Diseño de un prototipo de modelo con parámetros aplicables al aseguramiento de la calidad en el desarrollo de sistemas expertos como rama de la inteligencia artificial apoyado en arquitectura de tecnología de la información y de la solución.

Iván Alejandro Veloza Peñuela

Asesor

Mgtr. Roberto Mauricio Cárdenas Cárdenas

Universidad Nacional Abierta y a Distancia UNAD

Escuela de Ciencias Básicas Tecnología e Ingeniería ECBTI

Maestría en Gestión de Tecnologías de la Información

2023

Agradecimientos

Quiero expresar mi más sincero agradecimiento por el apoyo brindado a mi hija Angie Mariana, en especial, quisiera destacar la dedicación y paciencia que tuvo durante los días y meses que no pude estar compartiendo con ella y por su inmenso amor y apoyo como hija para poder lograr la culminación de mi proyecto de grado.

Agradezco profundamente la oportunidad de contar con un equipo de expertos y profesionales como lo es el cuerpo docente del programa de maestría, quienes me han brindado la asesoría y las herramientas necesarias para desarrollar el proyecto de grado de manera exitosa.

Quiero destacar la labor de mi director de tesis y los jurados, quienes no solo brindaron su conocimiento y experiencia, sino también su comprensión y tolerancia lo cual ha sido un gran apoyo en mi camino hacia la culminación del proyecto de grado el cual ha sido una experiencia enriquecedora e inolvidable.

Resumen

Con el crecimiento en el volumen y la complejidad en el desarrollo de las aplicaciones de software, se ha hecho evidente la importancia de evaluar la calidad en estas aplicaciones. Este trabajo, por lo tanto, encaja en el contexto de identificar la importancia en la evolución del estado del arte referente al aseguramiento de la calidad en la producción de software vinculando en específico a los sistemas expertos.

Esta situación expone la necesidad sentida que tienen las organizaciones de contar con una gestión de calidad para definir, planificar y controlar las actividades a realizar con el objetivo de garantizar la calidad del producto.

Se exponen aspectos relacionados a la metodología de verificación y validación para el desarrollo de software donde se contempla una revisión de los modelos de evaluación, parámetros, atributos y métricas empleadas en la evaluación de la calidad dado que no son un área bien investigada en la construcción y desarrollo de sistemas expertos.

Como resultado se genera un prototipo de modelo como resultado de la aplicación de una metodología para el diseño de prototipos asociado al desarrollo de software, el cual incorpora técnicas que se proyectan como parte del proceso de aseguramiento de la calidad en la producción y desarrollo de software en específico en los sistemas expertos.

Palabras clave: Sistemas expertos, calidad de software, aseguramiento de la calidad, gestión del conocimiento, verificación de calidad.

Abstract

The growth in volume and complexity in software application development, it has become evident the importance of evaluating the quality of these applications. This work fits within the context of identifying the importance in the evolution of the state of the art concerning quality assurance in software production, specifically linking to expert systems.

This situation exposes the felt need of organizations to have quality management to define, plan, and control activities with the aim of ensuring product quality.

Aspects related to the methodology of verification and validation for software development are presented, which include a review of the evaluation models, parameters, attributes, and metrics used in quality evaluation, given that they are not a well-investigated area in the construction and development of expert systems.

As a result, a prototype model is generated from the application of a methodology for prototype design associated with software development, which incorporates techniques that are projected as part of the quality assurance process in software production and development, specifically in expert systems.

Keywords: *Expert systems, software quality, quality assurance, knowledge management, quality verification.*

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Introducción

Con el crecimiento en el volumen y la complejidad en el desarrollo de las aplicaciones de *software*, se ha hecho evidente la importancia de evaluar la calidad en estas aplicaciones. Este trabajo, por lo tanto, encaja en el contexto de identificar la importancia en la evolución del estado del arte referente al aseguramiento de la calidad en la producción de software vinculando en específico a los sistemas expertos.

Esta situación expone la necesidad sentida que tienen las organizaciones de contar con una gestión de calidad para definir, planificar y controlar las actividades a realizar con el objetivo de garantizar la calidad del producto.

Se exponen aspectos relacionados a la metodología de verificación y validación para el desarrollo de *software* donde se contempla una revisión de los modelos de evaluación, parámetros, atributos y métricas empleadas en la evaluación de la calidad dado que no son un área bien investigada en la construcción y desarrollo de sistemas expertos.

Como resultado se genera un prototipo de modelo como resultado de la aplicación de una metodología para el diseño de prototipos asociado al desarrollo de software, el cual incorpora técnicas que se proyectan como parte del proceso de aseguramiento de la calidad en la producción y desarrollo de *software* en específico en los sistemas expertos.

Contexto

La Calidad En el Desarrollo de Software

Con el crecimiento en el volumen y la complejidad en el desarrollo de las aplicaciones de software, se ha hecho evidente la importancia de evaluar la calidad. A partir de esto al realizar la construcción del estado del arte se evidencia un ápice de un problema relacionado con el desarrollo de *software*. A nivel internacional se evidencian avances en el estudio en el área de evaluación de la calidad de *software* desarrollando normas como la ISO las cuales no suficientes en la rama de los sistemas expertos. Con base en el desarrollo de estas normas se puede evidenciar que las organizaciones están iniciando la adopción de los principios de calidad en el desarrollo de *software*, los cuales son capaces de asegurar la construcción de artefactos de *software* con mejor calidad.

Planteamiento del Problema

Un obstáculo para la plena aceptación de los sistemas expertos es la falta de una metodología de verificación y validación que se ve obstaculizada por la falta de documentación estable y métodos no muy adecuados para la validación de los resultados de las pruebas (Grogono et al., 1993). Por lo tanto, se necesitan enfoques metodológicos y disciplinados, así como herramientas para apoyar estos enfoques (Alebeisat et al., 2018).

Así mismo las métricas en la evaluación de la calidad utilizadas en la construcción y desarrollo de sistemas expertos no son un área bien investigada, dado que la aplicación de métricas hace que surjan nuevos e interesantes problemas relacionados con la estructura del conocimiento (Stephen & Addison, 2002).

(Toro & Peláez, 2018) identifico un círculo vicioso al interior de las organizaciones, el cual impedía el desarrollo de métodos para evaluar la calidad de los sistemas expertos: los

sistemas expertos no fueron evaluados porque nadie preguntó, nadie sabía qué y cómo evaluar porque nadie había hecho evaluaciones. A partir de esto sugirió que romper este ciclo y avanzar hacia el desarrollo de una metodología de evaluación efectiva la cual requiere experimentación. Basado en lo anterior se ha logrado identificar que varios de los autores consultados han experimentado mucho con diversos enfoques con el objetivo de diseñar una metodología (McCall et al., 1977) (Sun, 1988) (Grogono et al., 1993).

No todos los problemas se pueden resolverse a través de sistemas expertos. Hay características que indican si esta tecnología debe instrumentalizar o no un problema particular, entonces el análisis de problemas constituye la primera etapa del ciclo de desarrollo de sistemas expertos, contribuyendo fuertemente a la implementación exitosa del sistema. Para facilitar el proceso de análisis de problemas, se distingue algunas condiciones que, si se observan, entre otras, que pueden contribuir a la identificación del nivel de adecuación del uso de tecnología de sistemas expertos para resolver el problema:

Existencia de expertos que dominan el segmento de conocimiento que encierra el problema, pues con exactitud este exactamente este conocimiento el que será el responsable directo de resolver el problema:

Existencia de tareas que para ser realizadas requieren la participación de varios especialistas que, de forma aislada, no tienen los conocimientos suficientes para realizarla, esto indica que el conocimiento necesario para el análisis y la resolución del problema es multidisciplinario;

Existencia de tareas que requieren conocimiento de detalles que, si se olvidan, causan degradación del desempeño o rendimiento;

Existencia de tareas que muestran grandes diferencias entre el desempeño de los mejores y los peores expertos;

Escasez de mano de obra especializada sobre el conocimiento requerido para resolver el problema.

Con el surgimiento de esta técnica, existen algunos aspectos importantes aun inexplorados como, por ejemplo, el aumento significativo en la productividad de un experto en la ejecución de tareas especializadas, cuando es asistido por un sistema experto.

Otro aspecto relevante es la portabilidad de estos sistemas expertos, ya que pueden desarrollarse y usarse en microcomputadores, esto los hace bastante populares y asequibles. En general los sistemas con pensamiento automatizado pueden usarse incorporando bases de datos que ya existen en la organización, o incorporándose al conjunto de herramientas disponibles en las bases de datos.

Desde este punto de vista la ciencia de la información y muchas otras áreas pueden encontrar los sistemas expertos como herramientas eficientes para la gestión de la información. Brindar herramientas para apoyar la toma de decisiones, en este caso va más allá de proporcionar gráficos y tablas al usuario, significa brindarles orientación para identificar sus necesidades, simular escenarios, permitir una mayor precisión y confiabilidad en sus resultados.

Por lo tanto, un obstáculo para la plena aceptación de los sistemas expertos es la falta de una metodología de verificación y validación en el desarrollo de *software* que se ve obstaculizada por la falta de documentación estable y métodos no muy adecuados para la evaluación de los resultados de las pruebas (Alebeisat et al., 2018). Por lo tanto, se necesitan enfoques metodológicos y dispersos, así como herramientas para apoyar estos enfoques (Alebeisat et al., 2018) para garantizar la calidad del *software* diseñado como sistema experto

donde deben correlacionarse con criterios y métricas específicos, preferiblemente especializados para los diversos dominios de aplicación. Esto último se logra realizando evaluaciones de productos de *software* organizando estas características de calidad en modelos.

A lo largo del proceso de desarrollo de *software*, es necesario que la gestión de calidad defina, planifique y controle las actividades que se realizarán para garantizar la calidad del producto (Andrew, 1998). También es función de la gestión organizar los equipos necesarios para llevar a cabo estas actividades y sembrar la importancia de la calidad en la organización. Esto último implica crear una cultura en la organización con respecto a la calidad al destacar sus beneficios (Basili & Musa, 2002) y que el enfoque está en evaluar el producto y no las personas involucradas en su desarrollo (SinghThapar et al., 2012).

Objetivos

Objetivo General

Diseñar un prototipo de modelo que contenga un conjunto de parámetros aplicables para el aseguramiento de la calidad en el desarrollo de sistemas expertos como rama de la inteligencia artificial apoyado en Arquitectura de Tecnología de la Información y de la solución.

Objetivos Específicos

Desarrollar el estudio del arte relacionado al aseguramiento de la calidad en el desarrollo de sistemas expertos.

Identificar un conjunto de parámetros para el aseguramiento de la calidad a lo largo del desarrollo de un sistema experto.

Definir un conjunto de procedimientos para el aseguramiento de la calidad en el desarrollo de sistemas expertos.

Calidad de Software

Con el avance del tiempo y del desarrollo tecnológico en el campo del *software*, se observa un crecimiento respecto al volumen y la complejidad de las aplicaciones de *software*, este crecimiento también ha hecho evidente la importancia de incorporar el termino calidad en el desarrollo de software. Dado que el concepto de calidad ha sido un tema discutido por diversos autores, aunado al aumento del volumen y la complejidad en el desarrollo de las aplicaciones de *software*, este tema ha logrado situarse como el vértice del problema en el desarrollo de nuevas aplicaciones. El progreso en esta área ha sido significativo, pero no suficiente. Dado que las organizaciones incorporan un gran esfuerzo para adoptar principios de aseguramiento de la calidad en el desarrollo de *software* capaces de garantizar la construcción de productos de mejor calidad.

Contexto histórico y tendencias

La propia evolución histórica del desarrollo de *software* ya vaticinaba un momento en que la mera existencia no satisfaría al mercado, que se volvería más exigente en el sentido no solo de producir o comprar productos de *software*, sino de producir y comprar *software* de calidad (Basili & Musa, 2002). Por lo tanto, la construcción de *software* requiere cada vez más un proceso sistemático y controlado en su desarrollo. El mercado de consumo se ha vuelto más riguroso en la elección de sus productos y su calidad se ha convertido en un factor crítico para garantizar el éxito. Por otro lado, los desarrolladores de *software* sienten la necesidad de mejorar la calidad de sus productos para seguir siendo competitivos en el mercado.

Como parte de la construcción del estado del arte se resalta la década de 1960 la cual se conoció como la era funcional debido a la introducción de las Tecnologías de la Información (TI) en las instituciones y el comienzo de la adaptación de sus servicios y funciones al *software*. En la

década de 1970, la introducción de modelos de ciclo de vida buscó cumplir con la necesidad del mercado con relación al desarrollo de *software* de forma planificada, controlada y en los plazos establecidos. En la década siguiente 1980, la progresiva caída del precio del *hardware* impulsó la preocupación por la búsqueda de un menor coste en el desarrollo. Esto permite la observación del fenómeno que muestra la expansión y el incremento del uso de modelos para estimar costos y recursos como parte importante de la productividad para la construcción de *software* (Karg et al., 2011).

En el momento actual se evidencian dos tendencias, la primera es la incorporación del concepto de calidad en la construcción de *software*, el cual se lleva al centro del proceso de desarrollo. La segunda está orientada a la reducción del número de proveedores de *software* por parte de las organizaciones sin importar su naturaleza (pública o privada), excluyendo a aquellos fabricantes de *software* que no son capaces de brindar calidad en sus productos. Algunas instituciones, como el gobierno y las multinacionales en Europa, ya utilizan la certificación según la norma de calidad ISO como requisito previo para elegir sus proveedores de *software* (Moreno et al., 2010). Por lo tanto, esta tendencia permite entender y discernir que, a mayor madurez del *software* en el mercado, mayor es la demanda de calidad por parte de los usuarios.

(Collins et al., 1994) considera que el proceso de calidad en el desarrollo de *software* desde su creación hasta su uso efectivo, como un proceso social al que llama penumbra, dado que involucra a compradores, desarrolladores, usuarios y otras personas que pueden verse afectadas por el uso del *software*. En este contexto, propone que cada uno de estos actores tenga responsabilidades éticas para minimizar los riesgos encontrados y derivados de su participación en este proceso y con ello contribuir a mejorar la calidad del producto. Estas responsabilidades se traducen a partir de la definición de principios que definen que el *software* no debe aumentar el

grado de riesgo (financiero o personal), en situaciones ya vulnerables por desconocimiento del artefacto de *software* a desarrollar.

Se están estableciendo estándares de calidad, garantizando a los compradores el establecimiento de los requisitos de desempeño del *software* determinados en el contrato, lo que puede llevar a los desarrolladores a recibir acusaciones de negligencia, desvío de propósitos, incluso poca profesionalidad en caso de incumplimiento de los requisitos, debiendo por lo tanto ser conscientes de las normas legales de competencia al construir sus productos. Según (Collins et al., 1994), las consideraciones éticas en el desarrollo de *software* no solo pueden unificar aspectos de estos estándares legales, sino también contribuir de forma significativa a mejorar y estandarizar el tema de la calidad en el *software*.

Un factor que contribuye a la expansión de la política de calidad es el reconocimiento que han dado los desarrolladores de *software* que cuentan con un sistema efectivo de calidad, lo cual se traduce a un aumento de la productividad y en la reducción de los costos en el desarrollo de sus productos (Karg et al., 2011).

Es cierto que el costo de la corrección tardía de los defectos del *software* construido es mucho mayor que si esta corrección se realiza durante el proceso de construcción del *software* (Karg et al., 2011). La reducción de los costos de desarrollo se logra reorientando los recursos hacia la prevención de errores. Por lo tante la introducción e incorporación de un sistema de calidad es un paso esencial en esta dirección. Además, el costo inicial de establecer un sistema de calidad pronto se ve compensado por el resultado generado en el trabajo de construcción.

La necesidad de calidad en el *software* está impulsando el mercado actual tanto internamente en las empresas en el desarrollo de sus productos, como externamente en el competitivo entorno empresarial. (Ortega et al., 2003)sugiere emplear el ciclo Deming dado que

este proceso es una reacción en cadena al ser iterativo y aplicable al proceso de construcción del *software*, como se muestra en la Figura 1. Por tanto, se entiende que mejorar la calidad en el desarrollo implica mejorar la productividad, lo que conduce a la reducción de costos, aumentando así el mercado, dando como resultado un crecimiento en el negocio de la organización que genera el rendimiento de las inversiones.

Figura 1

Adaptación del modelo Deming como parte del proceso de desarrollo de software centrado en la calidad

Nota. Elaboración propia

La calidad del software en Colombia

Con el crecimiento del mercado internacional y la demanda de mejores productos, es necesario hacer una pequeña revisión en la industria del producción de *software* nacional, dado que en ese contexto se requiere de una profunda modernización industrial a través de la creación de la política pública, siendo esencial vincular la calidad y la productividad en la construcción de *software* nacional.

Al hacer la revisión de documentos académicos nacionales que hablen sobre el tema del calidad en el *software* se evidencia que se encuentran muy poca literatura relacionada de forma directa o indirecta. Ahora en materia de política pública en Colombia no se encuentra algún tipo de documentación, literatura o iniciativa generada por el gobierno nacional a través de sus ministerios y/o entidades como lo es el Ministerio de las Tecnologías de la Información y las Comunicaciones (MINTIC).

Características de la calidad del software

En la actualidad es incuestionable que la calidad es una preocupación fundamental en el desarrollo de *software*. (Tucupa, 2020) valora y justifica esta preocupación al afirmar que la calidad es fundamental para la supervivencia y el éxito de las empresas. Ante este contexto, las empresas que desarrollan *software* buscan adoptar procedimientos cuidadosos y rigurosos a lo largo del proceso de desarrollo de sus productos para lograr la meta de calidad en sus productos.

De acuerdo con el glosario incluido por la Organización Internacional de Estándares (*International Organization for Standardization* ISO, por sus siglas en inglés) consignado en la primera edición de la norma ISO 8402:1986, la calidad es el conjunto de características de un producto o servicio que le otorgan la capacidad de satisfacer las necesidades implícitas y explícitas de sus usuarios. Esta definición, por tanto, identifica la calidad con la satisfacción del

usuario. McCall et al., (1977) resume este concepto definiendo la calidad del *software* como un conjunto de propiedades que deben ser satisfechas en cierto grado, para que el *software* satisfaga las necesidades de sus usuarios.

Según (Basili & Musa, 2002) la calidad es un concepto multidimensional cuyas dimensiones incluyen la entidad de interés por ejemplo: especificación de requisitos, proyecto, producto final, entre otros, el punto de vista sobre esta entidad por ejemplo: visión del desarrollador, visión del usuario final, vista de gestión, entre otros y las características de calidad relevantes para la organización. Esto permite comprende y asociar que el proceso de desarrollo de *software* está relacionado con la calidad en su proceso de desarrollo y las características de calidad del producto en sí. En investigaciones como la realizada por (P. Miguel et al., 2014) confirman el reconocimiento de esta realidad por parte de desarrolladores y usuarios.

Los procedimientos para el aseguramiento de la calidad del *software* en relación con el proceso de desarrollo se describen en la norma ISO 90003 denominada Ingeniería de *software*, la cual es una guía para la aplicación de la norma ISO 9001 al *software* informático, esta ha sido desarrollada para organizaciones que aplican la norma ISO 9001 en la adquisición, suministro, desarrollo, operación y mantenimiento de *software* informático y servicios de soporte relacionados (P. Miguel et al., 2014) (SinghThapar et al., 2012) (Moreno et al., 2010). Esta norma consta de tres partes:

Estructura

Que describe los aspectos organizativos a considerar en la producción de *software* (responsabilidad de la dirección, sistemas de calidad, auditorías internas del sistema de calidad, acción correctiva)

Actividades del ciclo de vida

Que define las acciones necesarias para llevar a cabo cada fase del proceso de desarrollo (revisión del contrato, especificación de los requisitos del cliente, planificación del desarrollo, planificación de la calidad, diseño e implementación, prueba y validación, aceptación, copia, entrega y mantenimiento)

Actividades de soporte

Que define las actividades para apoyar la producción, entrega y mantenimiento de *software* (gestión de configuración, control de documentos, registros de calidad, mediciones, reglas generales y convenciones, herramientas y técnicas, compras, adquisición, producto de *software* incluido y capacitación).

Características de Calidad del Software según ISO 9126 & ISO/IEC 25010

Las características o propiedades de calidad contenidos en la norma ISO 8402 es descrita y aplicada a los productos de *software* a partir de la norma ISO/IEC 9126, la cual fue transformada y especificada para definir modelos de evaluación de la calidad de *software* y sistemas desde 2011 por la ISO/IEC 25010. De acuerdo con (Moreno et al., 2010) los atributos o propiedades de un producto de *software* a través de los cuales se puede describir y evaluar su calidad. Estas características deben estar relacionadas con los requisitos de calidad establecidos para el producto.

Sin embargo, los requisitos de calidad para los productos de *software* son difíciles de definir, varios factores contribuyen a esta dificultad. Dado que diferentes áreas de aplicación tienen diferentes requisitos de calidad. Además, diferentes proyectos en la misma área de aplicación pueden tener diferentes requisitos de calidad. Los requisitos de calidad para una aplicación también pueden entrar en conflicto, ya que lograr un requisito en mayor grado puede significar una pérdida en otro requisito.

En el estándar ISO/IEC 25010 se tiene por objetivo definir seis (6) características genéricas de calidad deseables para un producto de *software*:

Funcionalidad se refiere a la existencia de un conjunto de funciones que satisfacen necesidades establecidas o implícitas y sus propiedades específicas.

Confiabilidad se refiere a la capacidad del *software* para mantener su nivel de desempeño bajo condiciones establecidas por un período de tiempo establecido.

Usabilidad se refiere al esfuerzo requerido para poder usar el *software,* así como el juicio individual de este uso por parte de un conjunto establecido o implícito de usuarios.

Eficiencia se refiere a la relación entre el nivel de rendimiento del *software* y la cantidad de recursos utilizados, en condiciones establecidas.

Mantenibilidad se refiere al esfuerzo requerido para realizar cambios específicos en el *software*.

Portabilidad se refiere a la capacidad del *software* para transferirse de un entorno a otro.

En la tabla 1 se consolidan las características mencionadas con su correspondientes subcaracterísticas y sus descripciones correspondientes con el fin de establecer los atributos de calidad.

Tabla 1

Características y subcaracterísticas de calidad del software contenidas en la norma ISO/IEC 25010

Características	Subcaracterísticas	Descripción
Funcionalidad	Adecuación	Atributos de *software* que evidencian la presencia de un conjunto de funciones y su idoneidad para tareas específicas
	Precisión	Atributos del *software* que evidencian la generación de resultados y efectos correctos o esperados
	Interoperabilidad	Atributos del producto de *software* que evidencian su capacidad para interactuar con sistemas específicos
	Conformidad	Atributos del *software* que hacen que cumpla con los estándares, convenciones o reglamentos previstos en las leyes y descripciones similares relacionadas con la aplicación
	Seguridad	Atributos del *software* que evidencian su capacidad para evitar el acceso no autorizado, accidental o deliberado a programas y datos.
Confiabilidad	Madurez	Atributos de *software* que muestran la frecuencia de fallas debido a defectos del *software*.
	Tolerancia a fallos	Atributos del *software* que evidencian su capacidad para mantener un nivel específico de rendimiento en caso de falla del *software* o violación de interfaces específicas
	Recuperabilidad	Atributos del *software* que demuestran su capacidad para restaurar su nivel de rendimiento y recuperar los datos directamente afectados, en

		caso de falla, y el tiempo y esfuerzo necesarios para hacerlo
Eficiencia	Comportamiento en relación con el tiempo	Atributos del *software* que muestran su tiempo de respuesta, tiempo de procesamiento y velocidad en la ejecución de sus funciones
	Comportamiento con relación a los recursos	Atributos del *software* que muestran la cantidad de recursos utilizados y la duración de su uso en la ejecución de sus funciones
Utilizabilidad	Inteligibilidad	Atributos del *software* que muestran el esfuerzo del usuario por reconocer el concepto lógico y su aplicabilidad
	Aprehensibilidad	Atributos de *software* que demuestran el esfuerzo del usuario para aprender su aplicación (por ejemplo, control de operación, entradas, salidas)
	Operabilidad	Atributos del *software* que muestran el esfuerzo del usuario para su funcionamiento y control de su funcionamiento
Mantenibilidad	Analizabilidad	Atributos de *software* que resaltan el esfuerzo requerido para diagnosticar deficiencias o causas de fallas o para identificar partes a modificar
	Modificabilidad	Atributos del *software* que muestran el esfuerzo requerido para modificarlo, eliminar sus defectos o adaptarlo a los cambios ambientales
	Estabilidad	Atributos de *software* que muestran el riesgo de efectos inesperados causados por modificaciones
	Testabilidad	Atributos de *software* que demuestran el esfuerzo requerido para validar el *software* modificado
Portabilidad	Adaptabilidad	Atributos del *software* que demuestran su capacidad para adaptarse a diferentes entornos especificados, sin necesidad de aplicar otras

	acciones o medios más allá de los proporcionados para esta habilidad por el *software* considerado
Capacidad para ser instalado	Atributos de *software* que demuestran el esfuerzo requerido para instalarlo en un entorno específico
Conformidad	Atributos del *software* que lo hacen compatible con los estándares relacionados con la portabilidad
Capacidad para reemplazar	Atributos del *software* que demuestran su capacidad y esfuerzo requerido para reemplazar en el entorno establecido otro *software*

Nota. Elaboración propia

Una encuesta realizada en varios países de la Comunidad Europea (Moreno et al., 2010), comprobó la importancia ya alcanzada por la ISO/IEC 25010 (al menos el 90% de los entrevistados ya había oído hablar de la norma) aunque se trata de una norma muy reciente. El autor también concluye que ISO/IEC 25010 es realmente el estándar para el modelado de la calidad del *software*.

Sin embargo, para permitir la evaluación del *software*, de acuerdo con las características y subcaracterísticas identificadas, es necesario correlacionarlas con criterios y métricas específicas, preferiblemente especializadas para los diferentes dominios de aplicación.
Estos procedimientos asociado a la norma ISO son nombrados solo como parte de la reconstrucción literaria abordado por uno de los objetivo del presente documento, pero estos no serán abordados ya que están fuera del alcance.

Modelos para la Evaluación de la Calidad

Los objetivos de calidad se logran a través de factores de calidad que pueden estar compuestos por subfactores y se evalúan a través de criterios. Los criterios definen atributos de calidad para los factores. Las medidas son valores resultantes de la evaluación de un producto según un criterio específico.

Por lo tanto, la calidad del *software* se logra a través de los resultados obtenidos a través de las medidas aplicadas en un conjunto de características. Para poder realizar evaluaciones en productos de *software*, es necesario organizar estas características de calidad en modelos.

Existen varios modelos en la literatura para evaluar la calidad del *software*: el *Software Science* propuesto por (Moreno et al., 2010) basado en el conteo de operadores y operandos que aparecen en la implementación. El modelo propuesto por Boehm (1978) citado por (Moreno et al., 2010) (Singh & Kannojia, 2013) que define un árbol de características de calidad y los modelos resultantes de las experiencias de la *Computer Services Association* (Moreno et al., 2010).

En el contexto del presente trabajo se considera como base el modelo propuesto por (McCall et al., 1977) dado que este modelo permite definir características de calidad y procedimientos para la evaluación de *software*, actividad que será la base para la construcción del modelo de prototipo y sobre todo la base para la selección de los parámetros que se encuentren en la revisión documental.

Modelo propuesto por McCall

El modelo definido por (McCall et al., 1977) se basa en los siguientes conceptos:

Objetivos de calidad: son propiedades generales que debe tener el producto;

Factores de calidad: son atributos que determinan la calidad del producto desde el punto de vista de sus diferentes usuarios;

Criterios: son atributos primitivos que se pueden evaluar;

Procesos de evaluación: determinar las métricas a utilizar para medir el grado de presencia, en el producto, de un determinado criterio

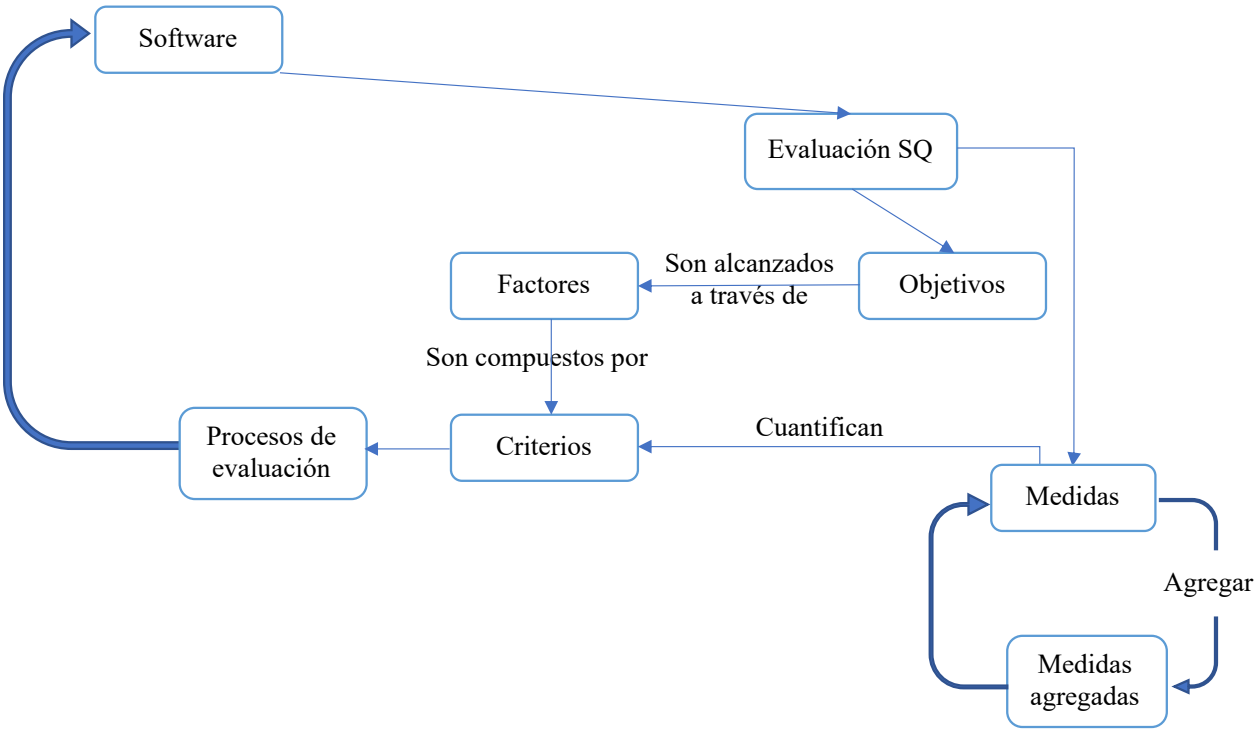
Medidas: indicar el grado de presencia, en el producto, de un determinado criterio

Medidas agregadas: indican el grado de presencia, en el producto, de un determinado factor.

Los objetivos y factores no se pueden medir directamente y solo se pueden evaluar utilizando criterios. Un criterio es un atributo primitivo, en otras palabras, un atributo es independiente de todos los demás atributos. Los criterios únicos no constituyen una descripción completa de un factor o subfactor determinado. Del mismo modo, ningún factor por sí solo define completamente un objetivo como se puede observar en la figura 2 en la cual se muestra la estructura de este modelo.

Figura 2

Adaptación de la estructura del modelo de evaluación de la calidad de software propuesto por McCall et al. (1977)



Nota. Elaboración propia

Inicialmente (McCall et al., 1977) emplearon el modelo para definir la calidad y permitir la evaluación de las especificaciones de requisitos. Posterior a esto, se amplió el trabajo considerando las características de especificaciones de proyecto (Groundwater et al., 1995) (Flores Zafra & Gardi Melgarejo, 2020) (Moreno et al., 2010) (Singh & Kannojia, 2013) y código (Madou et al., 2009) y las particularidades de varias áreas de aplicación (Herrera & Ramírez, 2003) en el *software* científico (León Martínez et al., 2011) y en MyPimes (Toro & Peláez, 2018).

Métricas de software

Las métricas de evaluación de *software* se utilizan para permitir la cuantificación del grado en que los atributos están presentes en un producto de *software*, de acuerdo con un modelo de medición específico.

En la literatura mucho se ha discutido sobre el significado de métricas y medidas. Por ejemplo, (Stephen & Addison, 2002) define la medición como el proceso mediante el cual se asignan números o símbolos a atributos de entidades del mundo real de tal manera que se describen de acuerdo con reglas claramente definidas. Por lo tanto, para que la medición sea posible debemos saber qué entidades se están analizando y los atributos de la entidad a cuantificar (Grogono et al., 1993).

Por tanto, se entienden las métricas como medidas numéricas o simbólicas que sirven para indicar la adecuación de *software* a las características de calidad con las cuales está relacionado (Hauge et al., 2006) (Gao et al., 2019) (Martín & Sáenz, 2016).

Existen varias clasificaciones para las métricas en la literatura: métricas objetivas y subjetivas (Garousi & Mäntylä, 2016), directas e indirectas (Adewumi et al., 2016) (Kaur, 2020) y métricas de producto y proceso (Tosun et al., 2017)

En la clasificación de (Hauge et al., 2006) las métricas objetivas se cuantifican y miden fácilmente a través de expresiones numéricas o representaciones gráficas de expresiones numéricas que se pueden calcular a partir de documentos de *software* como código fuente, proyecto, datos de prueba, entre otros. Las métricas subjetivas son menos cuantificables, siendo medidas relativas basadas en estimaciones personales o grupales y generalmente obtenidas por clases de respuesta como excelente, regular, buena, mala.

Las métricas pueden ser medidas directas o indirectas. Las mediciones directas son aquellas que no dependen de la medición de otro atributo, permitiendo la cuantificación de un factor observado en el producto. Las medidas indirectas de un atributo implican medidas de uno o más atributos relacionados con él (Wedyan & Abufakher, 2020) (Madou et al., 2009) (Herrera & Ramírez, 2003) utiliza una clasificación similar a ésta, nombrando métricas de predicción y de resultado y hace una asociación con su clasificación de métricas de producto y de proceso.

Una métrica de *software* es un producto cuando el valor se obtiene de algún documento o código de programa y métrica de proceso es cuando el valor describe el proceso de *software*. Como resultado la métrica de predicción por lo general se considera una métrica de producto que se puede usar para predecir otra métrica, esta también es denominada métrica de resultado, que suele ser una métrica de proceso. Usar las características de una especificación de sistema para predecir la cantidad de recursos que se necesitarán en un proyecto de *software* es un ejemplo de una especificación de sistema conocida como métrica de producto que se usa para predecir parte de los recursos de proyecto llamada también métrica resultante.

McCall et al., (1977) propone que para definir y seleccionar métricas que sean más aplicables a un entorno de desarrollo específico se deben analizar sus características organizativas y técnicas. En ese contexto se entiende que las características organizacionales se

refieren a la aplicación de métricas dentro de un ambiente organizacional, observando varios aspectos. Se debe desarrollar un plan de aplicación de métricas para áreas cuyos resultados iniciales traerán el mejor retorno de la inversión. Se debe garantizar la aplicación de métricas a lo largo del ciclo de desarrollo de *software* y se deben utilizar las métricas como una herramienta de calidad en la gestión de proyectos y como base para las mejoras en el proceso de desarrollo. Además, la organización debe incentivar, supervisar y apoyar el uso de métricas, enfatizando que las métricas no deben ser una determinación como parte de la política calidad, para que no tenga impactos morales negativos, buscando mostrar que las métricas no son para evaluar el desempeño individual de los miembros del equipo sino para mejorar el proceso de desarrollo.

Las características técnicas se aplican a la definición de la propia métrica. Se debe utilizar un número limitado de métricas que sean fáciles de determinar, cuyos datos necesarios estén fácilmente disponibles y que estén sujetos a experimentación. Además, debe haber un compromiso con los estándares organizacionales para las métricas.

Así mismo es importante indicar que el uso de métricas se puede emplear para diferentes propósitos, como por ejemplo para estimar los recursos del proyecto, como un mecanismo para evaluar el desempeño del proyecto y del equipo de desarrollo, para evaluar los métodos de desarrollo y para ayudar al equipo de desarrollo a obtener buenas estimaciones de la calidad de su trabajo.

Por lo tanto, se puede inferir que una de las principales preocupaciones de la industria y de los investigadores de métricas de *software* se refieren a la falta de validaciones empíricas de métricas, lo anterior en otras palabras se refiere a las validaciones para métricas de diseño y especificaciones, junto a la falta de experimentos para estas validaciones y asociado a la falta de

herramientas de apoyo (Herrera & Ramírez, 2003) (SinghThapar et al., 2012) (Basili & Musa, 2002).

Gestión de la calidad del software

A lo largo del proceso de desarrollo de *software* es necesario contar con una gestión de calidad para definir, planificar y controlar las actividades a realizar con el objetivo de garantizar la calidad del producto (Nguyen-Duc et al., 2015). También es función de esta gestión organizar los equipos necesarios para realizar estas actividades y sembrar la importancia de la calidad en la organización. Esto último implica crear una cultura en la organización respecto a la calidad, destacando sus beneficios (Nguyen-Duc et al., 2015) (Andrew, 1998) y que el foco buscado sea evaluar el producto y no las personas involucradas en su desarrollo (Sun, 1988).

La cultura de la calidad implica atención a la satisfacción del usuario, énfasis en la mejora continua del proceso de desarrollo, capacitación de los equipos y compromiso de todos con la gestión (Nguyen-Duc et al., 2015) (Schulmeyer, 2008). También es responsabilidad de la dirección de calidad informar sobre las normas y procedimientos a realizar, controlar y supervisar la realización de estos procedimientos y garantizar la calidad mediante procesos de auditoría y certificación de la calidad alcanzada por el producto a realizar por equipos específicos para tal fin (Guerra et al., 2014). Además, es necesario revisar y evaluar constantemente los resultados de la política de calidad para mejorar el proceso.

El control de calidad del *software* es el conjunto planificado y sistemático de todas las acciones necesarias para brindar la confianza adecuada de que el artículo o producto cumple con los requisitos técnicos establecidos ANSI/IEEE STD 730-2014. Gestionar la calidad del *software* implica planificar y llevar a cabo el control de calidad.

Planificación de control de calidad

Cuando se habla de calidad *software*, esto se refiere a todos los productos generados durante el desarrollo, ya sean especificaciones o el producto final como tal. De esta forma se

entiende que el control de calidad debe realizarse desde el inicio del desarrollo (McCall et al., 1977) (Moreno et al., 2010).

Por tanto, la planificación del control de calidad de un proyecto implica, en primer lugar, identificar las características de calidad de interés para el producto a desarrollar, establecer la importancia de cada una de estas características para la satisfacción del usuario e identificar las relaciones y posibles conflictos entre ellas. Para poder realizar evaluaciones sobre el producto, también se deben definir procesos (o métricas) de evaluación para cada característica identificada como relevante para el producto y especificar el grado en que se quiere lograr la característica.

Con estas definiciones se establecen hitos y puntos de control donde se evaluarán los productos generados durante el desarrollo. Luego se genera el Plan de Control de Calidad (PCC), que debe contener la descripción de todos los procedimientos a adoptar en el proyecto para el control de calidad a lo largo del desarrollo y evaluación de la calidad del producto final, además de definir el equipo de control de calidad. A partir de esto la tarea de controlar si se cumplen los requisitos de calidad es una actividad presente durante todo el ciclo de vida del producto de acuerdo con el PCC, el cual puede contar con más o menos elementos y establece una hoja de ruta con los elementos conceptuales que pueden ser incorporados en el PCC de acuerdo con el estándar ANSI/IEEE STD 730 - 2014.

Técnicas para el Control de Calidad del Software

Las evaluaciones para el control de calidad del *software* deben estar presentes durante todo el ciclo de vida para garantizar que se puedan lograr:

Los requisitos establecidos

Identificar los requisitos que no se pueden lograr

Garantizar que el *software* se represente de acuerdo con estándares predefinidos

Garantizar que el *software* se desarrolle de manera uniforme

Descubrir errores para tomar medidas correctivas lo antes posible y hacer que el proyecto sea más manejable.

Las evaluaciones incluyen procesos de verificación para garantizar la corrección y coherencia con respecto a los entregables de cada fase y los estándares proporcionados como entrada del proceso para cada fase; validación para evaluar la idoneidad del producto de *software* para sus propósitos, asegurando que cumple con los requisitos especificados y prueba con ejecución de código para producir resultados para ser analizados (Martín & Sáenz, 2016) (Collins et al., 1994).

Las evaluaciones deben realizarse utilizando técnicas de control de calidad. El uso de estas técnicas debe estar adecuadamente planificado y controlado para lograr los objetivos propuestos para la evaluación. Estas técnicas tienen diversos grados de formalidad y son adecuadas para diferentes tipos de proyectos (Gao et al., 2019). En la literatura se pueden encontrar muchas de estas técnicas (Moreno et al., 2010), por ejemplo:

Control estadístico

Prueba

Prueba formal

Inspección

Inspección por fases

Pruebas de *software*

A continuación, se realiza un resumen analítico de las técnicas seleccionadas en la descripción anterior, las cuales se ajustan al presente proyecto.

Técnica de inspección

Algunas organizaciones han empleado la técnica de inspección y evaluación las recopiladas por (Moreno et al., 2010) (Suryn, 2014), por considerarla superior a las demás. La técnica de inspección es similar a la de recorrido, aunque es más formal porque se basa en criterios previamente definidos para evaluar las características de calidad.

La realización de una inspección implica, además de los pasos previstos en la técnica de recorrido (planificación, preparación y reunión), un paso de presentación. En esta etapa, los desarrolladores realizan un tutorial, con una duración aproximada de treinta minutos, sobre el material a inspeccionar, destacando lo que debe ser analizado, facilitando así la preparación individual de los inspectores. En la preparación individual, los inspectores deberán evaluar el material de acuerdo con la lista de criterios previamente establecida. En la reunión, el narrador debe aspirar a ser objetivo, resumiendo cada sección del material. El moderador debe dirigir la reunión en un intento de mantener la objetividad y el cronograma. También debe asegurarse de que se tengan en cuenta todos los criterios establecidos. Al final de la reunión, la lista de criterios debería haber sido revisada minuciosamente y todas las preguntas acordadas. Al igual que con la técnica de recorrido, se debe tomar la decisión de aceptar o no el producto, definiendo si se debe o no realizar una reinspección. Los participantes de la reunión son los mismos previstos en la técnica de recorrido.

Las experiencias prueban que el uso de inspecciones contribuye significativamente a la mejora de la calidad del producto, así como a la reducción de costos relacionados con las pruebas, aunque no elimina la necesidad de realizarlas (Madou et al., 2009).

Técnica de inspección por fases

La inspección por fases (Moreno et al., 2010) es una variación de la técnica de inspección. Es una técnica disciplinada y rigurosa donde se pueden evaluar todos los productos generados durante el desarrollo. La evaluación del producto se realiza a través de una serie de evaluaciones parciales realizadas por fases. Cada evaluación parcial tiene como objetivo verificar si el producto tiene una o más propiedades. Los propiedades examinadas se organizan en la inspección escalonada, de modo que cada evaluación parcial pueda suponer la existencia de las propiedades verificadas o validadas en las evaluaciones parciales anteriores. Por lo tanto, la técnica de inspección por fases sirve para detectar errores, así como también para garantizar la calidad de acuerdo con los requisitos de calidad identificados al comienzo del proyecto.

En una inspección por fases, las evaluaciones parciales se pueden realizar con inspectores individuales o con múltiples inspectores. Una evaluación parcial, con inspectores individuales, es un proceso riguroso controlado por una lista de preguntas que debe responder el evaluador. Estas preguntas son métricas y/o indicadores del grado en que se lograron los atributos de calidad considerados en la evaluación y relevantes en ese momento. Se realiza una evaluación parcial, con múltiples inspectores, para analizar cuestiones subjetivas y donde se pretende, a través de una reunión, obtener el consenso de los evaluadores. El proceso de realizar una evaluación con múltiples inspectores consta de dos etapas: etapa inicial que consta de una evaluación individual donde cada inspector analiza el material preparándose para la segunda etapa que es la evaluación grupal. El procedimiento para realizar las reuniones de inspección es el tradicional.

Pruebas (*test*)

Las pruebas (*test* termino en idioma inglés) representan una actividad crítica para el aseguramiento de la calidad, ya que significan, de algún modo, una evaluación "visible" de la

calidad del producto que se está construyendo. La evaluación, al contrario de las técnicas anteriores, se realiza sometiendo el programa a ejecución.

De hecho, la prueba representa una etapa en el proceso de desarrollo donde hay un objetivo destructivo. Un buen caso de prueba es aquel que tiene una alta probabilidad de identificar errores y una prueba exitosa es aquella que resalta errores aún no identificados.

Las pruebas se pueden dividir en dos categorías: funcionales y no funcionales. Las pruebas funcionales se emplean cuando desea probar un programa nuevo, o un programa que ha sido modificado, para ver si produce el resultado correcto. Se refiere, por tanto, a la implementación de la función solicitada por el usuario. Sin embargo, incluso si la implementación de las funciones solicitadas por el usuario produce resultados correctos, no significa necesariamente que todos los requisitos del *software* estén satisfechos. Las pruebas no funcionales se refieren a la evaluación de que el *software* cumple con las obligaciones legales, se desempeña dentro de lo especificado, está escrito de acuerdo con las normas y estándares establecidos y otros requisitos solicitados por el usuario (Collins et al., 1994).

La aplicación de las pruebas (*testing* termino en idioma inglés), para que sea productiva, debe planificarse con antelación y realizarse de forma sistemática, realizando una evaluación progresiva del producto. Inicialmente, se deben realizar pruebas de cada módulo de forma individual para garantizar que funcionen correctamente como una unidad del programa (pruebas unitarias). A continuación, se realiza la prueba de integración de las unidades, verificando la composición de los módulos para la formación del programa en su conjunto. Una vez integrado el *software*, se realizan pruebas de validación para asegurar que el sistema construido ha cumplido con los requisitos establecidos. Finalmente, se realizan pruebas del sistema en las que se evalúa el sistema dentro de su contexto, combinado con otros elementos (como *hardware*,

base de datos, usuarios, entre otros.), verificando así que todos los elementos encajan correctamente y que se han cumplido todos los requisitos del sistema.

Prueba formal

La prueba de verificación formal consiste en tratar el programa como un objeto matemático que puede evaluarse junto con sus especificaciones (Tosun et al., 2017) (Munidhanalakshmi Kumbakonam & Shafi, 2021). Si la especificación de requisitos y el lenguaje de programación pueden representarse de manera rigurosa tanto sintáctica como semánticamente (Madou et al., 2009), es posible aplicar pruebas matemáticas de corrección para demostrar que el programa coincide con su especificación.

Los intentos de probar la corrección de los programas no son nuevos (McCall et al., 1977) (Sun, 1988) (Grogono et al., 1993). Esto obedece a que un programa es visto como una secuencia de instrucciones que implementa alguna función específica. En varios puntos de esta secuencia puede ser posible determinar, a partir de la especificación, el valor correcto de las variables del programa, el estado adecuado de la información de control y otras relaciones internas. Progresivamente se puede garantizar que la aplicación del programa a una entrada producirá la condición de salida especificada.

Según Pressman (1992), algunos investigadores de pruebas formales creen que los desarrolladores deberían aplicar el enfoque descrito anteriormente u otras variaciones de este, para diseñar e implementar módulos de programa. Sin embargo, otros consideran que la única esperanza de usar pruebas formales radica en el desarrollo de herramientas automatizadas.

Control Estadístico de Calidad de Software

El control de calidad estadístico refleja una tendencia actual en el mercado, el cual está orientado a volverse más cuantitativo sobre el factor de la calidad de *software*. Propone que para

realizar el control estadístico de la calidad se debe recopilar y clasificar la información sobre los defectos del *software* y atribuirlos a un conjunto de causas tales como:

Especificación incorrecta o incompleta

Mala comprensión en la comunicación con el usuario

Incumplimiento de las especificaciones

Pruebas incorrectas o incompletas

Módulo de interfaz inconsistente, entre otros.

A partir de lo expuesto se debe determinar el número total de defectos por cada causa relacionada y de este total deben ser considerados los datos para la construcción una matriz que contempla el número de defectos considerados graves, moderados o triviales.

Por lo tanto, los desarrolladores de *software* pueden calcular un índice de defectos (IDF) para cada fase en el proceso de desarrollo. De esta forma debe ser determinado el número total de defectos (TD), el número de defectos serios (DS), el número de defectos moderados (DM), el número de defectos triviales (DT), el tamaño del producto (TMP) el cual se considera a partir del total de líneas de código, instrucciones de diseño, páginas de documentación y la determinación de un peso para defectos graves, moderados y triviales (wj, j = 1 hasta 3). Luego se calcula el índice para cada fase (IDFi = wl (DS/TD) + w2 (DM/TD) + w3 (DT/TD)). El índice de defectos (ID) se calcula por el efecto acumulativo de cada IDFi, de la siguiente forma: IDF = ∑(i x IDFi)/TMP (M Kumbakonam, 2021) (Munidhanalakshmi Kumbakonam & Shafi, 2021) (Tosun et al., 2017) (Ceran et al., 2022) (Martín & Sáenz, 2016) (Madou et al., 2009). En conclusión, la tasa de defectos se puede calcular y almacenar en una tabla a partir de la información recopilada para determinar una indicación de las mejoras en la calidad del *software*.

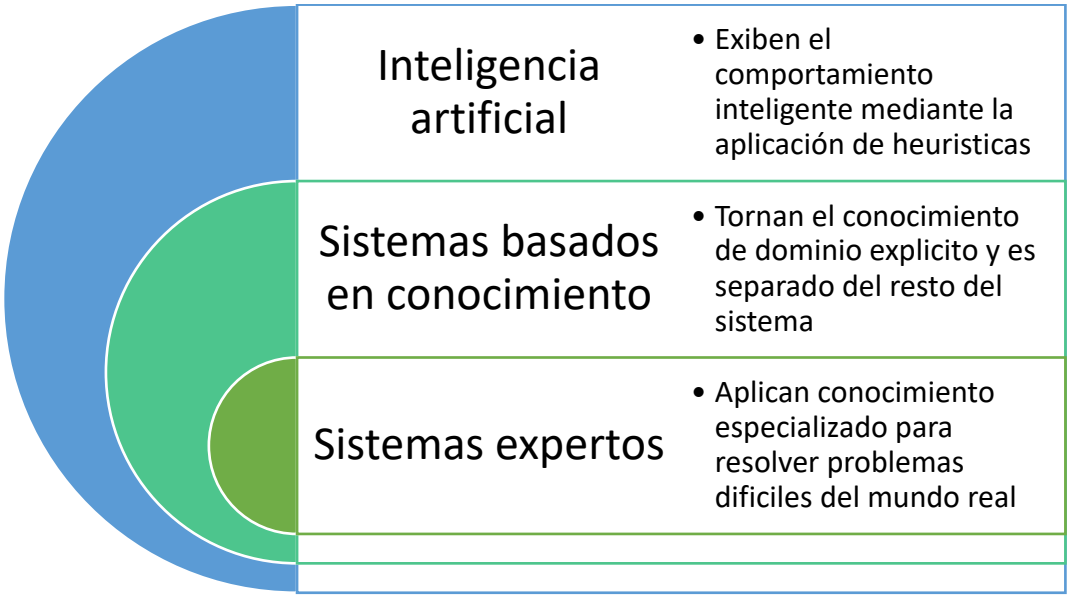
Sistemas Expertos

Los sistemas expertos representan un logro en el campo de la Inteligencia Artificial en la búsqueda de programas informáticos que resuelven problemas de una manera que se consideraría inteligente si los realizan expertos humanos. Los programas de Inteligencia Artificial exhiben un comportamiento inteligente mediante la aplicación de heurística en lugar del proceso algorítmico de los programas convencionales. Para que un programa sea inteligente, debe proporcionarse con alta calidad y conocimiento específico sobre un área del problema (Buchanan & Smith, 1988).

Según (Buchanan & Smith, 1988) el conocimiento necesario para el sistema se organiza separando el conocimiento sobre el dominio del problema y el conocimiento sobre la solución del problema. Los sistemas basados en el conocimiento organizan el conocimiento de esta manera. Por lo tanto, los sistemas expertos son sistemas basados en el conocimiento que aplican el conocimiento experto de un dominio de aplicación conforme se representa en la figura 3.

Figura 3

Adaptación del modelo de clasificación de los sistemas de inteligencia artificial



Nota. Elaboración propia

Buchanan & Smith (1988) define los sistemas expertos como programas encargados de resolver problemas de dominios especializados en los que la solución eficaz del problema requiere normalmente de un experto humano, considerando, por tanto, otro factor de los sistemas expertos: el experto.

El conocimiento del dominio del problema almacenado se denomina base de conocimiento y consiste en una colección de elementos de conocimiento. Cada elemento de conocimiento representa una unidad de conocimiento del dominio de la aplicación. Las relaciones entre estos elementos de conocimiento permiten deducciones que son las soluciones a problemas en el dominio de la aplicación. La implementación de la solución del problema se denomina motor de inferencia (Buchanan & Smith, 1988) (Martín & Sáenz, 2016) (Madou et al., 2009).

El motor de inferencia es un elemento esencial para la existencia dado que es la clave para el funcionamiento de un sistema experto por lo tanto se le considera como el núcleo del sistema (Martín & Sáenz, 2016). Este es quien interpreta la base de conocimientos para resolver problemas en el dominio de la aplicación. La forma en que se estructura depende de la naturaleza del dominio del problema y la forma en que se representa y organiza el conocimiento en el sistema experto. Los lenguajes más poderosos para construir sistemas expertos ya ofrecen el motor de inferencia incorporado, otros requieren que se implemente (Buchanan & Smith, 1988).

Es a través de él que los hechos, las reglas y la heurística que conforman la base de conocimiento se aplican en el proceso de resolución de problemas. La capacidad del motor de inferencia se basa en una combinación de procedimientos de pensamiento regresivo y progresivo.

En forma de razonamiento progresivo el usuario proporciona la información al sistema, quien, con sus respuestas, estimula el proceso de búsqueda, navega a través de la base de conocimiento, busca los hechos, las reglas y las heurísticas que mejor se aplican a cada situación. El sistema continúa esta interacción con el usuario, hasta que encuentra la solución al problema que se le envía.

En el modelo de razonamiento regresivo, los procedimientos de inferencia se invierten. El sistema parte de una opinión concluyente sobre el tema, incluso puede provenir del propio usuario e inicia una búsqueda de la información a través de las reglas y los hechos de la base de conocimiento, tratando de probar si esa conclusión es la solución más adecuada para el problema analizado.

La función de cualquier esquema de representación del conocimiento es capturar las características esenciales del dominio de un problema y producir información accesible a los procedimientos de resolución de problemas (Madou et al., 2009). En este sentido, se utilizan dos paradigmas de Inteligencia Artificial: el simbolista que considera la inteligencia solo en el comportamiento del individuo y considera que no existe inteligencia relacionada con la máquina y el conexionista que determina una estructura de red de elementos computacionales y las redes neuronales en las que se basa esta inteligencia.

El desarrollo de sistemas expertos normalmente utiliza el paradigma simbolista. Sin embargo, existen sistemas expertos híbridos que representan el conocimiento mediante el uso de dos tipos de paradigmas. La representación del conocimiento puede caracterizarse como conocimiento orientado a hechos u otro conocimiento que puede usarse en el razonamiento e implementarse mediante un lenguaje de programación (Buchanan & Smith, 1988). Los hechos son verdades relevantes para algún dominio del problema y son lo que uno quiere representar y

la representación de los hechos es un formalismo expresado en estructuras de conocimiento que podemos manipular.

Las estructuras de representación del conocimiento se pueden clasificar en cuatro categorías (Martín & Sáenz, 2016) (Madou et al., 2009) (Buchanan & Smith, 1988):

Lógica: utiliza expresiones de la lógica formal para representar el conocimiento y aplica reglas de inferencia y procedimientos de prueba formales a instancias de problemas, como el cálculo de predicados de primer orden.

Procedimental: utiliza un conjunto de instrucciones para resolver el problema para representar el conocimiento, como las reglas de producción

Redes: representan el conocimiento como un grafo en el que los nodos representan objetos o conceptos del dominio del problema y los arcos las relaciones entre ellos, como las redes semánticas

Estructurado: amplía el esquema de red al permitir que cada nodo sea una estructura de datos compleja, como marcos y objetos.

Características de los Sistemas Expertos

Los sistemas expertos se diferencian de los sistemas convencionales, por ejemplo, por su carácter no procedimental, por la existencia de una base de conocimiento y por su forma recursiva de desarrollo frente a la línea tradicional de desarrollo de los sistemas convencionales (Madou et al., 2009). Según (Buchanan & Smith, 1988), la diferencia básica entre estos sistemas es que los sistemas expertos manipulan el conocimiento mientras que los sistemas convencionales manipulan los datos.

Así mismo (Madou et al., 2009) se refiere a las diferencias entre los sistemas convencionales y los sistemas expertos según cuatro puntos de vista: el tipo de problema considerado, los métodos de solución empleados, los lenguajes y estilos de programación empleados y el uso efectivo de ambos sistemas. Lo anterior expresado en otras palabras expresa que la forma de caracterizar los sistemas expertos es comparándolos con otros sistemas. Se infiere que los sistemas expertos son sustancialmente diferentes de otros sistemas.

Según (Buchanan & Smith, 1988), la principal diferencia entre los sistemas expertos y los sistemas convencionales se refiere al tipo de problema considerado para cada uno de ellos. Los problemas tratados por los sistemas convencionales están bien definidos, permiten una clara descomposición en subpartes y funcionan con entradas bien definidas y completas, produciendo salidas precisas. Sin embargo, los sistemas expertos resuelven problemas resueltos por expertos humanos, al estar mal estructurados y tener que lidiar con información incompleta, incierta o incluso inconsistente. Además, un problema puede tener múltiples soluciones igualmente aceptables.

Estas diferencias se reflejan en el grado de precisión de la especificación del sistema. En los sistemas convencionales existe una traducción completa de los requisitos funcionales en sus

especificaciones, lo que no es posible en el caso de los sistemas expertos. En la práctica, resulta que en los sistemas expertos existen requisitos de usuario que no pueden expresarse en términos formales en un lenguaje de especificación. Aunque no es posible una traducción completa de los requisitos del usuario en especificaciones, se puede lograr una traducción parcial. Esto sugiere dividir los requisitos del usuario en dos clases: requisitos totalmente formalizables y requisitos parcialmente formalizables (Madou et al., 2009) (Flores Zafra & Gardi Melgarejo, 2020) (Martín & Sáenz, 2016) (Mendoza, Luis E, Pérez, 2005). Un requisito es completamente formalizable si se puede traducir completamente en especificaciones de como corregir la base de conocimiento. Así mismo, un requisito es parcialmente formalizable si no puede traducirse por completo en especificaciones. Por lo tanto, se requiere de una definición precisa de los conceptos implicados para que los requisitos formalizables puedan traducirse completamente en especificaciones. Los requisitos parcialmente formalizables son, por naturaleza, más difíciles de validar.

Otra diferencia importante entre los sistemas expertos y los sistemas convencionales es el enfoque de resolución de problemas. Los sistemas convencionales se resuelven mediante algoritmos que se puede demostrar que son correctos independientemente de cualquier implementación. Obviamente esto no garantiza la validez de la implementación ya que se pueden introducir errores cuando se codifica el algoritmo. En el caso de los sistemas expertos, los problemas se resuelven mediante asociaciones heurísticas que componen modelos que simulan el comportamiento de expertos humanos basados en experiencias personales para un problema de dominio. De esta forma, se pospone una validación efectiva hasta la fase de implementación.

La solución generada está directamente vinculada a la programación. Los programas en los sistemas convencionales se desarrollan utilizando lenguajes procedimentales mientras que los sistemas expertos utilizan lenguajes declarativos que son interpretados por motores de inferencia.

Los estilos de programación en los sistemas convencionales y los sistemas expertos son totalmente diferentes: los programas en los sistemas convencionales tratan con variables, condiciones, bucles y procedimientos; a diferencia de los sistemas expertos que tratan con valores de verdad, dependencias de reglas y asociaciones heurísticas.

Al analizar las diferencias en el uso efectivo de los sistemas expertos y convencionales, (Buchanan & Smith, 1988) considera dos aspectos: el nivel organizacional, que analiza el uso del *software* en la organización, y el nivel de usuario, que analiza cómo el usuario final trata con el *software*. Desde una perspectiva a nivel de organización, la tecnología de sistemas convencionales está completamente consolidada y es comercialmente aceptable. Sus aplicaciones son a menudo la única forma posible de procesar información. Los cambios provocados en la organización por estas aplicaciones se aceptan como una consecuencia natural del uso del *software*. Los sistemas expertos, por el contrario, generalmente no están integrados en el flujo de información de una organización, ya que son programas independientes para el apoyo a la toma de decisiones, advertencias o capacitación de principiantes. El impacto de sus aplicaciones en las organizaciones es aún un campo por investigar. Esta diferencia tiene mucho que ver con el nivel de madurez y el grado de confianza en los dos tipos de sistemas. Desde el punto de vista del usuario, hay dos diferencias. La primera se refiere a que mientras que las aplicaciones de sistemas convencionales suelen ser la única forma de procesar la información, las aplicaciones de sistemas expertos suelen ser opcionales y el usuario tiene la decisión de utilizar o no la solución del sistema experto. Esto requiere ganar la confianza del usuario con el sistema experto, que normalmente requiere que el sistema explique sus deducciones y recomendaciones de manera que el usuario las entienda. Otra diferencia se refiere a las responsabilidades legales en el uso del *software*. En el caso de los sistemas convencionales, el usuario no tiene responsabilidad por

fallas de desempeño en la ejecución del *software*, ya que las soluciones siguen protocolos bien definidos para el procesamiento de la información. En el caso de los sistemas expertos el usuario tiene toda la responsabilidad de sus decisiones, incluidos los errores inducidos por recomendaciones erróneas del sistema experto, lo que justifica aún más la necesidad del sistema experto de ganarse la confianza del usuario.

Además de las características asociadas a la forma de resolución de problemas, así como a las diferencias planteadas con los sistemas convencionales, también existen otros aspectos técnicos, así como ambientales y de diseño que distinguen completamente a los sistemas expertos de otros sistemas (Madou et al., 2009).

En los aspectos técnicos se incluye la característica de los sistemas expertos de procesar información simbólica (*no sólo numérica*) y así permitir resolver problemas no numéricos que por lo general son menos precisos que los numéricos, debido a que requieren un esquema de representación de conocimiento especial para hacerlo accesible (no es necesario en otros sistemas porque carecen de representación de conocimiento) y desarrollarse con lenguajes de Inteligencia Artificial que procesen información simbólica o utilizando herramientas de sistemas expertos (*shells*) que ya ofrecen una máquina de inferencia y facilitan la entrada de conocimiento.

Dos aspectos ambientales distinguen de una forma clara a los sistemas expertos de otros sistemas. El primero se refiere a la característica de los sistemas expertos de que influyen de forma directa en la decisión o incluso en el hecho de que toman decisiones. El segundo aspecto se refiere a la especialidad a ser modelada por el sistema experto, dado que esta característica suele darse en pequeñas cantidades, esto se debe a que es un recurso costoso y/o no está disponible en un módulo específico.

La metodología de diseño también distingue a los sistemas expertos de otros sistemas, ya que por lo general se desarrollan según la filosofía *middle-out* en lugar del enfoque de diseño tradicional de arriba hacia abajo (*top-down*) o de abajo hacia arriba (*bottom-up*). Además, como los sistemas expertos siempre están evolucionando, el proceso de toma de decisiones se comprende y modela gradualmente.

Verificación, Validación y evaluación en sistemas expertos

De acuerdo con la revisión literaria los términos verificación, validación y evaluación se confunden constantemente. En una revisión de la literatura publicada sobre validación (Grogono et al., 1993), encontraron que no existe una definición estándar para estos términos y que más de dieciséis términos asociados son utilizados de formas diferentes. Sun (1988) propone una terminología para los términos basada en el análisis de diferentes definiciones propuestas en la literatura. Según este autor, casi todos los autores han desarrollado sus propias terminologías.

Dada esta particularidad se emplearán los conceptos verificación, validación y evaluación en el sentido más tradicional, propuesto por Sun (1988) y también aceptado como terminología para el caso de los sistemas expertos según el análisis de Groundwater et al. (1995), en el que la verificación es la valoración de que el producto se está construyendo correctamente, mientras que la validación es la evaluación de que está creando el producto adecuado. En otras palabras, la verificación consiste en asegurar que el sistema implementa correctamente su especificación (Groundwater et al., 1995) y la validación consiste en probar si el sistema se desempeña con un nivel aceptable de precisión (Tucupa, 2020), con respecto a la tarea propuesta (Madou et al., 2009) según las necesidades y requerimientos del usuario de forma total o parcial (Sun, 1988).

McCall et al. (1977) define la verificación como el proceso de demostrar que el *software* tiene las características especificadas en su documentación (especificación), mientras que la

validación es la demostración de que el *software* tiene las características deseadas por el usuario final. Con eso, afirma que sin especificación no es posible realizar la verificación, pero sí la validación.

Problemas para la Verificación y Validación de Sistemas Expertos

Un obstáculo para la plena aceptación de los sistemas expertos es la falta de una metodología de verificación y validación que se ve obstaculizada por la falta de documentación estable y métodos poco adecuados para evaluar los resultados de las pruebas (Madou et al., 2009). Por lo tanto, son necesarios enfoques metodológicos y disciplinados, así como herramientas que apoyen estos enfoques (Hauge et al., 2006).

Sun (1988) señaló la existencia de un círculo vicioso dentro de las organizaciones que impedía el desarrollo de métodos para evaluar la calidad de los sistemas expertos: la evaluación de los sistemas expertos no se hacía porque nadie preguntaba, nadie preguntaba porque nadie sabía qué ni cómo evaluar y nadie sabía evaluar porque nunca nadie había hecho evaluaciones. Este mismo autor sugirió que para romper este ciclo y avanzar hacia una metodología de evaluación efectiva, es necesario experimentar. El estado actual del arte muestra que se ha intentado mucho para lograr esta metodología (Grogono et al., 1993) (Garousi & Mäntylä, 2016) (Hauge et al., 2006).

De hecho, el estado del arte muestra que aún no existe una metodología universalmente aceptada para la verificación y validación de sistemas expertos debido a varios problemas específicos. Uno de los principales problemas es que los requisitos de los sistemas expertos son muy difíciles de determinar (Grogono et al., 1993), además de ser imprecisos o cambiar rápidamente (Tucupa, 2020). Dado que los sistemas expertos se construyen mediante

refinamientos e interacción con expertos, los requisitos se modifican y a veces se registran cada vez menos.

Las técnicas convencionales de seguimiento de la ejecución mediante el examen del código para verificar si el sistema implementado satisface los requisitos es muy difícil de llevar a cabo (Madou et al., 2009) (Martín & Sáenz, 2016). La secuencia de ejecución de un sistema experto a partir del examen de la base de conocimiento puede ser extremadamente difícil de determinar, incluso con las técnicas tradicionales de prueba basadas en comparaciones de precondiciones/postcondiciones (Groundwater et al., 1995). La capacidad del sistema experto para producir sus salidas es una propiedad de la interacción entre la base de conocimientos y el motor de inferencia y solo puede analizarse cuando se ejecuta el sistema.

Para analizar los requisitos del sistema en diseño y código, es necesario identificar unidades de función en varios niveles. Un diseño modular, de arriba hacia abajo y descompuesto jerárquicamente puede no ser posible en algunas arquitecturas de sistemas expertos. Usando reglas es difícil tener una estructura modular, aunque diferentes reglas se ocupan de diferentes casos. La orientación a objetos ofrece una forma de lograr cierta modularidad, pero no hay un patrón de diseño definido (Wedyan & Abufakher, 2020). Otro problema es que sistemas expertos que trabajan con incertidumbre o datos incompletos pueden tener muchos estados posibles, lo que hace impracticable la aplicación de pruebas exhaustivas (Groundwater et al., 1995).

Además de todas estas dificultades, no existen métodos para evaluar resultados de pruebas de sistemas expertos aceptables y confiables. El enfoque de tener expertos en el dominio de la aplicación tiene algunos inconvenientes. Es posible que los expertos no estén disponibles o no sean independientes cuando se necesita una evaluación independiente (Groundwater et al., 1995).

Características de calidad para sistemas expertos

Varios autores identifican características de calidad para ser evaluadas en sistemas expertos. (Moreno et al., 2010)utilizando el concepto de evaluación basado en la definición de calidad de ISO, determinó un conjunto de características de calidad según dos aspectos: técnico y social.

Los aspectos técnicos de un sistema comprenden la evaluación del sistema como un producto tecnológico en términos de la tarea que realiza el sistema y la tecnología a través de la cual se implementa el sistema (Hauge et al., 2006). En cuanto a la tarea son consideradas las características de la tarea elegida para la aplicación de sistemas expertos (dificultad, escasez y utilidad de la tarea), la idoneidad de la tarea para sistemas expertos (confianza, realismo y confianza) y la reelaboración del trabajo (en lo que respecta a la innovación y simplificación del trabajo con sistemas expertos). En cuanto a tecnología, criterios de entrada (interacción y codificación), procesamiento (como velocidad, inferencia y explicación), salida (en cuanto a precisión, especificidad y presentación) e interfaz (amigable, fácil de cambiar y compatible con otros sistemas).

Los aspectos sociales se refieren a las personas involucradas con el sistema y el contexto de la organización. Con respecto a las personas Suryn (2014) considera factores relacionados con el impacto del sistema experto en sus usuarios como estimulación y reducción de trabajo y factores relacionados con las personas durante el proceso de desarrollo del sistema como entusiasmo, calificación del personal y participación de ayuda de los expertos. En cuanto a la organización, analiza las características organizativas como burocracia, formación y actualización de aprendizajes y los beneficios económicos que pueden derivarse de los sistemas expertos rentabilidad y aumento de la competitividad en el mercado (Basili & Musa, 2002).

Winkler et al. (2021) considera el enfoque de Karg et al. (2011) sobre la validación técnica y social de los sistemas expertos, aunque considera como aspectos sociales las características de idoneidad de la tarea para los sistemas expertos y la reelaboración del trabajo. En cuanto a la verificación de sistemas expertos, este autor considera aspectos conductuales y ontológicos de estos sistemas. Los aspectos de comportamiento se refieren al desempeño externo esperado cuando el sistema está en operación, por lo tanto, se debe verificar qué puede hacer el sistema en su alcance, cómo se desempeña el sistema en su corrección, cómo es su comportamiento en su robustez, transparencia, facilidad de aprendizaje y operación, eficiencia de tiempo y recursos y la confiabilidad del sistema. Los aspectos ontológicos se refieren a la estructura interna de la organización y componentes que producen el comportamiento observable y la estructura del proyecto como la estructura de representación del conocimiento y los algoritmos de razonamiento debe ser verificada en cuanto, por ejemplo, a la idoneidad y especificidad; los componentes del *software* con respecto a la corrección, mantenibilidad, facilidad de prueba, reutilización, portabilidad y compatibilidad con otros sistemas; y finalmente, el contenido de la base de conocimiento en cuanto a consistencia, concisión, validez, completitud y mantenibilidad (Madou et al., 2009).

La verificación de la base de conocimiento se considera de gran importancia en la literatura, siendo abordada por varios autores (Buchanan & Smith, 1988) (Madou et al., 2009) (Grogono et al., 1993) (Guerra et al., 2014) (Ceran et al., 2022) (Martín & Sáenz, 2016). Las características más discutidas en este sentido consisten en verificar la consistencia e integridad de la base de conocimiento.

Verificar la consistencia implica analizar discrepancias, ambigüedades y redundancias en el conjunto de reglas en la base de conocimiento (Grogono et al., 1993). Para garantizar la

consistencia de la base de conocimiento, se debe verificar la ausencia de condiciones redundantes, conflictivas, incluidas, circulares, innecesarias y valores de atributos ilegales (Buchanan & Smith, 1988) (Madou et al., 2009) (Grogono et al., 1993) (Guerra et al., 2014) (Ceran et al., 2022) (Martín & Sáenz, 2016). Debido a la importancia de estos aspectos, se definen algunas cada una de las características identificadas y se emplea la notación de cálculo de predicados para las definiciones formales, siendo *x, y & z* las variables y *p, q & r* las relaciones lógicas (Meseguer, 1991) como ejemplo práctico de comprensión:

Reglas redundantes: dos reglas son redundantes si ocurren en la misma situación y tienen la misma conclusión. Formalmente en notación de cálculo de predicados decimos que la regla p(x) -> q(x) es equivalente a la regla p(y) -> q(y)

Reglas en conflicto: dos reglas están en conflicto si ocurren en la misma situación, pero tienen conclusiones contradictorias o contradictorio. Formalmente en notación de cálculo de predicados decimos que la regla p(x) -> *not*(q(x)) es contradictoria con la regla p(x) -> q(x)

Reglas incluidas: Una regla está incluida en otra si ambas reglas tienen la misma conclusión, pero una contiene condiciones adicionales. Cuando ocurre la regla más restrictiva, también ocurre la menos restrictiva, lo que implica redundancia (Meseguer, 1991). Formalmente en notación de cálculo de predicados decimos que la regla (p(x) *and* q(y)) ->r(z) está incluida en la regla a p(x) -> r(z);

Condiciones innecesarias: dos reglas contienen condiciones (*if*) innecesarias si ellas tienen la misma conclusión, y una condición en una regla entra en conflicto con una condición en la otra regla, siendo que todas las demás condiciones en las dos reglas son equivalentes. Formalmente en notación de cálculo de predicados decimos que si tenemos las reglas ((p(x) *and*

q(y)) -> r(z) & (p(x) *and not*(q(y))) -> r(z) la condición que implica q(y) en cada regla es innecesaria

Reglas circulares: un conjunto de reglas son circulares si la cadena de estas reglas forman un ciclo, presentado formalmente en la notación de cálculo de predicados cuando tenemos un conjunto p(x) -> q(x), q(x) -> r(x) & r(x) -> p(x) o una regla aislada del tipo p(x) -> p(x);

Valores de atributo ilegales: Ocurre en reglas que hacen referencia a valores de atributo que no están en el conjunto de valores posibles para ese atributo. Rothenberg et al. (1987)considera esta característica para verificar la integridad.

Completitud significa que la base de conocimientos está preparada para responder a todas las situaciones posibles que puedan surgir dentro del dominio de su problema (Rothenberg et al., 1987). Para garantizar la integridad de la base de conocimientos, debe existir la ausencia de atributos no referenciados, conclusiones inalcanzables, callejones sin salida y partes del dominio no cubiertas denominadas como ausencia de conocimiento (Meseguer, 1991). A continuación, definimos cada una de estas características:

Atributos no referenciados: un valor de atributo no referenciado ocurre cuando algún valor de un posible conjunto de valores de un objeto no es utilizado por ninguna condición de ninguna regla

Conclusiones inalcanzables: conclusiones que nunca se deducen, ni se pueden cuestionar, en otras palabras, toda conclusión de una regla es la conclusión (objetivo del sistema) o debe ser una premisa de otra regla, si no la hay. estas situaciones a una conclusión, entonces se dice que es inalcanzable (Madou et al., 2009) (Buchanan & Smith, 1988).

Callejón sin salida: decimos que hay un objetivo "callejón sin salida" cuando los atributos necesarios para determinarlo no son consultados por el usuario ni obtenidos de una regla en el conjunto de reglas

Porciones de dominio no cubiertas: se refiere a la ausencia de algún elemento esencial del dominio del problema en la base de conocimiento de (Buchanan & Smith, 1988), es decir, hay una situación en la que se solicita un resultado específico, pero no se puede activar ninguna regla de base de conocimiento en esta situación de una manera que produzca el resultado deseado (Madou et al., 2009).

Verificación de los sistemas expertos

En general, el desarrollo de *software* genera productos intermedios entre los requisitos iniciales del proyecto y el producto final. Estos productos van desde representaciones abstractas hasta representaciones más concretas que se traducen y observan en el código final. El proceso de verificación implica garantizar que cada producto generado a lo largo del desarrollo sea la expresión correcta de lo que se desea en cuanto a la implementación de todos los requisitos y se produzca correctamente (Meseguer, 1991).

Mucho se ha discutido sobre el problema de la verificación. Por ejemplo, Moreno et al. (2010) (Gao et al., 2019) (Collins et al., 1994) resume varios de los enfoques, definiendo las cuestiones relacionadas con qué comprobar, comprobar qué comprobar y con qué comprobar.

Dos componentes definen la primera pregunta sobre qué verificar: la base de conocimiento y el motor de inferencia. En la actualidad, la gran mayoría de los sistemas expertos se construyen utilizando herramientas (*shells*) que ya brindan el mecanismo de inferencia, por lo que no es necesario verificarlo. Los motores de inferencia personalizados para el sistema que se está desarrollando pueden tener fallas y efectos indeterminables en el procesamiento, incluso si

la base de conocimiento es perfecta. Sin embargo, la base de conocimiento es el componente crucial para verificar los sistemas expertos (Meseguer, 1991).

Con respecto a qué verificar, este autor afirma que la verificación presupone la existencia de una especificación. Es cierto que a menudo no es posible una especificación completa del sistema, pero de forma general se necesitan muchas propiedades deseables que se pueden usar como especificaciones parciales o incompletas. Así mismo el autor considera que este metaconocimiento sobre la base de conocimiento es vital y necesario para la verificación.

La última pregunta sobre qué herramientas y técnicas se pueden utilizar para verificar la base de conocimiento no tiene una respuesta única y bien definida. Se han desarrollado muchas herramientas para aplicaciones específicas de sistemas expertos que utilizan lenguajes específicos. En cuanto a la justificación de construir estas herramientas que simplemente ayudan a depurar/verificar bases de conocimiento, Meseguer (1991) & (Rothenberg et al., 1987) dice que pueden identificar rápidamente problemas en la base de conocimiento y posiblemente permitir que los expertos descubran lagunas en su conocimiento o errores en su razonamiento.

La literatura presenta varios enfoques para verificar las bases de conocimiento. Muchos autores proponen herramientas para estos enfoques y aplican sus ideas en proyectos reales. Moreno et al. (2010) realizó una colección de enfoques y proyectos en los que se aplicaron verificaciones e identificó los siguientes enfoques: metaconocimiento, tablas, tablas de decisión, deducción, valores de confianza, matrices, gráficos, redes, proceso analítico jerárquico, verificación temprana en el ciclo de vida, verificación a través de la ejecución.

Herramientas para Verificación de Sistemas Expertos

A continuación, se describen algunos de los enfoques presentando las herramientas empleadas.

Rothenberg et al. (1987) propuso un enfoque para comprobar la coherencia y la redundancia de la base de conocimientos: la reducción de la base de conocimientos. Según este autor, esta técnica en principio puede detectar todas las posibles contradicciones y redundancias que existen en las bases de conocimiento. Reductor basado en conocimiento (*KB-Reducer* por su nombre en inglés) es un sistema que implementa una versión especial de la técnica de reducción de la base de conocimiento. Se basa en un modelo de inferencia abstracta que enfatiza la relación entre las formas de certeza del razonamiento del sistema experto y la deducción natural en la lógica proposicional, analizando la base de conocimiento escrita en un lenguaje de representación de reglas canónicas. Así mismo el autor admite que la reducción de la base de conocimientos es un primer paso en un procedimiento para proporcionar a los sistemas expertos la capacidad de modificar automáticamente su base de conocimientos para adaptarse a diferentes entornos.

Herramienta de Verificación basada en la Unificación (*Unification-based Verification Tool* – UVT por sus siglas en inglés) es otra herramienta para la verificación de la base de conocimiento descrita por Rothenberg et al. (1987). Las reglas de la base de conocimiento con predicados comunes se pueden relacionar. Durante la verificación UVT compara los literales para determinar las relaciones entre ellos. Estos predicados comunes pueden ser equivalentes, aunque no exactamente iguales. Los predicados de conjunción pueden estar en cualquier orden, aunque podría haber alguna restricción impuesta por la lista de sustitución. Para identificar la equivalencia de predicados comunes se utiliza la técnica de unificación. El programa comienza analizando todas las reglas inferidas y agregándolas a una base de reglas. El programa asume que las reglas están en un formato establecido. El segundo paso es construir dos tablas a partir de las relaciones entre las reglas. Una tabla contiene la información relacional obtenida al comparar los

parámetros (consecuente y antecedente) de pares de reglas usando la unificación. La segunda tabla se utiliza para detectar posibles reglas circulares. A partir de estas tablas, UVT verifica la base de conocimiento para los requisitos de integridad y consistencia.

Existe otro enfoque para verificar la base de conocimiento es el uso de redes de Petri, utilizadas en la herramienta *PREPARE* propuesta por Meseguer (1991) para verificar reglas inconsistentes, redundantes, incluidas, circulares e incompletas en la base de conocimiento. *PREPARE* se basa en el modelado de la base de conocimientos en un tipo especial de red de Petri denominada redes de Predicado/Transición (Pr/T), que son representaciones gráficas de la lógica de predicado de primer orden. En *PREPARE*, las redes Pr/T se utilizan para describir las relaciones entre una regla y las diferentes reglas y hechos en la base de conocimientos. A partir de ahí, la verificación de la base se realiza mediante la identificación de patrones de estructuras de subred en la red Pr/T. Para cada característica a comprobar, se define un patrón de estructura. *PREPARE*, por lo tanto, contiene cuatro componentes: un transformador, que traduce las reglas y los hechos de la base de conocimientos en una representación de la red Pr/T; un verificador que detecta y ubica los límites de patrones inconsistentes, redundantes, circulares e incompletos; un formulador que codifica y formula un texto para cada patrón circular inconsistente/redundante y un clasificador, que determina el tipo de patrón reconociendo el texto producido por el formulador y descubre el tipo de incompletitud analizando los resultados obtenidos por el verificador.

Validación de Sistemas Expertos

La validación del sistema experto es más que un simple proceso destinado a encontrar errores. Es el proceso de determinar que un sistema experto representa el conocimiento de un experto. Esto significa que el proceso de validación implica investigar qué sabe el sistema, qué

no sabe o sabe incorrectamente; en el análisis del nivel de experticia en la toma de decisiones del sistema; en determinar si el sistema experto está basado en la teoría y en analizar su confiabilidad (Rothenberg et al., 1987).

Algunas preguntas sobre el proceso de validación se abordan constantemente en la literatura en referencia a qué validar, cuándo validar, cómo validar, cómo controlar el costo de la validación, cómo controlar el sesgo contra la validación y cómo lidiar con resultados múltiples (Rothenberg et al., 1987) (Meseguer, 1991).

Groundwater et al. (1995) responde a estas preguntas pensando que debemos validar cualquier resultado intermedio, el resultado final, el razonamiento del sistema o cualquier combinación de estos tres. Por supuesto, esto implica validar el proceso de razonamiento, porque un mal proceso con resultados correctos no puede ser adecuado para un dominio mayor. Por lo general, debemos validar el proceso de razonamiento lo antes posible y el resultado final cuando la base esté más completa. Para Meseguer (1991) la validación de procesos requiere la elaboración del proceso de razonamiento dinámico en términos de algunas características, tales como modelos de causas, dependencias secuenciales entre inferencias, tareas ordenadas o dependencias de metas.

El qué validar está intrínsecamente relacionado con la etapa de desarrollo que se esté realizando, es decir, con el momento en que se esté realizando la validación. Varios autores incluyen el proceso de validación como una actividad adicional en el proceso de desarrollo del sistema (Meseguer, 1991) (Groundwater et al., 1995) (Madou et al., 2009) (Guerra et al., 2014).

En cuanto a cómo validar (Meseguer, 1991) propone que los sistemas expertos sean validados tanto en términos de resultados conocidos como en términos de opinión de expertos. Además, los sistemas expertos deben validarse utilizando varios casos, previamente

documentados, que representen el trabajo de muchos expertos sobre un conjunto completo de problemas. Una validación más detallada requerirá probar el sistema contra una pequeña cantidad de casos complejos y oscuros. En estos casos, lo que importa no es el número de pruebas, sino el alcance que cubren las pruebas.

Es difícil determinar el esfuerzo y el costo necesarios para realizar la validación del sistema. El costo debe controlarse mediante el diseño de métodos de validación formales que se integren en el proceso de desarrollo. El valor de la validación, sin embargo, está directamente relacionado con el valor del sistema para sus usuarios y el riesgo que implica utilizar un sistema poco fiable.

Los sistemas expertos a veces se validan con expertos que sienten que su experiencia se ve amenazada en comparación con otros expertos o el sistema mismo. Con eso hay prejuicio, o prevención contra la actividad de validar los sistemas, dificultando esta actividad (Meseguer, 1991). Este problema se soluciona utilizando técnicas de validación que omiten la identidad de los evaluadores y que utilizan en la validación expertos que no han participado en el desarrollo del sistema (Rothenberg et al., 1987).

Cuando validamos un sistema experto con múltiples resultados, existe la dificultad relacionada con un problema con múltiples respuestas (por ejemplo, en los sistemas de prescripción médica para el tratamiento de una enfermedad, dos tipos de medicamentos son válidos si se consideran por separado, pero su combinación no es válida). aceptable. Por lo tanto, el conjunto de respuestas del sistema no es válido). En estas situaciones, no podemos probar la validez de las distintas respuestas del sistema experto validando cada respuesta por separado. Se necesita un enfoque múltiple, para incorporar la correlación entre los diversos resultados para validar el sistema en su conjunto.

Rothenberg et al. (1987) considera que la validación puede ser formal o informal. Una validación formal establece cuándo debe ocurrir la validación dentro del ciclo de desarrollo e identifica los métodos de validación, la entrada de especificación del dominio y, cuando corresponda, la relevancia de las técnicas estadísticas. La validación informal, que normalmente se realiza tarde en el desarrollo, emplea métodos ad-hoc y, a menudo, es un análisis subjetivo.

De acuerdo con Suryn (2014), la validación difiere en cada situación en términos de formalidad y la medida en que se implementa la validación. En la literatura técnica existe un conjunto de propuestas para validar cualitativa y cuantitativamente un sistema experto (Madou et al., 2009; Meseguer, 1991; Rothenberg et al., 1987).

Validación Cualitativa

La validación cualitativa emplea la comparación subjetiva, lo que no significa que sean validaciones informales. Encontramos en la literatura los siguientes ejemplos de validaciones cualitativas: validación cara a cara, validación predictiva, prueba de campo, validación por subsistemas, análisis de sensibilidad, interacción visual y prueba de Turing (Madou et al., 2009; Meseguer, 1991).

La propuesta de validación presencial consiste en evaluar la consistencia entre la visión de los desarrolladores y la visión de los expertos, con el fin de comparar el desempeño del sistema con el desempeño del experto humano. En este enfoque, también pueden participar otras personas con conocimiento del dominio de la aplicación (Meseguer, 1991) (Moreno et al., 2010). Según Rothenberg et al. (1987), esta técnica funciona como un mecanismo de retroalimentación para que el desarrollador refine la base de conocimiento, rediseñe y reformule etapas del proceso de desarrollo, finalizándose cuando desarrolladores, especialistas y otros evaluadores acuerdan que el prototipo evaluado refleja adecuadamente el problema y está bien estructurado.

La validación predictiva requiere la existencia de casos históricos para pruebas y resultados conocidos u obtenidos de expertos humanos. La validación ocurre con la comparación de los resultados obtenidos del sistema con los resultados conocidos para los casos de prueba históricos. Esta comparación verifica si el sistema tiene la precisión satisfactoria deseada (M Kumbakonam, 2021) (Tosun et al., 2017).

La prueba de campo es la validación del sistema en el entorno en el que será operado. Desde el punto de vista de los desarrolladores, este tipo de pruebas ofrece dos ventajas. En primer lugar, traslada la tarea de realizar las pruebas a los usuarios del sistema. En segundo lugar, la aceptación del rendimiento del sistema se gana implícitamente cuando los usuarios no tienen más quejas sobre el sistema para informar. Este tipo de validación solo es posible en aplicaciones no críticas, donde los usuarios pueden evaluar la corrección del sistema experto sin riesgo (Rothenberg et al., 1987).

La validación por subsistemas requiere que el sistema experto se descomponga en subsistemas a validar (Meseguer, 1991). En este enfoque, los subsistemas se validan a medida que se desarrollan. Este tipo de validación tiene tres ventajas: incorpora la actividad de validación dentro del ciclo de desarrollo, es más fácil de realizar ya que valida subsistemas menos complejos y más manejables que un sistema completo y facilita la detección de errores. Sin embargo, puede que no sea posible observar el comportamiento de entrada-salida de un subsistema y la validación exitosa de cada subsistema no implica la validez de todo el sistema, ya que las tolerancias de pequeños errores acumulados pueden ser significativas en el rendimiento del sistema final. Por lo tanto, al igual que con la validación cara a cara, el enfoque de este enfoque está en los detalles del prototipo que se está evaluando y sirve para identificar

áreas en el prototipo que requieren un desarrollo o revisión más detallados (Rothenberg et al., 1987).

El enfoque denominado análisis de sensibilidad consiste en cambiar los valores de las variables y parámetros de entrada según alguna variación de interés para observar los efectos que produce el sistema. Es decir, consideremos un sistema que produce un resultado satisfactorio para un caso de prueba X que usa n entradas. El análisis de sensibilidad consiste en cambiar cada una de estas n entradas y analizar el efecto sobre el resultado que ofrece el sistema. Este tipo de validación es muy útil para sistemas que trabajan con incertidumbre, ya que se pueden cambiar cuando se desee y se puede examinar el efecto de las medidas de incertidumbre intermedias y finales (Madou et al., 2009; Meseguer, 1991).

La validación a través de la interacción visual proporciona una animación visual del funcionamiento del sistema experto y permite que el experto interactúe cambiando los parámetros cuando lo desee. La interacción visual puede verse como un entorno para la validación interactiva cara a cara, la validación de subsistemas interactivos y el análisis de sensibilidad interactivo. Este tipo de validación ha sido ampliamente utilizado en la validación de modelos de investigación operativa (Rothenberg et al., 1987).

La prueba de Turing (McCall et al., 1977; Meseguer, 1991; Moreno et al., 2010; Rothenberg et al., 1987) consiste en validar el sistema experto comparando su desempeño con el de expertos humanos. Esta evaluación la realizan expertos sin que éstos tengan conocimiento de quién (sistema o expertos) está siendo evaluado. Proponen que los especialistas que participan en el test de Turing no deben haber participado en el desarrollo del sistema. Inicialmente, un grupo de expertos (al que se llamara G1) genera un conjunto de casos de prueba (casos generales de su experiencia como experto y casos interesantes que intentan hacer fallar el sistema). Estos casos

son enviados a otro grupo de especialistas (G2) y al propio sistema de especialistas para generar resultados. El grupo inicial (G1) organiza los resultados obtenidos suprimiendo la identidad de los elementos de diagnóstico (especialistas humanos y sistema experto), teniendo así un papel neutral en el proceso de evaluación. Así mismo el autor sugiere organizar los resultados en dos estructuras: una con solo los resultados finales para cada caso y otra con toda la información considerada y rechazada en el diagnóstico. Finalmente, los casos y conclusiones se someten a un grupo de especialistas que deben atribuir un grado de desempeño a las respuestas generadas, desconociendo cuáles de ellas fueron generadas por el especialista humano y cuáles por el sistema especialista. Meseguer (1991) sugiere que este grupo es diferente de los dos grupos que participaron en el proceso en las primeras fases, mientras que (Rothenberg et al., 1987) sugiere que es el propio grupo G2. El autor también sugiere, con su enfoque de ubicar los resultados en dos estructuras, que primero se someta a evaluación la estructura con solo los resultados finales para que la precisión del diagnóstico se haga sin la base de los elementos percibidos por los elementos de diagnóstico expresados en el otra estructura más completa.

Validación Cuantitativa

La validación cuantitativa emplea técnicas estadísticas para comparar el rendimiento del sistema experto con casos de prueba o expertos humanos. Los métodos de validación cuantitativa se dividen en dos categorías. En el primero, se define un intervalo de codificación para una o más medidas que se comparan subjetivamente con una variación de rendimiento aceptable. El segundo utiliza una prueba de hipótesis para comparar medidas con un valor predeterminado de desempeño aceptable, determinando si el sistema es válido o no (Meseguer, 1991).

Hay muchos métodos cuantitativos en la literatura. Collins et al. (1994) analiza tres de ellos: la prueba t pareada, la prueba de una muestra y los intervalos de confianza simultáneos. La

prueba t pareada es adecuada para comparar las diferencias entre los resultados obtenidos y se puede utilizar para medir las diferencias entre el sistema experto y los expertos humanos. La prueba de una muestra es adecuada para validar sistemas que producen múltiples resultados, proporcionando correlación entre ellos. Los intervalos de confianza simultáneos también se utilizan para sistemas expertos con múltiples respuestas, definiendo intervalos de confianza o uniendo regiones de confianza para diferencias en pares de respuestas.

Herramientas para validación de Sistemas Expertos

La validación de sistemas expertos ha sido ampliamente discutida, en el sentido de buscar metodologías y herramientas que reduzcan el esfuerzo necesario para validar estos sistemas. A continuación, reportaremos algunas herramientas presentadas en la literatura.

El uso de prototipos en el desarrollo de sistemas expertos ha tenido éxito. El proceso de validación sobre los prototipos de demostración implica un mayor detalle del problema y garantiza el desarrollo del sistema experto deseado. Además, es la validación sucesiva de prototipos lo que posibilita una base de conocimiento completa.

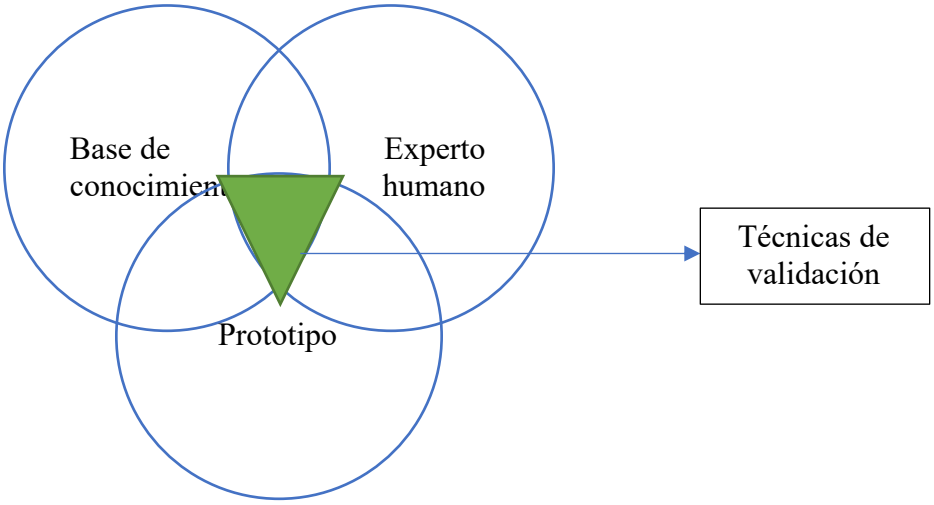
Muchos autores valoran el prototipo inicial o de demostración. Madou et al. (2009) en la metodología de verificación y validación que propone, considera el desarrollo inicial de un prototipo como base para establecer los requisitos del sistema, sin el cual no es posible realizar ninguna evaluación. Aun sabiendo que estos prototipos pueden estar incompletos o que, poco a poco, se van perdiendo, considera que su construcción da una idea clara del problema desde la perspectiva del ingeniero del conocimiento y que esta idea permite la preparación para la validación en el experto en sistemas exactamente lo que va a hacer.

Meseguer (1991) & Rothenberg et al. (1987) ubica a los prototipos como un componente de interacción que relaciona los procesos de evaluación como se muestra en la figura 4. A partir

de esta interacción entre especialistas, prototipos e ingenieros del conocimiento (desarrolladores y evaluadores) durante el proceso de validación, a medida que los participantes encuentran inconsistencias o limitaciones inaceptables en el prototipo, realizan reformulaciones, rediseños, afinan y corrigen las tareas. De esta forma, la validación se convierte en el motor de la evolución desde un prototipo inicial hasta el propio sistema.

Figura 4

Adaptación del paradigma de validación y sus interacciones



Nota. Elaboración propia

Según Rothenberg et al. (1987) la descripción de un prototipo es la base ideal para los requisitos, porque el prototipo es un programa de computadora real y evaluable, y cada prototipo generado proporciona un conjunto parcial de requisitos para construir el siguiente prototipo.

Meseguer (1991) describió un método y una herramienta desarrollada para probar y validar bases de conocimiento (Entorno de Prueba y Validación). El método se basa en los siguientes supuestos: las bases de conocimiento tienen carácter declarativo y representan información estructural sobre algún dominio de aplicación; las condiciones de integridad, coherencia y corrección pueden representarse uniformemente mediante gráficos y verificarse

mediante algoritmos sobre gráficos. El entorno interactivo de prueba y validación (TVIE por sus siglas en inglés) es independiente del entorno que especifica la base de conocimiento para que pueda adaptarse a diferentes problemas y lenguajes de representación. TVE recibe la base de conocimiento a probar, que se mapea en gráficos a través de un componente llamado *mapper*. A partir de ahí, ejecuta un conjunto de algoritmos para gráficos, generando la salida de errores y problemas para el usuario. Este proceso de análisis es interactivo y está respaldado por gráficos. Con base en esta experiencia, los autores se dieron cuenta de que el resultado de la evaluación depende del enfoque de programación de inteligencia artificial utilizado y que el carácter declarativo de la base de conocimiento permite la aplicación de técnicas automáticas de validación y pruebas que pueden mejorar significativamente la seguridad y confiabilidad de *software* de gran tamaño.

A partir de la observación de que las bases de conocimiento, que representan el conocimiento adquirido de los especialistas, deben ser validadas contra la propia fuente de conocimiento. Meseguer (1991) implementó una herramienta para la validación del conocimiento a través de casos. La herramienta *Knowledge Validation Tool* (KVAT por sus siglas en inglés) valida objetos de conocimiento representados en líneas o proposiciones, utilizando casos definidos por expertos. La herramienta proporciona un conjunto de funciones que permiten inspeccionar, registrar y eliminar casos de validación con descripciones, comparar la solución de la base de conocimiento con las soluciones de los expertos e incluir o excluir proposiciones de la base de conocimiento a través de un editor.

La herramienta *Expert System Validation Associate* (EVA por sus siglas en inglés) (Rothenberg et al., 1987) fue desarrollada con el objetivo de mejorar el proceso de validación mediante la búsqueda de errores y omisiones en la base de conocimiento. EVA fue escrito en

LISP y ART (herramienta de razonamiento automatizado) para sistemas expertos basados en ART y en realidad comprende la verificación y validación de sistemas expertos que combinan análisis estático y dinámico. EVA interactúa con la herramienta de sistemas expertos a través de una interfaz específica. Luego, los algoritmos de conversión traducen reglas, hechos y esquemas de un objeto de herramienta al formato de herramienta EVA. Las reglas, los hechos y el esquema de una aplicación de sistemas expertos se verifican en busca de errores estructurales, lógicos y semánticos. Tanto la verificación estructural como la lógica constan de dos pasos. El primero se realiza utilizando algoritmos especiales sin información semántica. En la segunda, denominada verificación extendida, utiliza la información semántica contenida en una base de metaconocimiento. La validación semántica se realiza a través de meta-reglas que operan sobre otras meta-reglas, meta-hechos, reglas y hechos de objetos. EVA aumenta la fiabilidad del sistema experto, acelera su desarrollo y ayuda en el proceso de modificación continua.

Los sistemas expertos por analizar están escritos en el entorno de desarrollo de sistemas expertos (ESPE) de IBM. El sistema experto se convierte en gráficos acíclicos donde un par de valores de parámetros o una regla es un nodo y una instancia de regla para pares de valores de parámetros son arcos entre las reglas y los valores de los parámetros. En este gráfico, el sistema cuenta la cantidad de caminos entre todos los pares de nodos y genera una tabla para el análisis, se realiza un análisis de sensibilidad para los valores que pueden asumir los parámetros. Con esto, los autores creen que la herramienta le permite al usuario probar la integridad del sistema experto para el problema, le da al usuario una sensación intuitiva de cómo fluye la información en el sistema y lo ayuda a juzgar la idoneidad de los sistemas expertos. para resolver el problema tarea.

Se desarrolló un conjunto de siete herramientas de validación en el proyecto *Esprit*-II *VALID* para ayudar a los ingenieros del conocimiento en una variedad de tareas de validación necesarias para construir un sistema basado en el conocimiento como un todo (Meseguer, 1991). El entorno *VALID* es un *software* que integra este conjunto de herramientas y ofrece una interfaz gráfica para apoyar el proceso de validación. Una de las herramientas es *CONCRET*, utilizada para apoyar el refinamiento del conocimiento y la revisión del conocimiento estratégico de la base de conocimiento a través del procesamiento automático de un conjunto de casos de ejemplo, análisis del proceso de razonamiento de los casos erróneos y visualización al ingeniero del conocimiento. de los posibles errores en la base de conocimiento. El CONTROLADOR también utiliza un conjunto de casos para verificar la consistencia de los datos de entrada solicitados, el resultado generado y el resultado contra las entradas. CURRICULUM KB se utiliza durante la implementación y prueba de la base de conocimiento para inspeccionar su evolución a través de sus diferentes objetos (reglas, instancias, etc.). Para seguir la ejecución existe el *EXECUTION-VISUALIZER* (EV por sus siglas en inglés), que ofrece un seguimiento de ejecución para un caso de entrada, permitiendo además colocar puntos de control con información específica. La consistencia de la base de conocimientos se verifica mediante la herramienta *IN-DEPTH* basada en las restricciones de integridad establecidas, que también incluye la gestión de la incertidumbre y el metacontrol. Otra herramienta para verificar la consistencia de la base de conocimientos es el *PENIC*; sin embargo, solo se utiliza para sistemas basados en lógica proposicional, sin metacontroles ni incertidumbre. La última herramienta es *WITNESS*, que sirve para registrar las opiniones de diferentes personas (usuarios finales, expertos e ingenieros del conocimiento) sobre el sistema basado en el conocimiento que se está evaluando. De esta manera, se crea una base de

comentarios que puede ser referenciada por el equipo de desarrollo para proponer modificaciones bien fundadas.

Metodologías de Verificación y Validación

Según Collins et al. (1994) & Gao et al. (2019) la falta de una metodología para la verificación y validación de sistemas expertos implica la necesidad de probar diferentes enfoques para seleccionar los métodos apropiados que serán más aplicables y brindarán mejores resultados en cada circunstancia. En efecto, lo que se encuentra en la literatura son experiencias prácticas en diferentes áreas de aplicación y con limitada generalización (Ceran et al., 2022).

Según (Meseguer, 1991; Rothenberg et al., 1987) los enfoques de verificación y validación de los sistemas expertos se dividen en tres clases de enfoques:

(i) Orientados a la base de conocimiento, que se ocupan de posibles problemas lógicos del conocimiento (tales como consistencia, completitud, redundancia, etc.)

(ii) Basados en criterios de evaluación, que se ocupan de lo que una evaluación debe medir y proponer criterios para la verificación y validación,

(iii) Basados en métodos de evaluación, que se ocupan de cómo el experto el sistema debe ser evaluado, encontrándose, muchas veces, como parte del ciclo de vida de estos sistemas, incluyendo evaluaciones cualitativas y cuantitativas.

A continuación, describiremos algunos enfoques presentados en la literatura, destacando la propuesta de cada autor para la verificación y validación de sistemas expertos.

El primer enfoque metodológico del que nos ocuparemos es el propuesto por McCall et al. (1977). Su propuesta no es más que la inclusión de actividades específicas de los sistemas expertos en el proceso de desarrollo de un sistema, que comprende cinco etapas: definición de requisitos, verificación de la base de conocimiento y *software* de soporte, preparación de casos

de prueba, ejecución de pruebas y evaluación de resultados. El paso de definición de requisitos es fundamental para el proceso de verificación y validación de sistemas expertos y puede requerir muchas interacciones entre usuarios y desarrolladores porque los requisitos a veces son difíciles de determinar. La especificación de requisitos para sistemas expertos difiere de la especificación de requisitos para sistemas convencionales. La verificación se realiza a partir de la especificación de requisitos. La selección de casos de prueba debe ser cautelosa, tratando de probar todos los requisitos, todas las posibles decisiones que se generarán y probar no solo la salida del sistema sino el proceso por el cual el sistema determinó la salida. A partir de la ejecución de los casos de prueba, se realiza la evaluación. Green considera que la evaluación debería ser realizada por expertos que no hayan estado involucrados en el desarrollo para que el proceso sea más independiente. Esta evaluación debe tener en cuenta si se obtuvo el resultado esperado o un resultado aceptable, si la justificación de los resultados (en el caso de sistemas que generan la justificación) está en una forma adecuada para el usuario, si el proceso de inferencia utilizado por el sistema es lo suficientemente completo y robusto, si el conocimiento utilizado es lo suficientemente profundo para garantizar la confianza del usuario en el sistema y si no se espera que el sistema alcance el 100% de precisión, también se debe analizar el impacto de los resultados incorrectos.

El enfoque de Rothenberg et al. (1987) & Meseguer (1991) considera un proceso formal de verificación y validación de sistemas expertos en áreas críticas. La metodología propuesta tiene aspectos del proceso de desarrollo clásico, especialmente en la descomposición de arriba hacia abajo, y consta de seis pasos. La primera etapa consiste en la definición de requisitos a partir del desarrollo de un prototipo rápido para una mejor comprensión del problema a resolver. La segunda etapa consiste en elaborar el proyecto en términos de paradigmas formales (como

hilos o reglas de producción) para garantizar una implementación analizable, esto implica elegir uno o más paradigmas de procesamiento del conocimiento que se utilizarán. Con esto se realiza el tercer paso que es la certificación de que la máquina de inferencia soporta todos los paradigmas elegidos. Este paso implica: definición formal de un paradigma de inferencia, especificación de cómo se representa este paradigma en la herramienta utilizada por el sistema, desarrollo de pruebas apropiadas para el paradigma en una representación abstracta, traducción de estas pruebas al lenguaje de la herramienta específica y certificación de las pruebas. El cuarto paso se refiere a la verificación a nivel de diseño de alto nivel, que consiste en determinar que el diseño cumple con todos los requisitos. Una vez completada la verificación del diseño de alto nivel, el desarrollo y la verificación pasan a un nivel inferior, ya sea con más detalle o directamente en el código, realizando así la verificación de la base de conocimiento. Esta verificación debe ser realizada por personas que no sean los desarrolladores. Debe confirmarse que la base de conocimientos se ajusta al paradigma especificado, verificar la estructura de determinados subproblemas, confirmar que la base de conocimientos es correcta o razonable mediante un análisis estático del código y posiblemente incluir reglas para marcar fallas de ciertas condiciones críticas. El último paso consiste en la validación formal del desempeño con la prueba de comportamiento del sistema. Los autores proponen el uso de pruebas formales. Para la validación, es necesario identificar los criterios de calidad que desea lograr, definir métricas objetivas para estos criterios, desarrollar una biblioteca de casos de prueba, validar el sistema con especialistas no involucrados en el proyecto, probar el sistema en paralelo con personas ajenas al proyecto. métodos automatizados durante un período de tiempo comparando resultados y manteniendo información detallada sobre el rendimiento del sistema a medida que se construye la base de conocimiento.

El enfoque de Meseguer (1991) se diferencia de los anteriores porque no es una metodología de desarrollo preocupada por la validación y verificación de sistemas expertos, sino una organización del trabajo para validar estos sistemas. La Tabla 2 muestra el resumen de lo observado. El primer aspecto se refiere a la validación del contenido mediante el examen directo de la base de conocimientos por parte de los especialistas, la prueba de Turing y la evaluación del sistema en relación con otros modelos para el mismo problema. En lo que refiere a la validez de los criterios el autor discute la dificultad de establecer medidas para definir el nivel de experiencia deseado, determina que se debe evaluar la consistencia, exhaustividad y adecuación de la base de conocimientos y que el método para evaluar estos criterios es particular para cada aplicación. La validez de la construcción se refiere a la importancia de la existencia de una teoría en la que se basa el sistema, ya que el uso de un enfoque empírico simplemente no es tan eficiente al desarrollar un sistema experto como un enfoque basado en una teoría. La objetividad se refiere a minimizar las variaciones en el juicio del sistema mediante la eliminación del sesgo de expertos en el sistema mediante la administración de pruebas independientes, el uso de técnicas que omiten la identidad de los participantes en la validación y la realización de pruebas informales periódicas realizadas por el programador. El análisis de costo-beneficio es fundamental en la validación, a pesar de ser una tarea difícil de realizar. Para analizar los beneficios se debe analizar cómo se utilizará el sistema (comercialmente, beneficios sociales, etc.) y en cuanto al costo de validar un sistema, se debe analizar su formalidad y hasta qué punto se requiere la validación. Para garantizar la confiabilidad del sistema, se deben realizar análisis de sensibilidad y patrones de problemas de prueba utilizados para la revalidación del sistema para garantizar, por ejemplo, que agregar conocimiento a la base de conocimiento no genere contradicciones. El autor también sugiere que debe haber una variación sistemática en las

pruebas para una validación exitosa. Por lo tanto, debe elegir pruebas que reflejen una gran cantidad de problemas encontrados, realizar variaciones en estas pruebas, elegir una buena cantidad de pruebas. El control de la variación externa significa que la influencia de las variables independientes externas se minimiza, anula o aísla. Para ello se consideran factores como la complejidad del sistema y el aprendizaje durante el proceso de validación. Finalmente, se considera que debe minimizarse la variación del error que normalmente provocan las respuestas expertas inadecuadas.

Tabla 2

Adaptación de la metodología para la gestión del trabajo de validación

Actividad	Descripción
Validez de contenido	Examen directo del sistema por especialistas
	Pruebas del sistema contra especialistas humanos (test de Turing)
	Pruebas del sistema contra otros modelos
Validez de criterios	Definición del nivel de validez del sistema
	Criterios para la base de conocimiento
	Determinación de la validación de los criterios
Validez de construcción	
Objetividad	Validación por el programador
	Gestión independiente para validación
	Validación por el usuario final
	Técnicas para una validación que no identifique los evaluadores
Costo-beneficio	
Confiabilidad	Pruebas del sistema contra el mismo
	Problemas de pruebas estándar para revalidación
Variación sistémica	
Variación externa	
Variación de error	

Nota. Elaboración propia

Madou et al. (2009) propuso una metodología para evaluar el desempeño y la calidad de los sistemas expertos, midiendo cada atributo de calidad y combinando los valores obtenidos. El autor considera los atributos según aspectos conductuales y ontológicos, definiendo una relación entre ellos. La combinación se define a través de una función que considera las medidas de los atributos. Estas medidas suelen ser valores reales y oscilan entre 0 y 1 para los atributos conductuales y un conjunto finito de valores cualitativos (como muy alto, alto, medio, bajo y muy bajo) para los atributos ontológicos. Para obtener estas medidas se utilizan pruebas con

análisis de entrada-salida para los atributos de comportamiento e inspecciones para evaluar los atributos ontológicos. Además, el autor sugiere que se debe determinar un peso para cada atributo evaluado de modo que represente su importancia relativa con respecto a los demás para determinar el valor de la función. La función de combinación luego considera todas estas medidas utilizando un promedio ponderado.

Características de calidad

Con la importancia otorgada a los procedimientos de aseguramiento de la calidad del *software*, se han realizado varios trabajos para definir los atributos de calidad de los productos de *software*, en general, y considerando las diferentes áreas de aplicación. A partir de esta revisión y reconstrucción literaria se define un conjunto de atributos de calidad que deben ser considerados para la evaluación de sistemas expertos, tanto en términos de especificaciones del sistema como del producto final. Estos atributos fueron organizados según el modelo propuesto por McCall et al. (1977) y definidos a partir de la ISO/IEC 25010, a partir de trabajos previos realizados en evaluación de la calidad (Grogono et al., 1993; Madou et al., 2009; Martín & Sáenz, 2016) y la literatura técnica en el área de los sistemas expertos basados en el conocimiento (Grogono et al., 1993; Meseguer, 1991; Rothenberg et al., 1987).

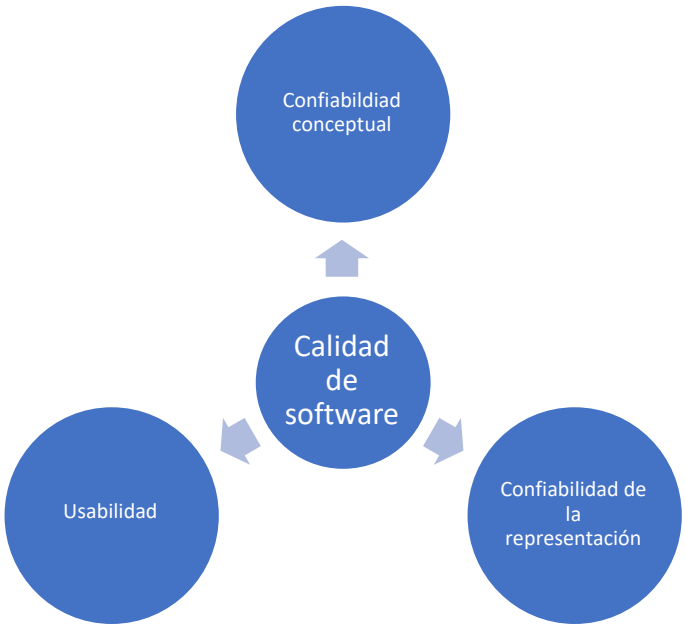
Atributos para la evaluación de la calidad

Según el modelo propuesto por McCall et al. (1977), los objetivos de calidad se logran a través de factores que pueden estar compuestos por subfactores. Los factores y subfactores se evalúan mediante criterios, para lo cual se deben definir procesos de evaluación. Es entonces donde se entiende que los productos de *software* se desarrollan para satisfacer ciertas necesidades de sus usuarios. Una vez puestos en funcionamiento, se espera que tengan una vida útil larga y productiva. Para que esto suceda, se deben alcanzar los siguientes objetivos de

calidad (McCall et al., 1977): usabilidad, confiabilidad conceptual y confiabilidad de representación (Figura 5).

Figura 5

Objetivos para alcanzar la calidad en el desarrollo de software



Nota. Elaboración propia

Usabilidad

La usabilidad es un objetivo fundamental, se refiere a las diferentes formas en que se puede utilizar el *software* durante su desarrollo y vida útil. En otras palabras, se refiere a las características de uso del *software* en las más diversas formas, tanto durante su proceso de desarrollo como durante su funcionamiento lo que se traduce en aumento de la vida útil. Un producto de *software* normalmente se utiliza durante su desarrollo para los siguientes propósitos:

Mantenimiento de especificaciones (productos intermedios del proyecto) para incorporar nuevos requisitos, funciones, objetos y/o conocimientos;

Evaluación de los productos generados durante el desarrollo;

Realizar cambios resultantes de las evaluaciones realizadas;

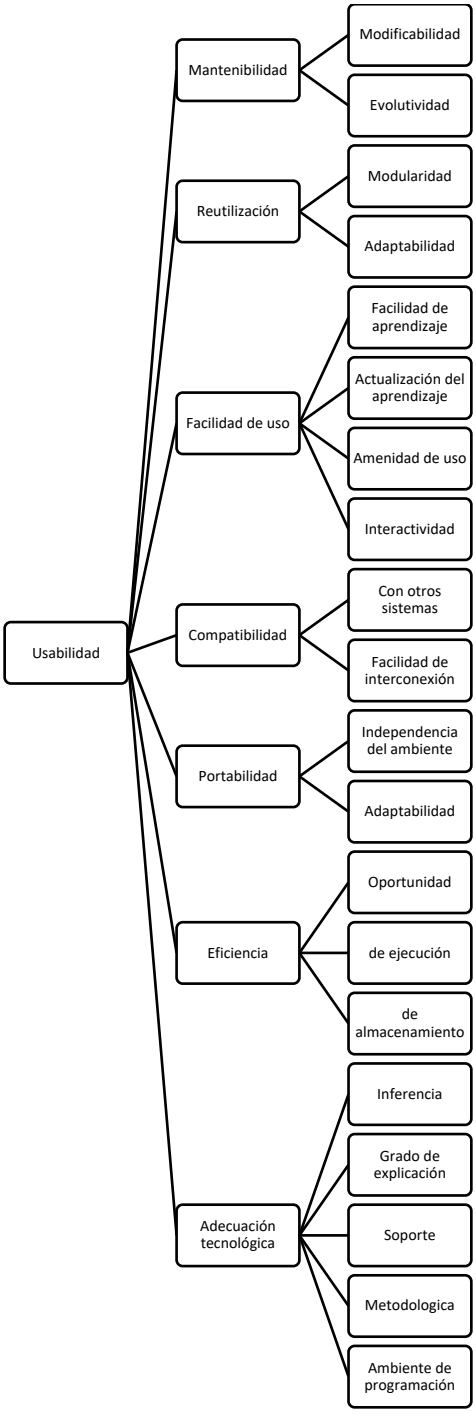
Implementación del producto a partir de sus especificaciones y proyecto.

Durante su vida operativa, el producto debe poder ser utilizado por sus diferentes usuarios, facilitando sus tareas y cumpliendo con los requisitos que motivaron su construcción. Para ello, el producto debe operar de forma eficiente y rentable, como resultado debe ser compatible con otros sistemas que operen en el mismo entorno y con los que necesite interactuar. Además, debe ser posible convertirlo a otro entorno de *hardware*/*software* cuando sea necesario. Estas formas de uso del *software* sugieren los atributos de calidad que debe tener para satisfacer las necesidades de sus usuarios.

En el caso de la implementación del sistema, es necesario verificar si se están utilizando técnicas de codificación apropiadas. Por lo tanto, existen los siguientes factores y subfactores de calidad relacionados con el objetivo de usabilidad, cuando consideramos productos de *software* tipo sistemas expertos. Por tanto, se considera que este objetivo se alcanza a través de los factores mantenibilidad, reutilización, facilidad de uso, compatibilidad, portabilidad, eficiencia y adecuación tecnológica como se muestra en la Figura 6.

Figura 6

Factores y subfactores relacionados con la Usabilidad



Nota. Elaboración propia

Teniendo en cuenta estos factores y los criterios relacionados con ellos, es posible evaluar si estas características se están logrando a lo largo del proceso de desarrollo y están presentes en el producto final. Es importante para cualquier tipo de sistema, asegurarse de que pueda ser fácilmente modificable y evolucionable, pero en el caso de los sistemas expertos, la capacidad de evolución es una característica fundamental ya que la base de conocimiento normalmente sufre un refinamiento progresivo para volverse cada vez más completa.

Confiabilidad de la Representación

La confiabilidad de la representación se refiere a las características que hacen confiable el producto para sus usuarios, considerando aspectos relacionados con su forma y que posibilitan su comprensión y manipulación, teniendo en cuenta las diferentes representaciones del producto, desde la especificación de requisitos hasta el producto en sí código. Esta característica hace posible que el *software* sea entendido y manipulado por sus diferentes tipos de usuarios durante el período de desarrollo y vida operativa.

Durante su desarrollo y vida operativa, los productos de *software* son manipulados por distintos usuarios, en general, distintos a sus desarrolladores y que muchas veces no tienen posibilidad de contacto con ellos. El objetivo de confiabilidad de la representación se refiere a los aspectos necesarios para facilitar la comprensión y manipulación del *software* en sus diversas formas de representación (especificación, proyecto y código). Así, los atributos identificados son necesarios en cualquier tipo de sistema, incluidos los sistemas expertos.

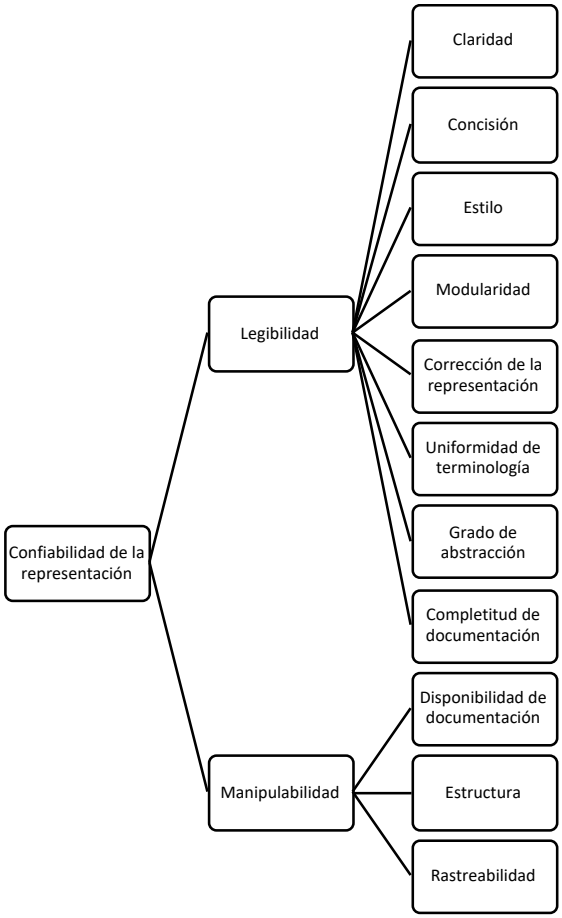
Al considerar el caso específico de los sistemas expertos, es importante tener en cuenta que, como el conocimiento es citado, debe ser representado de tal manera que pueda ser entendido y validado por los especialistas involucrados en su desarrollo. Para ello, además de proporcionar una documentación comprensible, es importante tratar de organizar el

conocimiento de forma jerárquica que pueda facilitar la manipulación y evaluación del conocimiento por parte de especialistas. Estos aspectos también son esenciales en las futuras evoluciones de los sistemas, una característica típica de los sistemas expertos.

Para que sea posible una implicación real de los especialistas en el desarrollo del sistema, es necesario que la documentación generada durante el desarrollo sea fácilmente comprensible para ellos y fácil de manipular, de manera que se pueda encontrar en todo momento la información deseada. Por tanto, este objetivo se alcanza a través de los factores de legibilidad y manipulabilidad como se muestra en la Figura 7.

Figura 7

Factores y subfactores relacionados con la Confiabilidad de la Representación



Nota. Elaboración propia

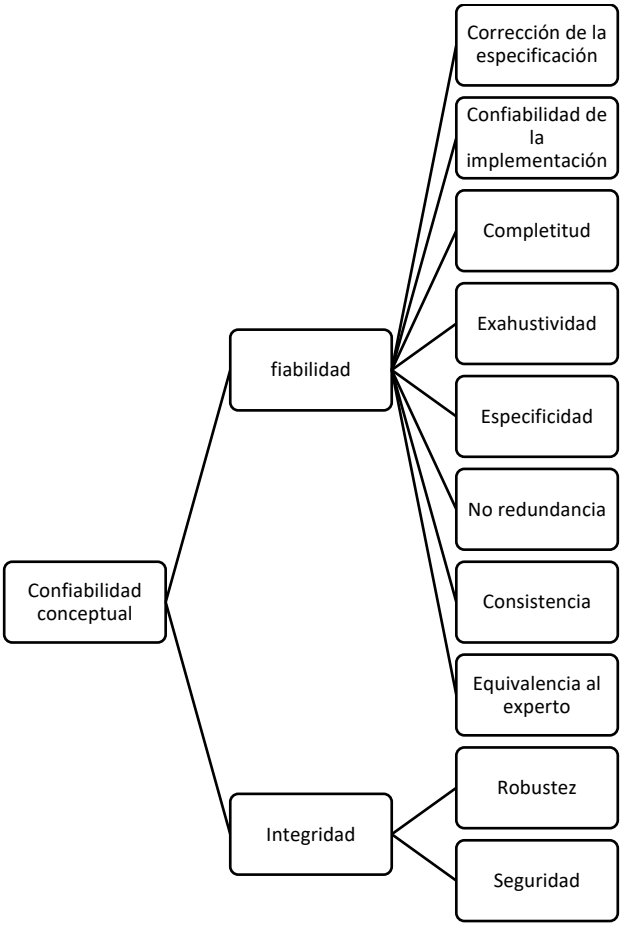
Confiabilidad Conceptual

La Confiabilidad conceptual es un objetivo de gran importancia para la calidad de un producto porque se refiere a las características que hacen que el producto sea confiable para sus usuarios, desde el punto de vista de su contenido, satisfaciendo los requisitos que motivaron su construcción, por lo tanto, buscan garantizar que el *software* corresponde a lo que propone. Por lo tanto, evaluar la confiabilidad conceptual significa evaluar el producto con respecto a su contenido.

La evaluación de la confiabilidad conceptual significa buscar que tanto la documentación generada durante el desarrollo como la base de conocimiento se correspondan con el conocimiento elicitado por el experto. En este sentido, por lo tanto, enfatizamos la importancia de tener una base de conocimiento lo más completa posible, que genere soluciones para todos los casos típicos, que no tenga un conocimiento redundante o contradictorio y que corresponda, en la medida de lo posible, al conocimiento del especialista humano, ya que el sistema actuará como tal. Por lo tanto, este objetivo se logra a través de los factores de confiabilidad e integridad como se representa en la Figura 8.

Figura 8

Factores y subfactores relacionados a la confiabilidad conceptual



Nota. Elaboración propia

Metodología

Tipo de investigación

El presente proyecto aplicado tiene como fundamento el desarrollo de una investigación exploratoria, la cual proporciona "una mayor familiaridad con el problema [...] su objetivo principal es la mejora de ideas o el descubrimiento de intuiciones" (Gil, 2002).

El estudio empírico es la delimitación de esta investigación. En un primer paso da inicio a la revisión de la literatura de los trabajos producidos por otros autores y referentes tanto en artículos científicos como en libros que permita edificar que tan desarrollado está el tema y que se encuentra como conclusiones por parte de los referentes encontrados.

La metodología con la que se pretende alcanzar cada uno de los objetivos planteados está basado en el método de investigación cualitativo “Análisis de contenido”, cuyo propósito es una aproximación al diseño definitivo de una investigación en la que el análisis de contenido es una técnica elegida para elaborar, registrar y tratar datos sobre “documentos” como método de la investigación exploratoria no experimental (Leiva, 2015).

A través de los términos de búsqueda se diseña la consulta específica relacionada con el objeto de estudio para plantear un problema de análisis que permita definir los parámetros, categorías y criterios para medir y obtener el aseguramiento de la calidad en el desarrollo de sistemas expertos.

El origen epistemológico de este método lleva al análisis del conjunto de procedimientos que permiten la interpretación cuantitativa desde la estadística basada en el recuento de unidades, incluyendo la cualitativa desde la lógica basada en la combinación de categorías, tienen por objeto elaborar y procesar datos relevantes sobre las condiciones mismas en que se han producido aquellos textos, o sobre las condiciones que puedan darse para su empleo posterior. El

análisis de contenido, de hecho, se convirtió a finales del siglo XX en una de las técnicas de uso más frecuente en muchas ciencias sociales, adquiriendo una relevancia desconocida en el pasado a medida que se introdujeron procedimientos informáticos en el tratamiento de los datos (Luis & Raigada, 2002).

Al mapear las contribuciones teóricas y metodológicas utilizadas en la investigación en aseguramiento de la calidad en el desarrollo de sistemas expertos, se tiene como intención identificar la variación en la metodología utilizada cuando la investigación se refiere al análisis y la aplicación en un conjunto de parámetros aplicables para el aseguramiento de la calidad en el desarrollo de sistemas expertos, permitiendo una visión general organizada de producción, el registro de tendencias o brechas en el tema investigado y la tipología de la investigación.

Metodología del análisis de contenido

De acuerdo con los anteriores planteamientos un análisis de contenido consta de los siguientes pasos:

a. Selección de la comunicación que será estudiada (tipo de textos escritos) permite enmarcar fenomenológicamente el objeto material de análisis y en su virtud, también las fuentes del material que haya de configurar el cuerpo del estudio. Para esta etapa se ha considerado la búsqueda de textos escrito tipo artículo científico, (*review*) libros y memorias de eventos de investigación utilizando los términos de búsqueda.

a. Selección de las categorías que se utilizarán: Las categorías siempre derivan de las miradas, o lo que es más preciso, de las representaciones que permiten la mirada del objeto de análisis. La vigencia de estas representaciones en el conocimiento vulgar, comparada con la del conocimiento científico, reside en una confianza contingente que va pegada al conocimiento particular de una actividad o acontecer particular. La vigencia de sus representaciones en el

conocimiento científico reside en la transcendencia de su refutabilidad, más allá del propio conocimiento de una actividad o acontecer particular, en la investigación básica concierne a todo el capital de representaciones cognitivas de que se dispone para extraer, a medio o largo plazo, posibles prácticas, los objetos de estudio, para la refutación, adquieren singularidad si son representativos, y si no la pierden por su forma de elaboración. En consecuencia, el análisis de contenido nunca puede ser independiente, cuando se aplica, una teoría. Las categorías del objeto de estudios serán establecidas de acuerdo con la revisión exploratoria inicial basado en los que fenomenológicamente se identifique.

a) Selección de las unidades de análisis: Las unidades para el análisis adquieren entonces una refutabilidad que procede de la teoría en virtud de la cual se han decidido cuáles sean sus categorías. Esto brinda la posibilidad de elaborar, registrar y después procesar datos a partir del tratamiento de los productos singulares guardados, grabados o conservados pertenecientes a los criterios de inclusión y exclusión extraídos de ellos; no todos los datos que pueden ser elaborados, registrados y tratados, a partir de la disección de productos comunicativos o “textos”, resultarán adecuados, y serán significativos y suficientes, para representar científicamente la situación comunicativa que ha de constituir el objeto científico de análisis. De acuerdo con las categorías se registrará en los instrumentos de recolección de la información y del material objeto de estudio para procesar datos a partir del tratamiento de los productos.

b) Selección del sistema de evaluación: El análisis de contenido con diseño longitudinal consiste en analizar el cuerpo objeto del estudio en diferentes momentos de su trayectoria, ya sea aplicando medidas repetidas o sirviéndose de muestras independientes. Se trata de los análisis de tipo sistémico, que desarrolla una teoría sistémica del cuerpo objeto del estudio, según la cual hay que analizar siempre bajo los mismos parámetros el cambio o la

evolución de un mismo cuerpo objeto del estudio. Si el análisis abarca a un número representativo de medios de comunicación, el autor señala que los resultados pueden considerarse como indicadores temáticos. Se debe establecer unos indicadores de tipo sistémico para analizar siempre bajo los mismos parámetros resultantes de los análisis.

En el trascurso y desarrollo de la investigación se realiza un levantamiento de información acudiendo a diferentes fuentes de información como lo son repositorios institucionales, bases de datos especializadas y bibliografía física de las diferentes bibliotecas de la región, de la misma manera, se realizó una revisión de más de 60 documentos que incluyen artículos científicos y académicos, reportes de entidades gubernamentales y estándares internacionales donde se plantean diferentes modelos y estándares de calidad a nivel de proceso, producto y gestión, tanto de orden internacional como latinoamericano, esto con el objetivo de indagar, entender y proponer el contexto del problema que tiene como objeto el estudio. Como resultado de este proceso de indagación y reconstrucción del estado del arte se seleccionaron 37 documentos los cuales fueron la base e insumo principal para la construcción del modelo de prototipo y el documento en general.

Para comenzar, se investigó sobre los fundamentos de la calidad del *software*, incluidas las principales metodologías y técnicas utilizadas para evaluar la calidad del *software*. También se consideró importante entender los requisitos del usuario y los objetivos de negocio en la creación de un sistema de *software*, ya que estos elementos afectan el enfoque de aseguramiento de la calidad que se debe tomar. Luego se tomó como base de exploración la aplicación de sistemas expertos para el aseguramiento de la calidad de *software*. Esto incluyó la identificación y definición de los parámetros necesarios para una evaluación efectiva de la calidad, además de

estudiar como establecer reglas específicas para cada parámetro, utilizando el control estadístico para la calidad del *software*.

Resultados

En la actualidad, el *software* se ha convertido en un elemento crítico en la mayoría de los sistemas y aplicaciones. El aseguramiento de la calidad de *software* es una práctica esencial para garantizar que el *software* producido cumpla con los requisitos y estándares de calidad esperados ya que puede influir en la satisfacción del usuario, el éxito comercial y la seguridad del sistema. Por esta razón, el aseguramiento de la calidad es un proceso clave para garantizar su correcto funcionamiento y reducir el riesgo de fallos y errores que puedan afectar el rendimiento de los sistemas.

Para garantizar que el *software* cumpla con los requisitos de calidad y funcione correctamente, se deben utilizar una serie de parámetros que permitan evaluar su desempeño en diferentes aspectos. Los parámetros son una parte importante de un sistema experto de calidad de *software*, ya que proporcionan una forma de evaluar la calidad de un programa. Un parámetro puede ser cualquier característica que se pueda medir o calcular, como la eficiencia, la fiabilidad, la facilidad de uso o la compatibilidad. Por lo tanto, se resalta la importancia de elegir los parámetros adecuados basado en el propósito que se desea lograr con el desarrollo del *software*. Según el modelo de (McCall et al., 1977) (Moreno et al., 2010) (Singh & Kannojia, 2013) se pueden identificar 11 características que pueden utilizarse como parámetros para evaluar la calidad de un *software*: funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad, capacidad de prueba, capacidad de seguridad, facilidad de instalación, facilidad de formación y documentación.

Así mismo, es importante describir las técnicas utilizadas para optimizar los parámetros y las fuentes de información utilizadas para construir un sistema efectivo.

Un modelo de calidad es un conjunto de características y subcaracterísticas que se utilizan para evaluar la calidad del *software*. Un sistema experto es una herramienta que utiliza el conocimiento y la experiencia de un experto en un dominio específico para resolver problemas o tomar decisiones. Dado que este tipo de sistema puede ser una herramienta valiosa para evaluar y mejorar la calidad del *software*. Para ello, es importante incluir una serie de parámetros que permitan una evaluación exhaustiva del *software*.

Basado en la información expuesta en el estado del arte, en los conceptos y la teoría vinculada al presente proyecto y sobre todo en los estándares y modelos expuestos se desarrolló el modelo de prototipo que incluye un esquema de parámetros para el aseguramiento de la calidad de *software* en el desarrollo de sistemas expertos, el cual en principio va a ser descrito de forma cualitativa.

Por la tanto este modelo de prototipo se presenta como una guía general que se puede adaptar a las necesidades y particularidades de cada proyecto. Es importante recordar que el aseguramiento de la calidad es una tarea continua que se debe integrar en todo el proceso de desarrollo de *software*.

Se definieron los criterios y procesos de evaluación para el caso específico de un sistema experto conceptual, considerando los factores y subfactores a través de los cuales se logra cada uno de los objetivos de calidad considerando un sistema experto en general en los diferentes productos generados a lo largo del proceso de desarrollo. Esta opción es necesaria porque los criterios y los procesos de evaluación relacionados con ellos son extremadamente dependientes del dominio de aplicación e incluso del proyecto en cuestión.

Diseño del prototipo de modelo con parámetros aplicables al aseguramiento de la calidad

Desde el inicio del proyecto, existía la preocupación de que el prototipo tuviera un proceso sistemático de desarrollo y aseguramiento de la calidad, utilizando modernas técnicas de Ingeniería de *Software*. Para ello se definió un proceso de desarrollo adecuado para la construcción de sistemas basados en el conocimiento de acuerdo con los principios establecidos en la norma ISO/IEC 900003:2004.

Este documento define el modelo de ciclo de vida adoptado en el proyecto, los métodos y herramientas para construir el producto, la documentación que se tomó como referencia a lo largo del desarrollo, los procedimientos y métodos para el control de calidad y la gestión del proceso de desarrollo.

El ciclo de vida definido para el prototipo se basa en el ciclo de vida del prototipado evolutivo y propuesto por Hull (1991) para el desarrollo de sistemas expertos. Este ciclo de vida como se observa en la Figura 9 está compuesto por tres etapas: Análisis de Viabilidad, Evolución y Madurez.

Las etapas de desarrollo se dividen, a su vez, en etapas. El Análisis del Dominio del Problema es un paso único al comienzo de la primera etapa (Análisis de Factibilidad). Su producto es la Especificación del dominio del problema, que abarca la definición del problema de las áreas de conocimiento y el alcance del proyecto. El modelo de ciclo de vida para cada etapa se organiza en hasta siete pasos que se aplican de forma incremental enfocado a madurar el prototipo de acuerdo a lo descrito por Hauge et al. (2006):

Planificación del Proyecto, en el que se elabora inicialmente el Plan del Proyecto. En cada etapa se detalla este Plan de acuerdo con el objetivo y características de la etapa;

Análisis de conocimiento, durante el cual los desarrolladores obtienen conocimiento y producen el modelo conceptual para la etapa, descrito en la Especificación de requisitos;

Proyecto, donde se traduce el modelo conceptual al paradigma de Inteligencia Artificial a utilizar. El producto de esta fase es la Especificación del Proyecto;

Construir, donde se construyen el programa y la base de conocimiento;

Evaluación de Producto, donde expertos y otros evaluadores prueban el sistema. Las pruebas de aceptación determinan si se han cumplido los requisitos descritos en la Especificación de requisitos;

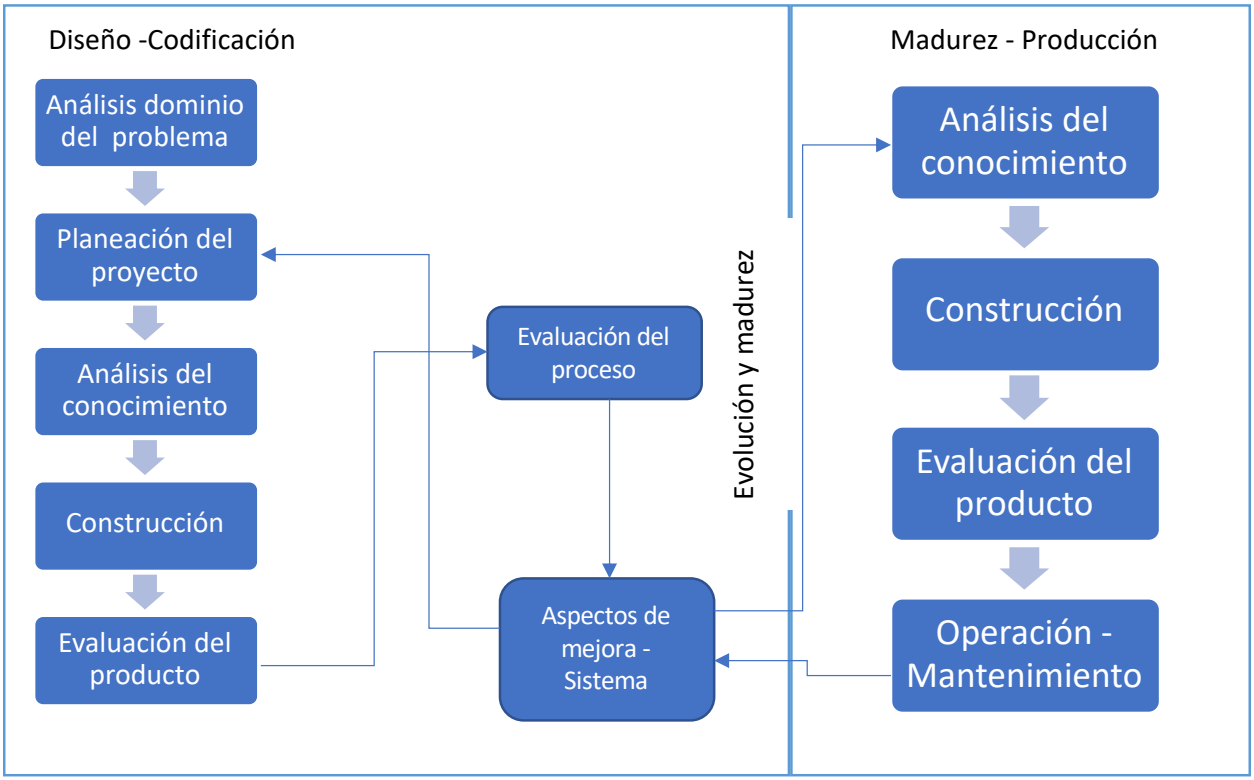
Operación, es la actividad que permite utilizar el sistema aceptado;

Evaluación del proceso, paso clave, realizado al final de cada etapa, que es la base para las revisiones en el proceso de desarrollo y para la planificación y ejecución de la siguiente etapa.

Para el modelado del sistema se optó por el método Conocimiento Adquisición y Estructuración del Diseño (*Knowledge Acquisition and Design Structuring* KADS por sus siglas en inglés) (Buchanan & Smith, 1988). Se puede encontrar una evaluación del uso del proceso de desarrollo en la construcción de la primera versión del prototipo en (Madou et al., 2009; Martín & Sáenz, 2016; Meseguer, 1991; Rothenberg et al., 1987).

Figura 9

Representación del ciclo de vida para sistemas expertos



Nota. Elaboración propia

Planeación de calidad

La Planificación de la Calidad del prototipo se realizó al inicio de la primera etapa de desarrollo asociada al Análisis de Factibilidad de acuerdo con el modelo definido en el ciclo de vida. Este debe evaluarse al final de cada etapa en la fase de evaluación del proceso y si es necesario, reformularse al planificar una nueva etapa de desarrollo. Por lo tanto, el Plan de Control de Calidad hace parte integral del Plan del Proyecto.

Sin embargo, antes de iniciar cualquier procedimiento de control de calidad, es fundamental identificar los requisitos de calidad específicos de un proyecto. Estos requisitos deben incorporarse al documento de especificación de requisitos de *software* y serán la base para

el control de calidad. Para establecer los requisitos de calidad se definió un proceso donde las características de calidad se dividieron en dos conjuntos: atributos relacionados con la calidad externa del producto y atributos relacionados con la calidad interna calidad.

Los atributos identificados de calidad interna del producto fueron definidos como esenciales y deseables. La tabla 3 muestra, a modo de ejemplo, el formulario construido para la identificación de los requisitos de calidad. La tabla 4 consolida los requisitos de calidad definidos, con los cuales se planeó llevar a cabo la evaluación de la calidad a lo largo del proceso de desarrollo.

Tabla 3

Formulario de Identificación de Requisitos de Calidad

Atributo	Descripción	Imprescindible	¿Deseable?	Importante	Importante para el prototipo
Funcionalidad	Característica de uso para tener consecuencias significativamente útiles				
Amenidad de uso	Característica para presentar los resultados claramente, para poder proporcionar asistencia rápidamente y para presentar estabilidad en el uso				
Equivalente al especialista	Característica para tener un desempeño con confiabilidad equivalente a un especialista humano, tanto en términos de alcance del problema como de la solución generada				
Seguridad	Características de confiabilidad desde el punto de vista de su uso, sin riesgo de causar daños materiales y/o vidas humanas,				

	protegiéndose de violaciones y garantizando la privacidad
…	

Nota. Elaboración propia

Identificación de los parámetros

Como se pudo verificar el estándar ISO/IEC 25010 presenta un modelo permite identificar los parámetros que se utilizarán para evaluar la calidad del *software*, este estándar define ocho características principales para la búsqueda de calidad en el *software* que incluye el rendimiento, la fiabilidad, la usabilidad, la seguridad, la facilidad de mantenimiento, entre los otros descritos.

Funcionalidad: Se define como "la capacidad del *software* para proporcionar funciones que satisfagan las necesidades declaradas e implícitas, cuando se utiliza en las condiciones especificadas", en otras palabras, se refiere a la capacidad del *software* para hacer lo que se espera de él y para hacerlo bien.

Eficiencia: Es uno de los parámetros más importantes para evaluar la calidad del *software*. Se refiere a la velocidad con la que el *software* puede realizar sus tareas, la cantidad de recursos que utiliza y la capacidad para manejar grandes volúmenes de datos. Un sistema experto debe incluir una evaluación de rendimiento que permita medir la velocidad de ejecución del *software* y la eficiencia en la gestión de recursos.

Confiabilidad: Se refiere a la capacidad del *software* para funcionar correctamente y sin errores durante un período prolongado de tiempo. Un sistema experto debe incluir una evaluación de fiabilidad que permita detectar errores y fallos en el *software*, así como la capacidad para recuperarse de fallos.

Usabilidad: Se refiere a la facilidad de uso del *software* por parte de los usuarios, esta característica se descompone en sub características como la comprensibilidad, la facilidad de aprendizaje, la operabilidad y la estética visual. Al definir los requisitos de calidad para un *software* específico, se pueden utilizar estas subcaracterísticas como criterios para evaluar la calidad del *software* en términos de usabilidad. Un sistema debe incluir una evaluación de usabilidad que permita medir la facilidad de uso del *software*, la claridad y la accesibilidad de la interfaz de usuario.

Mantenibilidad: Se refiere a la capacidad del software para ser mantenido y actualizado con facilidad. Un sistema experto debe incluir una evaluación de mantenibilidad que permita medir la facilidad de mantenimiento y la capacidad del *software* para ser actualizado sin afectar su funcionamiento.

Portabilidad: Se refiere a la capacidad del *software* para funcionar en diferentes plataformas y sistemas operativos. Un sistema experto debe incluir una evaluación de portabilidad que permite medir la capacidad del *software* para ser utilizado en diferentes entornos y sistemas operativos.

Seguridad: Es un parámetro crítico para evaluar la calidad del *software*, ya que este puede ser vulnerable a ataques maliciosos que comprometan la información y la privacidad de los usuarios. Un sistema experto debe incluir una evaluación de seguridad que permita detectar vulnerabilidades y riesgos de seguridad en el *software*.

Estándares específicos - Definición de las reglas

A partir de los parámetros definidos, se deben establecer estándares específicos que también pueden ser llamadas o contempladas como las reglas del sistema. Por ejemplo, si se utiliza el parámetro de rendimiento, se deben definir las reglas que establezcan el rendimiento

mínimo aceptable para el *software*. Se identifica que para cada parámetro se puede incluir atributos. A partir de un atributo, la métrica es una función que, aplicada sobre sus medidas determina una cuantificación de la calidad del atributo.

Definir los requisitos de calidad: Se refieren a los criterios que se utilizan para evaluar la calidad del *software*. Estos criterios se establecen en función de las necesidades y expectativas de los usuarios, los objetivos del *software* y las características técnicas del sistema, esto incluye definir los objetivos de calidad, los criterios de aceptación, las especificaciones funcionales y no funcionales, entre otros aspectos. Es importante destacar que los requisitos de calidad pueden variar según el tipo de *software* que se esté desarrollando. Una forma de definir los requisitos de calidad es a través de la elaboración de un modelo de calidad.

Definir el conocimiento: A partir del dominio definido, se debe identificar y recopilar el conocimiento de un experto en ese dominio. Esto incluye la definición de reglas, heurísticas y otros conocimientos que el sistema experto utilizará para realizar las recomendaciones o tomar decisiones.

Definición de la base de conocimientos: A partir de las reglas establecidas, se debe crear la base de conocimientos del sistema experto. Esta base de conocimientos incluirá información sobre los parámetros y las reglas establecidas para su evaluación.

Identificar el dominio: Lo primero es definir el dominio del sistema experto, es decir, los aspectos específicos del aseguramiento de la calidad de *software* en los que el sistema va a ser experto. Esto puede incluir pruebas de funcionalidad, pruebas de rendimiento, pruebas de seguridad, entre otros.

Diseñar la arquitectura del sistema: A partir del conocimiento definido, se debe diseñar la arquitectura del sistema experto, que incluye los componentes y módulos que lo componen. Esto

puede incluir un motor de inferencia, una base de conocimientos, una interfaz de usuario, entre otros.

Implementar la base de conocimientos: Este es el corazón del sistema experto y contiene toda la información y reglas que el sistema utilizará para realizar las recomendaciones o tomar decisiones. Se debe implementar la base de conocimientos utilizando herramientas específicas para sistemas expertos, como *CLIPS* o *Jess* (Madou et al., 2009).

Implementar el motor de inferencia: Es el componente que utiliza la base de conocimientos para tomar decisiones o realizar recomendaciones. En este caso, el motor de inferencia utilizará las reglas definidas para evaluar la calidad del *software* y proporcionar recomendaciones en consecuencia. Se debe implementar el motor de inferencia utilizando una herramienta específica, como *Pyke* o *Drools*.

Codificación: La codificación del *software* debe seguir las normas y estándares establecidos por la organización. Además, se deben realizar pruebas de código para asegurar que el código esté libre de errores y cumpla con los requisitos.

Implementar la interfaz de usuario: Debe ser diseñada para que los usuarios puedan interactuar con el sistema experto. Los usuarios deben ser capaces de ingresar los parámetros relevantes del *software*, y el sistema experto debe proporcionar recomendaciones para mejorar la calidad del *software*. Se debe implementar una interfaz intuitiva y fácil de usar que permita al usuario obtener la información que necesita.

Tabla 4

Requisitos de calidad identificados

Imprescindible	Deseable
Grado de Explicación	Adecuación de la metodología
Soporte	Inferencia
Adecuación del entorno de programación	Estilo
Codificación	Uniformidad del grado de abstracción
Claridad	Trazabilidad
Concisión	Adaptabilidad
Modularidad	Compatibilidad con otros sistemas
Corrección de representación	Adaptabilidad
Uniformidad de terminología	Eficiencia de almacenamiento
Integridad de la documentación	Rentabilidad
Disponibilidad de documentación	Competitividad
Estructura	
Modificabilidad	
Evolución	
Facilidad de aprendizaje	
Actualización de aprendizaje	
Amenidad de uso	
Interactividad	
Facilidad de interconexión	
Independencia del entorno.	
Oportunidad	
Eficiencia de ejecución	
Beneficio social	
Beneficio laboral del usuario	
Relevancia	
Utilidad	
Justificabilidad	

Posibilidad
Idoneidad
Viabilidad económica
Viabilidad financiera
Factibilidad tecnológica
Viabilidad de la mano de obra
Factibilidad del cronograma
Viabilidad social
Corrección de especificación
Fiabilidad de implementación
Completitud
Agotamiento
Sin redundancia
Especificidad
Consistencia
Equivalencia a experto
Robustez
Seguridad

Nota. Elaboración propia

Durante las distintas etapas del proyecto, se planificaron dos tipos de evaluaciones: evaluaciones intermedias y evaluaciones finales de productos referentes a las fases del proceso de desarrollo.

Las evaluaciones intermedias deben realizarse cuando se concluye una tarea, cuyo resultado tendrá un fuerte impacto en la siguiente tarea. Esta evaluación tiene por objeto obtener la aprobación, según corresponda en cada situación, de expertos, usuarios o personal técnico sobre la idoneidad y exhaustividad del producto de la tarea.

Las evaluaciones del producto final siempre se realizan al completar las tareas de una etapa y antes de pasar a la siguiente etapa. Esta evaluación tiene como objetivo obtener la aprobación del coordinador del proyecto y de los especialistas para el producto de la etapa.

Las evaluaciones se realizan mediante la técnica de inspección por fases (Rothenberg et al., 1987). Se eligió esta técnica para realizar las evaluaciones ya que se deben analizar diferentes requisitos de calidad en un mismo producto, por evaluadores con diferentes perfiles y no siempre con posibilidad de estar presentes en las reuniones de inspección. Por ejemplo, se planeó que con el uso del método KADS se analizaran el modelado del prototipo para determinar la corrección de la representación, esta corrección de la especificación estaba prevista para ser evaluada como se describió en las figuras que representaban a modo ilustración los factores y subfactores.

Dado que existen amplias diferencias entre el desarrollo de *software* convencional y sistemas expertos se identifican algunas peculiaridades de aplicación en la validación de estos sistemas bajo este mismo prototipo de modelo. Como ningún método puede garantizar la corrección total de un sistema, se decidió utilizar una combinación de varios métodos de validación para tener una evidencia complementaria de la fiabilidad del sistema (Meseguer, 1991) (Rothenberg et al., 1987) (P. Miguel et al., 2014).

Por tanto, la planificación de la calidad incluye la realización de pruebas sistemáticas, con casos de prueba previamente definidos y evaluados en cuanto a su idoneidad y corrección del resultado esperado.

Proceso de control de calidad

A lo largo del desarrollo de la primera versión del prototipo, se pasó por un riguroso proceso de aseguramiento de la calidad donde se realizaron diversas evaluaciones de los hitos y puntos de control preestablecidos. Estas evaluaciones se realizaron a través de inspecciones del

material literario, de acuerdo con los requisitos de calidad establecidos en la Especificación de Requisitos y utilizando criterios específicos para la selección de cada producto intermedio y los productos finales de las etapas de diseño. Para esta versión, solo se consideraron parte de los atributos de calidad deseados como se mostró en la figura 10. En la tabla 5 se muestra la agrupación del producto y dominio del problema junto con las especificaciones del dominio del problema y qué atributos de calidad se vincularon a lo largo del desarrollo de la primera versión.

Tabla 5

Atributos de Calidad vinculados a lo largo del diseño

	Producto	Atributos
Especificación del dominio del problema	**Conocimiento general**	Claridad, concisión, uniformidad de la terminología, uniformidad del grado de abstracción, exhaustividad de la documentación, justificación, posibilidad, idoneidad, corrección de la especificación, especificidad, no redundancia, coherencia
	Documento final	Claridad, concisión, uniformidad de la terminología, uniformidad del grado de abstracción, integridad de la documentación, disponibilidad de la documentación, trazabilidad, pertinencia, utilidad, justificabilidad, posibilidad, idoneidad, corrección de la especificación, especificidad, no redundancia, consistencia
Plan del proyecto	**Documento final**	Claridad, completitud de la documentación, disponibilidad de la documentación, trazabilidad, factibilidad económica, factibilidad financiera, factibilidad tecnológica, factibilidad laboral, factibilidad de cronograma, factibilidad social,
Especificación de requisitos	**Objetivo, requisitos del sistema**	Claridad, concisión, uniformidad de la terminología, uniformidad del grado de abstracción, exhaustividad de la documentación, corrección de la especificación, especificidad, no redundancia, coherencia
	Modelado del sistema	Claridad, uniformidad de la terminología, integridad de la documentación, uniformidad del grado de abstracción, exactitud de la especificación, exactitud de la representación, especificidad, no redundancia, consistencia

	Documento final	Claridad, concisión, uniformidad de la terminología, uniformidad del grado de abstracción, exhaustividad de la documentación, trazabilidad, corrección de la especificación, especificidad, no redundancia, coherencia, disponibilidad de la documentación
Especificación del proyecto	**Base de conocimiento**	Claridad, uniformidad de la terminología, integridad de la documentación, estructura, uniformidad del grado de abstracción, codificación, viabilidad tecnológica, exactitud de la especificación, exactitud de la representación, especificidad, no redundancia, consistencia
	Documento final	Claridad, uniformidad de la terminología, uniformidad del grado de abstracción, integridad de la documentación, estructura, trazabilidad, viabilidad tecnológica, corrección de la especificación, corrección de la representación, especificidad, no redundancia, consistencia, codificación
Producto final	**Programa y documentación del proyecto**	Facilidad de aprendizaje, facilidad de uso, interactividad, oportunidad, eficiencia de ejecución, grado de explicación, idoneidad, confiabilidad de implementación, especificidad, no redundancia, consistencia, uniformidad de terminología, uniformidad en el grado de abstracción, integridad de la documentación, disponibilidad de documentación, trazabilidad, adecuación de la metodología, adecuación del entorno de programación, codificación, claridad, concisión, modularidad, corrección de la representación, estructura

Nota. Elaboración propia

Diseñar pruebas: A partir de los requisitos definidos, se deben diseñar pruebas que permitan verificar que el *software* cumple con los objetivos de calidad. Esto incluye pruebas de funcionalidad, rendimiento, seguridad, usabilidad, compatibilidad, entre otras.

Pruebas unitarias: Son una técnica que se enfoca en probar componentes individuales del *software* de manera aislada, con el objetivo de garantizar que cada componente funcione correctamente por separado antes de integrarlos. Estas pruebas pueden realizarse mediante *frameworks* de pruebas automatizadas, lo que permite realizar pruebas más rápidas y repetitivas. Por ejemplo, en el desarrollo de una aplicación web, se pueden realizar pruebas unitarias para probar que la función de registro de usuarios funciona correctamente, verificando que los datos de entrada sean válidos y que la información se guarde correctamente en la base de datos.

Pruebas de integración: Se enfocan en probar la interacción entre diferentes componentes del *software*, para detectar posibles errores que puedan surgir en el momento de la integración. Estas pruebas pueden realizarse a nivel de sistema o de módulo, y se enfocan en comprobar que los componentes interactúen correctamente entre sí, garantizando que la funcionalidad completa del *software* sea la esperada. Un ejemplo de aplicación de las pruebas de integración podría ser probar que un sistema de facturación se integra correctamente con el sistema de inventario, verificando que los datos de productos y precios sean correctos y que se generen las facturas de manera adecuada.

Pruebas de aceptación: Se realizan para verificar que el *software* cumple con los requisitos funcionales y no funcionales establecidos por el cliente o usuario final. Estas pruebas suelen ser realizadas por el usuario final o un representante del cliente, con el objetivo de garantizar que el *software* cumple con sus expectativas. Un ejemplo de aplicación de las pruebas de aceptación podría ser probar que un sistema de reservaciones de vuelos permite realizar reservaciones correctamente, que los precios son los correctos y que el proceso de pago funciona sin problemas.

Implementar pruebas: Una vez diseñadas las pruebas, se debe implementar un conjunto de herramientas y técnicas para realizar las pruebas de forma automatizada o manual. Esto puede incluir el uso de herramientas de automatización de pruebas, pruebas manuales, pruebas exploratorias, entre otras.

Técnicas de aplicación: Para lograr el aseguramiento de la calidad del *software* se estudian indicadores de calidad centradas en la fase de codificación, se debe considerar que existen una serie de atributos del *software* que se pueden medir de forma automática a partir del código fuente, sobre el cual se pueden aplicar diversas técnicas, entre las cuales se destacan:

Automatización de pruebas: La automatización de pruebas permite realizar pruebas de manera más eficiente y eficaz, lo que a su vez permite encontrar más errores y reducir el tiempo de prueba.

Revisión de código: La revisión de código permite detectar errores en el código y mejorar la calidad del *software*. Estas pueden ser realizadas por un equipo de revisores o mediante herramientas automatizadas de revisión de código, en esta técnica se aplica en dos métricas enfocada en el producto: el análisis estático y el análisis dinámico del código fuente.

Análisis estático de código: Esta métrica se desarrolla de forma exclusiva en la fase de codificación, la cual permite detectar errores en el código sin tener que ejecutarlo. Se refiere a la revisión automática del código fuente de un *software* sin ejecutarlo, con el objetivo de detectar posibles problemas de calidad de código que puedan afectar el rendimiento o la seguridad del *software*. Esta técnica permite detectar errores de programación, vulnerabilidades de seguridad, violaciones de estándares de codificación, entre otros problemas, a través del análisis sintáctico y semántico del código fuente. Un ejemplo de aplicación del análisis estático de código podría ser

la revisión automática de un código fuente para detectar vulnerabilidades de seguridad en una aplicación web.

Análisis dinámico de código: Esta métrica se enfoca en evaluar el comportamiento del *software* en tiempo de ejecución, mediante la realización de pruebas que simulan el uso del *software* en diferentes escenarios. Permite detectar problemas de rendimiento, problemas de seguridad, fallos de memoria, entre otros. Un ejemplo de aplicación del análisis dinámico de código podría ser la simulación de usuarios accediendo a un sistema de banca en línea para detectar problemas de rendimiento o seguridad.

Ejecutar pruebas: Una vez implementado el sistema experto, se deben realizar pruebas para comprobar su eficacia. Las pruebas pueden incluir la evaluación de la calidad del *software* a través del sistema experto, la identificación de posibles mejoras en las reglas y la base de conocimientos y la evaluación de la usabilidad de la interfaz de usuario.

Se deben ejecutar las pruebas diseñadas para comprobar que el *software* cumple con los requisitos de calidad establecidos. Es importante documentar y registrar los resultados de las pruebas para su posterior análisis. Una vez implementado el sistema experto, se deben realizar pruebas exhaustivas para comprobar que el sistema funciona correctamente y cumple con los requisitos de calidad establecidos.

Analizar resultados: Una vez ejecutadas las pruebas, se deben analizar los resultados obtenidos para determinar si el *software* cumple con los objetivos de calidad establecidos. Si se detectan errores o fallos, se deben documentar y corregir.

Realizar iteraciones al proceso: El aseguramiento de calidad es un proceso iterativo que se debe repetir durante todo el ciclo de vida del *software*. Es importante seguir mejorando las

pruebas y ajustar los criterios de calidad para garantizar que el *software* cumpla con los requisitos establecidos.

Mantenimiento y mejora continua: El sistema experto debe ser mantenido y mejorado de forma continua para asegurar su eficacia y eficiencia. Esto incluye la actualización de la base de conocimientos, la incorporación de nuevos conocimientos y la mejora de la interfaz de usuario.

Consideraciones finales de la primera versión del prototipo

Este modelo de prototipo como se muestra en la figura 13 vincula un sistema experto que esquematiza los parámetros para evaluar la calidad del *software* como dominio del problema y definición del conocimiento. Para asegurar la calidad del *software* se vinculan parámetros que permiten evaluar su rendimiento, fiabilidad, usabilidad, seguridad, mantenibilidad y portabilidad. Estos parámetros permitirán mejorar la calidad del *software*, reducir los errores y los fallos y mejorar la satisfacción de los usuarios. Es importante destacar que la inclusión de estos parámetros debe ser adaptada a las necesidades específicas de cada proyecto y a las expectativas de los usuarios.

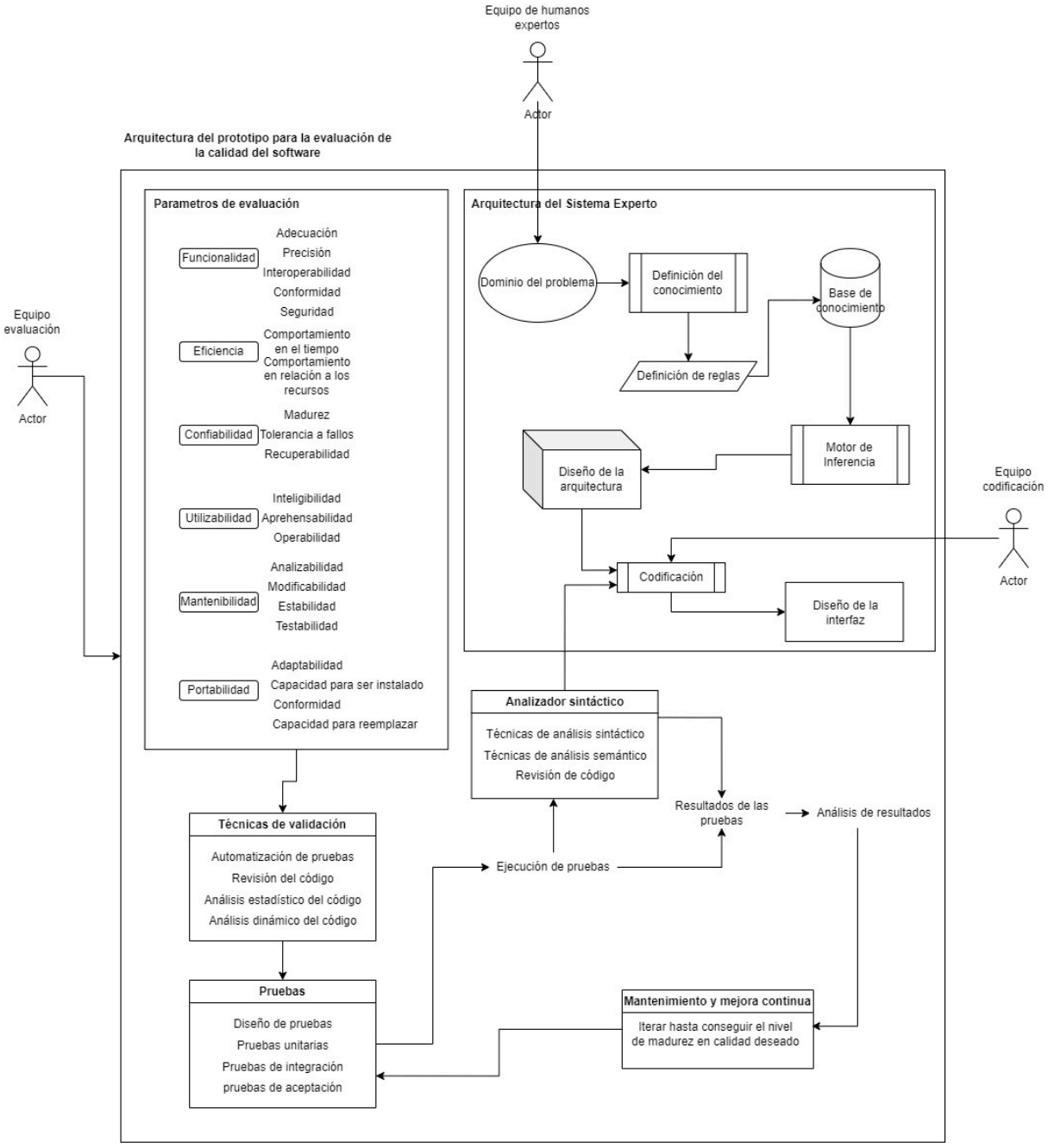
Al utilizar un motor de inferencia y una base de conocimientos, el sistema experto es capaz de proporcionar recomendaciones precisas y personalizadas para mejorar la calidad del *software*. Es importante recordar que el sistema experto es una herramienta valiosa para el aseguramiento de la calidad de *software*, ya que permite tomar decisiones informadas y precisas basadas en el conocimiento y la experiencia de un experto en el dominio.

Es importante mencionar que la inclusión de los parámetros en un sistema experto no es suficiente para garantizar la calidad del *software*. También es necesario contar con fuentes de información adecuadas para la construcción del sistema. Las fuentes de información pueden

incluir manuales de usuario, especificaciones técnicas, revisiones de código, pruebas de rendimiento, entre otros, que no se encuentran contemplados en el presente proyecto.

Figura 10

Primera versión del prototipo de modelo con parámetros aplicables al aseguramiento de la calidad en el desarrollo de sistemas expertos



Nota. Elaboración propia

Conclusiones

Este documento abordó cuestiones relacionadas con la evaluación de la calidad en el desarrollo de *software*. Se realizó un estudio sobre el estado del arte de la calidad en el *software*, modelos de calidad y estándares internacionales, donde se verificó la necesidad de una mejor definición de los atributos que contribuyen a su calidad. Luego se propuso un conjunto de atributos de calidad a considerar en la evaluación de los diferentes productos generados a lo largo del proceso de desarrollo tales como especificaciones y código aplicables al producto final de cualquier desarrollo de *software*.

A partir del estudio de este conjunto de atributos y del estudio de la literatura sobre verificación y validación de *software* y en concreto de sistemas expertos dedicados al aseguramiento, se definieron procedimientos para la construcción de un sistema experto orientado a evaluar la calidad de otros programas informáticos.

Fue posible concluir que a pesar de los avances significativos en temas relacionados con la verificación y validación de sistemas expertos aún queda mucho trabajo por hacer. Se observa en la literatura consultada una cantidad interesante de experiencias en diferentes proyectos con respecto a la evaluación de estos sistemas, principalmente relacionados con la base de conocimiento y su aplicabilidad, pero se resalta la necesidad de una conceptualización más amplia, completa y concisa con relación a las características de calidad que deben ser presentes tanto en la base de conocimiento como en el proceso de desarrollo de estos sistemas considerando los productos intermedios y finales.

Las técnicas de seguimiento de ejecución convencionales mediante el examen del código para verificar que el sistema implementado cumple con los requisitos, lo cual es muy difícil y complejo de realizar. La secuencia de ejecución de un sistema experto a partir del examen de la

base de conocimiento puede ser extremadamente difícil de determinar, incluso con las técnicas tradicionales de prueba de comparación precondición. La capacidad del sistema experto para producir sus resultados es una propiedad de la interacción entre la base de conocimiento y la máquina de inferencia y solo puede analizarse cuando se ejecuta el sistema.

La representación del conocimiento en calidad de aplicaciones informáticas aplicando estándares como ISO/IEC de forma explícita en procesos de decisión debe ser desarrollada por personas que tengan un profundo conocimiento y experiencia en procesos de calidad de software. Cuanto más motivados por los resultados de la investigación más tiempo y esfuerzo será dedicado al desarrollo del sistema experto

Es importante continuar la formación en la cultura de la gestión de la calidad en las organizaciones y en los desarrolladores de *software*, gestión que debe aplicar el uso de técnicas apropiadas para especificar y evaluar tanto la calidad del producto de *software* como la calidad de su proceso de desarrollo. La calidad es un factor importante para el éxito de la toma de decisiones que realiza un sistema y en consecuencia para su uso.

Recomendaciones

Se identifican múltiples posibilidades de trabajos futuros asociados al desarrollo de herramientas específicas que apoyen el proceso de verificación y validación de la calidad en sistemas expertos construidos para tal fin como parte de las pruebas automatizadas.

Se recomienda que este modelo de prototipo se lleve a la fase de diseño y posteriormente a la codificación de un sistema experto dado que se han realizado las fases de identificación y planificación como parte del ciclo de vida para el desarrollo de *software*.

Referencias

Adewumi, A., Misra, S., Omoregbe, N., Crawford, B., & Soto, R. (2016). A systematic literature review of open source software quality assessment models. *SpringerPlus*, *5*(1), 1–13. https://doi.org/10.1186/s40064-016-3612-4

Alebeisat, F., Alhalhouli, Z., & Alshabatat, T. E. (2018). Review of Literature on Software Quality Software Quality View project Project in "WIRELESS NETWORK" View project. *W*, *8*(October), 32–42. https://www.researchgate.net/publication/328495821

Andrew, A. M. (1998). Software quality engineering. In *Kybernetes* (Vol. 27, Issue 4). https://doi.org/10.1108/k.1998.27.4.457.4

Basili, V. R., & Musa, J. D. (2002). The future generation of software: a management perspective. In *Computer* (Vol. 24, Issue 9, pp. 90–96). https://doi.org/10.1109/2.84903

Buchanan, B. G., & Smith, R. Q. (1988). Fundamentals of expert system. *Springer Series in Materials Science*, *206*, 31–39. https://doi.org/10.1007/978-3-662-44497-9_3

Ceran, A. A., Ar, Y., Tanrıöver, Ö., & Seyrek Ceran, S. (2022). Prediction of software quality with Machine Learning-Based ensemble methods. *Materials Today: Proceedings*, *xxxx*. https://doi.org/10.1016/j.matpr.2022.11.229

Collins, W. R., Miller, K. W., Spielman, B. J., & Wherry, P. (1994). How good is good enough?: An Ethical Analysis of Software Construction and Use. *Communications of the ACM*, *37*(1), 81–91. https://doi.org/10.1145/175222.175229

Flores Zafra, D., & Gardi Melgarejo, V. (2020). Sistema experto para la SGTI en la empresa Sion Global Solutions. *INNOVA Research Journal*, *5*(3.2), 235–248. https://doi.org/10.33890/innova.v5.n3.2.2020.1568

Gao, J., Tao, C., Jie, D., & Lu, S. (2019). Invited paper: What is ai software testing? and Why.

Proceedings - 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, 10th International Workshop on Joint Cloud Computing, JCC 2019 and 2019 IEEE International Workshop on Cloud Computing in Robotic Systems, CCRS 2019, 27–36. https://doi.org/10.1109/SOSE.2019.00015

Garousi, V., & Mäntylä, M. V. (2016). A systematic literature review of literature reviews in software testing. *Information and Software Technology*, *80*, 195–216. https://doi.org/10.1016/j.infsof.2016.09.002

Grogono, P., Preece, A. D., Shinghal, R., & Suen, C. Y. (1993). A review of expert systems evaluation techniques. *AAAI Technical Report WS-93-05*, *January 1993*, 113–118. http://www.aaai.org/Papers/Workshops/1993/WS-93-05/WS93-05-016.pdf

Groundwater, E. ., Miller, L. A., & Mirsky, S. M. (1995). *Verification and validation of expert systems software and conventional software* (p. 114). https://inis.iaea.org/collection/NCLCollectionStore/_Public/26/060/26060098.pdf

Guerra, V. A., Rodriguez, G. A., & Aruquipa, M. (2014). Modelo de sistema experto para la auditoria de sistemas informáticos. *UNIVERSIDAD MAYOR DE SAN ANDRÉS; Tesis Maestria*, *AF4*, 85-87.

Hauge, O., Britos, P., & García-Martínez, R. (2006). Conceptualization maturity metrics for expert systems. *IFIP International Federation for Information Processing*, *217*(August), 435–444. https://doi.org/10.1007/978-0-387-34747-9_45

Herrera, E. M., & Ramírez, R. A. T. (2003). a Methodology for Self-Diagnosis for Software Quality Assurance in Small and Medium-Sized Industries in Latin America. *9th Americas Conference on Information Systems, AMCIS 2003*, 1229–1237. https://doi.org/10.1002/j.1681-4835.2003.tb00100.x

Karg, L. M., Grottke, M., & Beckhaus, A. (2011). A systematic literature review of software quality cost research. *Journal of Systems and Software*, *84*(3), 415–427. https://doi.org/10.1016/j.jss.2010.11.904

Kaur, A. (2020). A Systematic Literature Review on Empirical Analysis of the Relationship Between Code Smells and Software Quality Attributes. *Archives of Computational Methods in Engineering*, *27*(4), 1267–1296. https://doi.org/10.1007/s11831-019-09348-6

Kumbakonam, M. (2021). A Comparison Experiment On Software Quality Testing With Machine Learning Using Different Algorithms. In *Turkish Journal of Computer and Mathematics …*. https://www.turcomat.org/index.php/turkbilmat/article/view/7259%0Ahttps://www.turcomat.org/index.php/turkbilmat/article/download/7259/5901

Kumbakonam, Munidhanalakshmi, & Shafi, R. M. (2021). Application measurement and software quality testing using machine learning performance techniques. *Proceedings - 2021 4th International Conference on Computational Intelligence and Communication Technologies, CCICT 2021*, *July*, 372–376. https://doi.org/10.1109/CCICT53244.2021.00074

León Martínez, N. E., Mendoza Castellanos, A., & Gómez Flórez, L. C. (2011). *Propuesta de un sistema para la evaluación de calidad de software derivado de actividades de investigación*. 1–84.

Madou, F., Agüero, M., Esperón, G., & López de Luise, M. D. (2009). Sistemas Expertos en Evaluación de Calidad Java. *Articulo*, 1–7. https://www.researchgate.net/profile/Daniela-Lopez-De-Luise/publication/267305785_Sistemas_Expertos_en_Evaluacion_de_Calidad_Java/links/5

489b48e0cf214269f1ab888/Sistemas-Expertos-en-Evaluacion-de-Calidad-Java.pdf

Martín, E., & Sáenz, F. (2016). Sistema ANACONDA para el análisis automático de la calidad del software. *Telefónica Móviles España*.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in Software Quality - Volume 1 - Concept and Definitions of Software Quality. *Defense Technical Information Center*, *1*, *2* and *3*(ADA049014), 168. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA049014

Mendoza, Luis E, Pérez, M. E. S. B. (2005). Prototipo de modelo sistémico de calidad (MOSCA) del software. *Computación y Sistemas*, *8 Núm 3*, 196–217. http://www.scielo.org.mx/pdf/cys/v8n3/v8n3a5.pdf

Meseguer, P. (1991). *Verification of Multi-level Rule-based Expert Systems*. 323–328.

Moreno, J. J., Bolaños, L. P., & Navia, M. A. (2010). Exploración de modelos y estándares de calidad para el producto software. *Revista UIS Ingenierías*, *9*(1), 39–53.

Nguyen-Duc, A., Cruzes, D. S., & Conradi, R. (2015). The impact of global dispersion on coordination, team performance and software quality-A systematic literature review. *Information and Software Technology*, *57*(1), 277–294. https://doi.org/10.1016/j.infsof.2014.06.002

Ortega, M., Perez, M. A., & Rojas, T. (2003). CONSTRUCTION OF A SYSTEMIC QUALITY MODEL FOR EVALUATING A SOFTWARE PRODUCT. *Entomologia Experimentalis et Applicata*, *103*(3), 239–248. https://doi.org/10.1023/A

P. Miguel, J., Mauricio, D., & Rodríguez, G. (2014). A Review of Software Quality Models for the Evaluation of Software Products. *International Journal of Software Engineering & Applications*, *5*(6), 31–53. https://doi.org/10.5121/ijsea.2014.5603

Rothenberg, J., Paul, J., Kameny, I., Kipps, J. R., & Swenson, M. (1987). *Evaluating Expert Systems: A framework and Methodology*. 1–23.

Schulmeyer, G. G. (2008). Handbook of Software Quality Assurance. In *Artech House*.

Singh, B., & Kannojia, S. P. (2013). A review on software quality models. *Proceedings - 2013 International Conference on Communication Systems and Network Technologies, CSNT 2013*, 801–806. https://doi.org/10.1109/CSNT.2013.171

SinghThapar, S., Singh, P., & Rani, S. (2012). Challenges to Development of Standard Software Quality Model. *International Journal of Computer Applications*, *49*(10), 1–7. https://doi.org/10.5120/7660-0765

Stephen, H. K., & Addison, W. (2002). Metrics and Models in Software Quality Engineering. In *ACM SIGSOFT Software Engineering Notes* (Vol. 21, Issue 1). https://doi.org/10.1145/381790.565681

Sun, J. L. (1988). *Application of software engineering and quality assurance to expert systems development.* 547–553. https://doi.org/10.1109/icc.1988.13625

Suryn, W. (2014). Software Quality Engineering: A Practitioner's Approach. In *Software Quality Engineering: A Practitioner's Approach* (Vol. 9781118592496). https://doi.org/10.1002/9781118830208

Toro, A., & Peláez, L. (2018). Validación de un modelo para el aseguramiento de la calidad del software en MIPYMES que desarrollan software en el Eje Cafetero. *Entre Ciencia e Ingeniería*, *12*(23), 84–92. https://doi.org/10.31908/19098367.3707

Tosun, A., Bener, A. B., & Akbarinasaji, S. (2017). A systematic literature review on the applications of Bayesian networks to predict software quality. *Software Quality Journal*, *25*(1), 273–305. https://doi.org/10.1007/s11219-015-9297-z

Tucupa, T. C. (2020). *Modelo para identificar defectos en Control de Calidad , basado en Oráculos de pruebas y Sistemas Expertos*. 193–195.

Wedyan, F., & Abufakher, S. (2020). Impact of design patterns on software quality: A systematic literature review. *IET Software*, *14*(1), 1–17. https://doi.org/10.1049/iet-sen.2018.5446

Winkler, D., Biffl, S., Mendez, D., Wimmer, M., & Bergsmann, J. (2021). Software Quality Future Perspectives on Software Engineering Quality. In *Proceedings*. http://www.springer.com/series/7911