfaz gráfica en sencilla, intuitiva y fácil de usar: 2

- **Diseño minimalista. (1 punto)**

- Presenta un diseño atiborrado de opciones, colores que dificultan la lectura y la información que brinda no es concisa: 0

- Presenta un diseño cómodo pero el acceso a las distintas ventanas puede ser tedioso o poco intuitivo para su acceso: 1

- Se opta por un diseño con colores agradables para la lectura, una buena organización y fácil acceso a las distintas ventanas: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	3 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Portabilidad

Factores a considerar y su puntuación

- **Cambio de hardware o software. (2 puntos)**

  - No se tiene en cuenta el cambio de software o hardware que puede hacer la empresa o cliente que va a consumir el software: 0

  - Se desarrolla el software con compatibilidad para una actualización del sistema operativo o incluso una versión anterior: 1

  - Se tiene en cuenta los cambios de hardware y de software para que, si en el futuro se produce un cambio de estos, el desempeño del software no se vea afectado: 2

- **Facilidad de instalación. (2 puntos)**

  - El proceso de instalación es engorroso, al punto de no poder ser instalado por cualquiera persona: 0

  - Puede ser instalado por cualquier usuario, pero presenta un nivel de complejidad elevado: 1

  - Presenta un proceso de instalación sencillo, capaz de ser instalado por cualquier usuario: 2

- **Adaptación a distintos sistemas. (2 puntos)**

  - No posee la capacidad de adaptarse a otros sistemas operativos ni a versiones distintas al sistema para el cual fue desarrollado: 0

  - Se necesita un proceso complejo y una alta comprensión del código para ser llevado a un sistema diferente: 1

  - Presenta gran facilidad para ser llevado de un sistema operativo a otro o ser adaptado a versiones distintas del sistema: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Fiabilidad

Factores a considerar y su puntuación

- **Estimaciones poco realistas. (2 puntos)**

  - Realizar estimaciones imposibles de lograr en el tiempo de entrega del software: 0

  - Realizar estimaciones que, aunque halla posibilidad de lograrse por parte del equipo, comprometan directamente la fecha de entrega del software acordada previamente: 1

  - Realizar las estimaciones, en cuanto a requerimientos, de forma tal que se puedan lograr por parte del equipo de programación sin ningún problema ni contratiempo: 2

- **Planificar bien el tiempo, recursos y esfuerzos. (2 puntos)**

  - Desaprovechar el tiempo y los horarios programados para el desarrollo del software y los recursos que están a la disposición: 0

  - No tener un buen manejo del tiempo necesario para el desarrollo del producto o gastar recursos de forma innecesaria: 1

  - Aprovechar al máximo el tiempo de trabajo planificado y usar de forma eficaz los recursos que están a disposición: 2

- **Aprovechar los recursos del sistema y conocer limitaciones y restricciones del mismo. (2 puntos)**

- No tener idea del funcionamiento o las características del sistema para el cual va a ser desarrollado el software: 0

- Conocer vagamente el sistema en el cual va a ser implementado el software, sin conocer completamente sus limitaciones y características: 1

- Conocer el sistema en el cual va a ser desplegado el software para aprovechar los recursos del mismo, así como sus limitaciones: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	6 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Consumo de energía

Factores a considerar y su puntuación

- **Aprovechamiento energético en la etapa del desarrollo. (2 puntos)**

- No se aprovecha la luz natural, se usan luminarias que suponen un gasto energético elevado y no se ahorra energía apagando los equipos encendidos innecesariamente: 0

- No se utiliza iluminación natural o se gasta más energía de la necesaria por parte de los equipos electrónicos usados: 1

- Se utiliza la luz natural en lugar de la artificial, se cambian las lámparas o bombillas de alto consumo y se utilizan solo los equipos necesarios: 2

- **Ahorro energético del software. (2 puntos)**

- No optimizar el código del programa, tener el programa funcionando siempre de forma innecesaria, aunque no se esté utilizando en primer plano por el usuario: 0

- Hacer optimizaciones pequeñas y no implementar modos de funcionamiento dedicados al ahorro de energía: 1

- Tener en cuenta que el código del software condiciona el uso del hardware, usar líneas de código sencillas e implementar un modo de ahorro de energía, si fuese necesario, para que el programa no esté funcionando permanentemente: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	4-3	4 puntos
Poco Sostenible	2-1	
No Sostenible	0	

Luego de aplicar la guía a cada uno de los factores de las métricas, procedemos a calcular la sostenibilidad del software.

Sostenibilidad $= V(1) + V(2) + V(3) + V(4) + V(5)$

(siendo $V(1)$ la puntuación obtenida de la mantenibilidad, $V(2)$ la puntuación obtenida de la fiabilidad, $V(3)$ la puntuación obtenida de la facilidad de uso, $V(4)$ la puntuación obtenida de la portabilidad y $V(5)$ la puntuación obtenida del consumo de energía)

Clasificación	Valor	Puntuación
Sostenible	28-21	23 puntos
Poco Sostenible	20-9	
No Sostenible	8-0	

Al aplicar el método a Atel, da como resultado que cumple con los parámetros de sostenibilidad teniendo

3.1.4 - Tesis 4: Aplicación Web para Gestión de la información necesaria para la elaboración de la pre-nómina.

Este trabajo describe las diferentes etapas seguidas en el desarrollo de un sistema que permitió automatizar e integrar procesos de la gestión de la pre-nómina en la Unidad Organizativa de Capital Humano, de la Empresa de Telecomunicaciones de Cuba (ETECSA). El principal proceso que tiene lugar en la unidad organizativa y al cual está encaminada esta investigación es el Registro de Asistencia y Puntualidad. Durante el proceso de desarrollo de software se utilizó SQLite como sistema gestor de bases de datos, Python como lenguaje de programación y Django como framework. Como resultado se implementó una aplicación web que automatiza los procesos generando rapidez y exactitud con respecto a la información manipulada y reportes que se generan.

Mantenibilidad

Factores a considerar y su puntuación

- **Documentación del software. (1 punto)**

  - No presenta documentación: 0

  - Presenta una documentación pobre: 1

  - Presenta una documentación adecuada: 2

- **Simplicidad del código. (1 punto)**

  - Métodos sin comentar y variables no entendibles: 0

  - Métodos y variables poco entendibles: 1

  - Métodos y variables entendibles: 2

- **Lenguaje de programación usado. (2 puntos)**

  - C++, Ensamblador, Lisp: 0

  - Prolog, C: 1

  - Python, Java: 2

(Los lenguajes de programación están en constante evolución e incluso salen nuevas creaciones. Es recomendable ajustar esta lista teniendo en cuenta los lenguajes de programación más complejos en el momento de ser aplicada la guía)

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	4 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Facilidad de uso

Factores a considerar y su puntuación

- **Accesibilidad. (1 punto)**

  - No se tiene en cuenta: 0

  - Presenta un diseño accesible para la mayoría del público o clientes: 1

  - Además de su interfaz predeterminada, tiene ajustes que hacen de la aplicación más accesible y así llegar a todos con la máxima comodidad: 2

- **Interfaz gráfica. (1 punto)**

  - Presenta una interfaz gráfica compleja y poco intuitiva: 0

  - Presenta una interfaz gráfica con opciones ventanas poco ordenadas: 1

  - Su interfaz gráfica en sencilla, intuitiva y fácil de usar: 2

- **Diseño minimalista. (1 punto)**

  - Presenta un diseño atiborrado de opciones, colores que dificultan la lectura y la información que brinda no es concisa: 0

  - Presenta un diseño cómodo pero el acceso a las distintas ventanas puede ser tedioso o poco intuitivo para su acceso: 1

  - Se opta por un diseño con colores agradables para la lectura, una buena organización y fácil acceso a las distintas ventanas: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	3 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Portabilidad

Factores a considerar y su puntuación

- **Cambio de hardware o software. (2 puntos)**

  - No se tiene en cuenta el cambio de software o hardware que puede hacer la empresa o cliente que va a consumir el software: 0

  - Se desarrolla el software con compatibilidad para una actualización del sistema operativo o incluso una versión anterior: 1

  - Se tiene en cuenta los cambios de hardware y de software para que, si en el futuro se produce un cambio de estos, el desempeño del software no se vea afectado: 2

- **Facilidad de instalación. (1 punto)**

  - El proceso de instalación es engorroso, al punto de no poder ser instalado por cualquiera persona: 0

  - Puede ser instalado por cualquier usuario, pero presenta un nivel de complejidad elevado: 1

  - Presenta un proceso de instalación sencillo, capaz de ser instalado por cualquier usuario: 2

- **Adaptación a distintos sistemas. (2 puntos)**

- No posee la capacidad de adaptarse a otros sistemas operativos ni a versiones distintas al sistema para el cual fue desarrollado: 0

- Se necesita un proceso complejo y una alta comprensión del código para ser llevado a un sistema diferente: 1

- Presenta gran facilidad para ser llevado de un sistema operativo a otro o ser adaptado a versiones distintas del sistema: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	5 puntos
Poco Sostenible	3-2	
No Sostenible	1-0	

Fiabilidad

Factores a considerar y su puntuación

- **Estimaciones poco realistas. (1 punto)**

- Realizar estimaciones imposibles de lograr en el tiempo de entrega del software: 0

- Realizar estimaciones que, aunque halla posibilidad de lograrse por parte del equipo, comprometan directamente la fecha de entrega del software acordada previamente: 1

- Realizar las estimaciones, en cuanto a requerimientos, de forma tal que se puedan lograr por parte del equipo de programación sin ningún problema ni contratiempo: 2

- **Planificar bien el tiempo, recursos y esfuerzos. (1 punto)**

  - Desaprovechar el tiempo y los horarios programados para el desarrollo del software y los recursos que están a la disposición: 0

  - No tener un buen manejo del tiempo necesario para el desarrollo del producto o gastar recursos de forma innecesaria: 1

  - Aprovechar al máximo el tiempo de trabajo planificado y usar de forma eficaz los recursos que están a disposición: 2

- **Aprovechar los recursos del sistema y conocer limitaciones y restricciones del mismo. (1 punto)**

  - No tener idea del funcionamiento o las características del sistema para el cual va a ser desarrollado el software: 0

  - Conocer vagamente el sistema en el cual va a ser implementado el software, sin conocer completamente sus limitaciones y características: 1

  - Conocer el sistema en el cual va a ser desplegado el software para aprovechar los recursos del mismo, así como sus limitaciones: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	6-4	
Poco Sostenible	3-2	3 puntos
No Sostenible	1-0	

Consumo de energía

Factores a considerar y su puntuación

- **Aprovechamiento energético en la etapa del desarrollo. (1 punto)**

- No se aprovecha la luz natural, se usan luminarias que suponen un gasto energético elevado y no se ahorra energía apagando los equipos encendidos innecesariamente: 0

- No se utiliza iluminación natural o se gasta más energía de la necesaria por parte de los equipos electrónicos usados: 1

- Se utiliza la luz natural en lugar de la artificial, se cambian las lámparas o bombillas de alto consumo y se utilizan solo los equipos necesarios: 2

- **Ahorro energético del software. (2 puntos)**

- No optimizar el código del programa, tener el programa funcionando siempre de forma innecesaria, aunque no se esté utilizando en primer plano por el usuario: 0

- Hacer optimizaciones pequeñas y no implementar modos de funcionamiento dedicados al ahorro de energía: 1

- Tener en cuenta que el código del software condiciona el uso del hardware, usar líneas de código sencillas e implementar un modo de ahorro de energía, si fuese necesario, para que el programa no esté funcionando permanentemente: 2

Clasificación

Clasificación $= \sum C$ (siendo C la puntuación obtenida de cada factor)

Clasificación	Valor	Puntuación
Sostenible	4-3	3 puntos
Poco Sostenible	2-1	
No Sostenible	0	

Luego de aplicar la guía a cada uno de los factores de las métricas, procedemos a calcular la sostenibilidad del software.

Sostenibilidad $= V(1) + V(2) + V(3) + V(4) + V(5)$

(siendo $V(1)$ la puntuación obtenida de la mantenibilidad, $V(2)$ la puntuación obtenida de la fiabilidad, $V(3)$ la puntuación obtenida de la facilidad de uso, $V(4)$ la puntuación obtenida de la portabilidad y $V(5)$ la puntuación obtenida del consumo de energía)

Clasificación	Valor	Puntuación
Sostenible	28-21	
Poco Sostenible	20-9	18 puntos
No Sostenible	8-0	

Al aplicar el método a la Web para Gestión de la información necesaria para la elaboración de la pre-nómina, tenemos como resultado un software poco sostenible, según los parámetros propuestos por la guía.

3.2 - Conclusiones

Se valida la propuesta utilizando 4 softwares desarrollados en la carrera. Empleando la documentación generada por la ingeniería de software, permitiendo evaluar los resultados y compararlo con los resultados de los indicadores del software en explotación.

Conclusiones generales

Una vez concluida la investigación se arriban a las siguientes conclusiones:

- Se propone una guía para medición de sostenibilidad de un software antes de su desarrollo, que permita a desarrolladores y clientes tomar decisiones.
- Se valida la propuesta utilizando softwares desarrollados en la carrera demostrando la validez de la propuesta.

Recomendaciones

- Incorporar a la propuesta, métricas más específicas de medición de sostenibilidad de las TIC.
- Incluir la guía de medición de sostenibilidad como práctica en la asignatura ingeniería de requisitos, como instrumento que garantice que los softwares a desarrollar sean sostenibles.
- Automatizar la propuesta mediante una aplicación.

Referencias bibliográficas

[1] G. H. Brundtland, «Informe de la Comisión Mundial de Desarrollo y Medio Ambiente». marzo de 1987.

[2] P. Zarta Ávila, «La sustentabilidad o sostenibilidad un concepto poderoso para la humanidad.» 2018.

[3] H. Komiyama y K. Takeuchi, «Sustainability Science: Building a New Discipline.» Sustainability Science, 1, 1-6., 2006.

[4] M. Yarime y Y. Kajikawa, «Towards institutional analysis of sustainability science: A quantitative examination of the patterns of research collaboration». Sustainability Science 5(1):115-125, noviembre de 2010.

[5] A. King, W. Bartels, y A. Hoballah, «Carrots Sticks Global trends in sustainability reporting regulation and policy.» 2016.

[6] IUCN, «World Conservation Strategy». 1980.

[7] R. W. Bybee, «Planet earth in crisis: how should science educators respond?» The American Biology Teacher, 1991.

[8] Bryan. G. Norton, «Evaluating ecosystem states: Two competing paradigms». Ecological Economics, vol. 14, pp. 113– 127, agosto de 1995.

[9] J. M. Naredo, «Sobre el origen, el uso y el contenido del término sostenible». Documentación Social, 1996.

[10] C. Becker, «The Karlskrona Manifesto for Sustainability Design.» mayo de 2015.

[11] J. García, H. García, y D. López, «La sostenibilidad en los proyectos de ingeniería». ReV AENUI, vol. 6, 2013, 2013.

[12] L. E. Sánchez Alejo, «La vinculación de la informática con el desarrollo sustentable.» diciembre de 2012.

[13] A. Basar Bener, M. Morisio, y A. Miranskyy, «TI Verde y Software Verde». IEEECS, Oct-2014, octubre de 2014.

[14] S. Naumann, E. Kern, M. Dick, y T. Johann, «Sustainable Software Engineering: Process and Quality Models, Life Cycle, and Social Aspects». 2015.

[15] L. Fuentes Fernández, «Ingeniería del Software Sostenible», 2022. https://itis.uma.es/lineainvestigacion/ingenieria-del-software-sostenible/

[16] Visión Sustentable, «Desarrollo de software sostenible para reducir emisiones de carbono.», 2021. https://www.visionsustentable.com/2021/10/21/desarrollo-de-software-sostenible-para-reducir-emisiones-de-carbono/

[17] F. Albertao, J. Xiao, y C. Tian, «Measuring the Sustainability Performance of Software Projects». IBM Research Division, 29 de junio de 2010.

[18] M. Dick y S. Naumann, «Enhancing software engineering processes towards sustainable software product design.» EnviroInfo 2010: Integration of Environmental Information in Europe. Proceedings of the 24th International Conference on Informatics for Environmental Protection, octubre de 2010.

[19] G. Lamin, F. Fabbrini, y M. Fusani, «Software sustainability from a process-centric perspective». 2012.

[20] D. González Bilbao, «Propuesta de un Modelo de Procesos como base para la formación de informáticos del territorio hacia un desarrollo de software sostenible.», Umiversidad de Cienfuegos, Cuba, 2018.

[21] A. Guldner, M. Garling, M. Morgen, y S. Naumann, «Energy Consumption and Hardware Utilization of Standard Software: Methods and Measurements for Software Sustainability». Springer International Publishing AG, 2018, agosto de 2018.

[22] Lic. M. A. Constanzo, S. Casas, y C. Marcos, «Comparación de modelos de calidad, factores y métricas en el ámbito de la Ingeniería de Software». Informes Científicos - Técnicos UNPA 6(1):1, abril de 2014.

[23] A. Cervera Paz, «El modelo de McCall como aplicación de la calidad a la revision del software de gestion empresarial.» enero de 2000.
[24] E. A. Cabrera Ávila, «Modelo Espiral». 7 de septiembre de 2016.
[25] M. Callejas Cuervo, A. C. Alarcón Aldana, y A. M. Álvarez Carreño, «Modelos de calidad del software, un estado del arte.» junio de 2017. [En línea]. Disponible en: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1900-38032017000100236
[26] A. Rosete, «Modelo de Calidad GILB». 30 de octubre de 2015.
[27] L. Briand, K. El Emam, y S. Morasca, «Theoretical and Empirical Validation of SoftwareProduct Measures». Technical Report ISERN-95-03, Fraunhofer Institute for Experimental Software Engineering, Germany, 1995, 1995.
[28] HISOUR, «Métricas e índices de sostenibilidad.», 2020. https://www.hisour.com/es/sustainability-metrics-and-indices-40041/#:~:text=Aunque%20existen%20desacuerdos%20entre%20personas,de%20c%C3%B3mo%20medir%20el%20concepto.
[29] L. Olsina, «Ingeniería Web; Marco de medición y evaluación de calidad.» Departamento de informática.Universidad Nacional de San Luis - La Rioja – Catamarca, 2007, 2007.
[30] P. Lin y R. Bunger, «Guía de métricas de sostenibilidad ambiental para centros de datos». Schneider Electric – Informe técnico del Centro de Investigación de Gestión de la Energía 67 Ver 1 2, 2021.
[31] R. Black, «23 métricas de desarrollo de software que monitorear hoy.», 7 de septiembre de 2020. https://www.computerweekly.com/es/consejo/23-metricas-de-desarrollo-de-software-que-monitorear-hoy
[32] J. D. Erazo, A. S. Florez, y F. J. Pino, «Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte.», 19 de febrero de 2016.
[33] J. E. León Rojas, «Métricas técnicas del software», 2011. https://ingsoftwarejefer.webcindario.com/unidad-3/estrategias-de-prueba-del-software/metricas-tecnicas-del-software.html
[34] EconoSus, «Crean una guía para reducir la huella de carbono en la industria del software.», 10 de diciembre de 2021. https://economiasustentable.com/noticias/crean-una-guia-para-reducir-la-huella-de-carbono-en-la-industria-del-software
[35] S. Montes, «CodeCarbon, la herramienta capaz de calcular la huella de carbono generada por los sistemas informáticos.», 4 de diciembre de 2020. https://www.escudodigital.com/tecnologia/codecarbon-la-herramienta-capaz-de-calcular-la-huella-de-carbono-generada-por-los-sistemas-informaticos_21876_102.html
[36] D. Torres y S. Guzmán, «Gestión Sostenible de Residuos de Aparatos Eléctricos y Electrónicos en América Latina.» UIT, Convenio de Basilea, CRBAS- Centro Regional Basilea para América del Sur, UNESCO, OMS, ONUDI, OMPI, CEPAL 2015, 2015.