Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE

Julio César Ayapán Culajay
Asesorado por el Ing. Lenin Fernando Rodríguez Conde

Guatemala, enero de 2014

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

CARACTERÍSTICAS DE CALDIAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

JULIO CÉSAR AYAPÁN CULAJAY
ASESORADO POR EL ING. LENIN FERNANDO RODRÍGUEZ CONDE

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, ENERO DE 2014

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Inga. Elvia Miriam Ruballos Samayoa
VOCAL IV	Br. Walter Rafael Véliz Muñoz
VOCAL V	Br. Sergio Alejandro Donis Soto
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
EXAMINADOR	Ing. Marlon Francisco Orellana López
EXAMINADOR	Ing. José Alfredo González Díaz
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha enero de 2013.

Julio César Ayapán Culajay

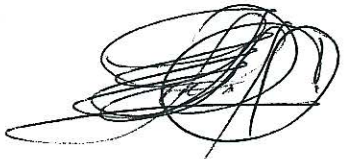
Guatemala, 28 de octubre de 2013

Señores
Comisión de revisión de trabajo de graduación
Carrera de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería
Universidad de San Carlos de Guatemala
Guatemala, Ciudad

Respetables señores:

Por este medio deseo comunicarles que el Sr. Julio César Ayapán Culajay quien se identifica con el carne 200714506 ha finalizado su trabajo de graduación titulado "CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE". El criterio de finalización fue otorgado debido a que: 1) el trabajo de graduación fue realizado bajo los términos y condiciones estipulados durante la asesoría, 2) los temas básicos y necesarios para generar las conclusiones del trabajo de graduación fueron generados y documentados, 3) las conclusiones obtenidas están directamente relacionadas a los objetivos del trabajo de graduación, 4) se han documentado las recomendaciones necesarias para que el trabajo de graduación sea el punto de partida para investigaciones futuras.

Agradecido por su atención



Ing. Lenin Fernando Rodríguez Conde
Col. 10694

Ing. Lenin Fernando Rodríguez Conde
Asesor de trabajo de graduación
Colegiado: 10694

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 6 de Noviembre de 2013

Ingeniero
Marlon Antonio Pérez Turk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **JULIO CÉSAR AYAPÁN CULAJAY,** con camé **2007-14506,** titulado: **"CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE",** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA - FACULTAD DE INGENIERÍA EN CIENCIAS Y SISTEMAS - COORDINACION TESIS

ESCUELA DE CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE"****, realizado por el estudiante JULIO CÉSAR AYAPÁN CULAJAY, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Türk
Director, Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 22 de enero 2014

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref.DTG.019-2014

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ciencias y Sistemas, al trabajo de graduación titulado: **CARACTERÍSTICAS DE CALIDAD APLICADAS A REUTILIZACIÓN DE SISTEMAS DE SOFTWARE,** presentado por el estudiante universitario: **Julio César Ayapán Culajay,** procede a la autorización para la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
Decano

DECANO
FACULTAD DE INGENIERIA

Guatemala, enero 2014

/cc

ACTO QUE DEDICO A:

Mis padres	Julio Ayapán y Enma Culajay, por ser la base fundamental de mis logros y brindarme apoyo incondicional durante todos estos años.
Mis hermanos	Karen, Kevin y Allan Ayapán, por estar junto a mí en todo momento y brindarme su apoyo para lograr esta meta.
Mis tías	Gloria y Rosa Ayapán, por brindarme su apoyo a lo largo de toda mi carrera y brindarme un cariño tan especial.

AGRADECIMIENTOS A:

Mis padres	Julio Ayapán y Enma Culajay, por creer en mí en todo momento y darme la fuerza para continuar siempre.
Universidad de San Carlos de Guatemala	Por abrirme sus puertas a esta grandiosa casa de estudios.
Mis amigos	A todos aquellos que de alguna manera colaboraron en el presente trabajo y, que su apoyo incondicional es parte de la historia durante toda mi carrera.
Asesor	Ing. Lenin Fernando Rodríguez, por estar ahí para asesorarme en este trabajo de graduación.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES V
LISTA DE SÍMBOLOS VII
GLOSARIO IX
RESUMEN XI
OBJETIVOS XIII
INTRODUCCIÓN XV

1. ESPECIFICACIÓN DE LA VIABILIDAD DEL SOFTWARE 1
   1.1. Fiabilidad 1
      1.1.1. Fiabilidad del software 3
      1.1.2. Fiabilidad del hardware 4
      1.1.3. Fiabilidad del operador 4
   1.2. Disponibilidad 5
      1.2.1. Métricas de fiabilidad 6
      1.2.2. Probabilidad de falla en demanda 7
      1.2.3. Tasa de ocurrencia de fallos 8
   1.3. Análisis de riesgo de fallos 8
   1.4. Costos de calidad en el desarrollo 10
      1.4.1. Costos de prevención 11
      1.4.2. Costos de evaluación 11
      1.4.3. Costos de fallas internas 12

2. CALIDAD DEL SOFTWARE Y EL CAMPO DE LA REUTILIZACIÓN 13
   2.1. ¿Qué es la calidad del software? 13
   2.2. Modelos de calidad de software 13

2.2.1. Modelo de Boehm 13
2.2.2. Modelo goal question metric (GQM) 15
2.2.3. Modelo de calidad funcionality, usability, reliability, performance, supportability (FURPS) 17
2.2.4. Modelo de calidad web quality evaluation method (WebQEM) 18
2.3. El campo de la reutilización 18
2.3.1. ¿Qué es la reutilización? 19
2.4. Métodos de reutilización 20
2.4.1. Reutilización basada en patrones de diseño 23
2.4.2. Reutilización basada en generadores 25
2.4.3. Reutilización basada en marcos de trabajo 28
2.5. Ventajas de la reutilización 29
2.6. Desventajas de la reutilización 30
2.7. Atributos que definen la calidad del software 32
2.7.1. Modelo ISO/IEC 9126-1:2001 32
2.8. ¿La reutilización de componentes asegura la calidad? 35

3. INGENIERÍA DE SOFTWARE BASADA EN COMPONENTES 37
3.1. Componentes de software 37
3.2. Atributos de los componentes 39
3.3. Modelos de calidad de componentes 42
3.3.1. Modelo ISO/IEC 9126-1 42
3.3.2. Atributos de calidad de los componentes software 43
3.4. ¿Cómo evaluar la calidad de un componente? 52
3.5. Composición de componentes 52

4. CALIDAD EN LA INGENIERÍA DEL SOFTWARE BASADA EN COMPONENTES 57
4.1. Ingeniería del software basada en componentes 57
4.2. Aspectos de calidad en la ingeniería del software basada en componentes 60
4.3. Propuestas de calidad en la ingeniería del software basada en componentes 60
4.4. Reutilización de componentes 63
4.5. Análisis y diseño para la reutilización 64
4.6. Modelos de desarrollo basado en componentes 66
4.7. Especialización de productos software 68
4.7.1. Herramientas enterprise resource planning (ERP) 68
4.7.2. Herramientas customer relationship management (CRM) 70
4.7.3. Herramientas business process management (BPM) 71

5. IMPACTO EN LA CALIDAD DE LOS SISTEMAS BASADOS EN COMPONENTES 73
5.1. Análisis estadístico de resultados sobre el impacto en la calidad de la ingeniería del software basada en componentes 74
5.1.1. Estadísticos descriptivos 75
5.1.2. Relaciones entre variables discriminantes y valorativas 78
5.1.3. Determinación de Alpha de Cronbach 80
5.2. Aproximación a un modelo de calidad para ISBC 84
5.3. Medición del impacto en la calidad 87

5.3.1. Regresión lineal multivariable....91
5.3.2. Explicación del modelo lineal 92
5.3.3. Error estándar de la estimación....93
5.3.4. Coeficiente de determinación del modelo 93
5.3.5. Pruebas de hipótesis global e individual....94
5.3.6. Media poblacional y diagnósticos por caso 95

CONCLUSIONES....99
RECOMENDACIONES 101
BIBLIOGRAFÍA....103
APÉNDICES....105

ÍNDICE DE ILUSTRACIONES

FIGURAS

1. Implementación probabilidad de falla en demanda 8
2. Modelo de calidad de Boehm 14
3. Modelo de calidad goal question metric (GQM) 16
4. Modelo de calidad ISO/IEC 9126-1:2001 33
5. Representación esquemática de un componente software 38
6. Clasificación de la composición de un componente 53
7. Configuración de un sistema ERP 69

TABLAS

I. Errores y fallos de sistema 2
II. Métricas para especificar la fiabilidad y disponibilidad del software 7
III. Modelo ISO/IEC 9126-1 42
IV. Modelo de calidad para componentes 44
V. Modelo de calidad basado en la arquitectura 60
VI. Modelo de calidad de Jan Bosch 62
VII. Análisis estadístico para variables discriminativas 75
VIII. Estadísticos descriptivos para todas las variables 76
IX. Matriz de correlaciones 79
X. Alpha de Cronbach sin la variable edad 80
XI. Fiabilidad de variables contra variable sexo 81
XII. Fiabilidad de variables contra variable área de trabajo 83
XIII. Validación del modelo 90

XIV. Modelo lineal....91
XV. Pruebas de hipótesis del modelo lineal....93
XVI. Anova del modelo....94
XVII. Diagnósticos por caso....96

LISTA DE SÍMBOLOS

Símbolo	Significado
&&	And*, o*perador lógico Y
@	Arroba, signo utilizado en informática como un separador de dominios
>=	Mayor o igual que
<=	Menor o igual que
||	Or**,** operador lógico O

GLOSARIO

Active server page Tecnología internet para la programación estilo interfaz de Microsoft, usado en servicios de información.

Adl Architecture definition language, define la arquitectura de una aplicación.

Artefacto Módulo de software.

Bpm Business process model, sistema de software para gestión de procesos.

Crm Customer relationship management, software para la administración de la relación con clientes.

Furps Funcionality, usability, reliability, performance, supportability. Representa un modelo de calidad del software.

Gqm Goal question metric, representa el modelo de calidad de software basado en métricas.

IEC Comisión Electrotécnica Internacional.

RESUMEN

El entorno de la ingeniería del software es cambiante y va enfocado a realizar una solución con base en los conceptos más simples y eficientes, logrando productos de calidad en los tiempos y costos deseados.

La calidad de un sistema software está orientada al nivel de satisfacción del cliente en aspectos como: funcionamiento, rendimiento y facilidad de uso.

Uno de los mecanismos más efectivos para la construcción de grandes sistemas, actualmente es la ingeniería del software basada en componentes, que encapsula todo el ciclo de vida de un sistema, desde el análisis de requerimientos hasta el mantenimiento durante la etapa de vida. Uno de los grandes problemas que existen para este mecanismo es la falta de métricas que definan la calidad de este tipo de sistemas, para lo cual es necesario comenzar desde la definición de calidad en un componente y como la aplicación de estos componentes impacta en la calidad de un sistema software hecho a la medida.

Aunque existen en la actualidad modelos que definan calidad de software y calidad en componentes, no existe un modelo capaz de definir la calidad en la aplicación de componentes software que cumplan con los estándares o que se aproximen a un modelo normalizado definido.

OBJETIVOS

General

Identificar el impacto que las características de calidad de un componente, generan sobre un sistema de software basado en componentes.

Específicos

1. Identificar atributos que determinan la calidad de un componente software.

2. Identificar el conjunto de cualidades que componen la calidad del software basado en componentes.

3. Identificar cómo los atributos de calidad de componentes de software impactan en las características del software.

4. Conocer cómo llevar a cabo el diseño de procesos que proporcionen valor a las organizaciones por medio de herramientas reutilizables como business process management y enterprise resource planning.

INTRODUCCIÓN

Se vive en una nueva era económica, que ha sido también llamada: La Era de la Informática o La Era del Conocimiento. Sin embargo, en la vida diaria de los negocios, lo importante para el desarrollo y el crecimiento de la empresa en este sentido no es solamente el acceso a la información, ni la búsqueda del conocimiento por el conocimiento mismo, sino la capacidad creciente de crear conocimientos nuevos y, por ende, la capacidad de aprender a aprender cada vez con mayor rapidez. De aquí el énfasis en las técnicas para obtener lo mejor de sistemas existentes y adaptarlos a un nuevo sistema que satisfaga las necesidades de una organización.

El proceso de diseño en la mayoría de las disciplinas de ingeniería se basa en la reutilización de sistemas o componentes existentes. Los ingenieros mecánicos o eléctricos no especifican, normalmente un diseño en el que cada componente tenga que ser fabricado de una forma especial. Basan su diseño en componentes que han sido utilizados y probados en otros sistemas.

La ingeniería del software basada en componentes, es una estrategia de ingeniería comparable en la que el proceso de desarrollo es adaptado a la reutilización del software existente.

La tendencia hacia el desarrollo de aplicaciones completas basadas en componentes, viene dada como respuesta a las demandas de una menor producción de software y de menores tiempos de mantenimiento, además de un incremento de la calidad del software.

1. ESPECIFICACIÓN DE LA VIABILIDAD DEL SOFTWARE

Los productos de software poseen atributos asociados que reflejan y describen su calidad. Estos atributos no se encuentran directamente asociados a lo que el software hace, sino reflejan su comportamiento durante su ejecución y en la estructura, y organización del programa fuente y en la documentación asociada.

Entre los atributos de un sistema se pueden considerar el tiempo de respuesta del software a una pregunta del usuario, y la comprensión que se puede tener al leer el programa fuente por cualquier tercero, estos atributos son medibles y existen dependiendo de la aplicación a la que esté enfocada el sistema, generando de cada uno, el conjunto específico de características.

1.1. Fiabilidad

La fiabilidad del software puede ser estimada mediante datos históricos o de desarrollo, y representa la probabilidad de que el sistema funcione correctamente como tal y como se ha especificado. Para expresar la disponibilidad y fiabilidad de un sistema, se han utilizado probabilidades numéricas que permiten dar un valor aproximado de estas dos propiedades que están estrechamente relacionadas.

La fiabilidad y disponibilidad de un sistema se pueden definir de forma más precisa como sigue:

- Fiabilidad: la probabilidad de que un sistema funcione adecuadamente (operaciones libres de caídas) durante un tiempo definido y bajo condiciones operaciones específicas.

- Disponibilidad: probabilidad de que un sistema esté en funcionamiento y tenga la capacidad de proporcionar respuesta y servicios solicitados en un cierto momento.

La fiabilidad y la disponibilidad están relacionadas con los fallos de funcionamiento del sistema. Estos pueden ser un fallo al proporcionar un servicio; un fallo provocado por la forma en que se proporciona dicho servicio o la prestación de un servicio, de modo que este sea inseguro o no protegido. Cuando se analiza la fiabilidad, es útil distinguir entre los términos defecto, error y fallo.

Tabla I. **Errores y fallos de sistema**

Error	Descripción
Fallo del sistema	Evento que tiene lugar en algún instante cuando el sistema no funciona como esperan sus usuarios.
Error del sistema	Estado erróneo del sistema que puede dar lugar a un comportamiento del mismo inesperado por sus usuarios.
Defecto del sistema	Característica de un componente de software que puede dar lugar a un error del sistema. Por ejemplo, un fallo en la ejecución al inicializar una variable puede hacer que dicha variable tenga un valor incorrecto cuando sea usada.

Fuente: elaboración propia.

La diferencia entre los términos mostrados ayuda a definir tres enfoques complementarios, usados para mejorar la fiabilidad de un sistema.

- Evitación de defectos: por medio de técnicas se busca minimizar la probabilidad de cometer equivocaciones o detectarlas antes de que provoquen la introducción de defectos en un sistema.

- Detección y eliminación de defectos: se utilizan técnicas de verificación que se aplican antes de la utilización de sistema, estas aumentan la probabilidad de detectar y eliminar defectos.

- Tolerancia a defectos: técnicas y procedimientos cuyo objetivo es asegurar que los defectos existentes en un sistema no conduzcan a errores del sistema o que los errores del sistema no den lugar a fallos en el funcionamiento interno.

Considerando un sistema de computadora, una sencilla medida de la fiabilidad es el tiempo medio entre fallos, donde TMEF = TMDF + TMDR.

TMDF corresponde al tiempo medio de fallo mientras que TMDR al tiempo medio de reparación.

1.1.1. Fiabilidad del software

El concepto de fiabilidad está ligado al término fallo. Los defectos del software ocasionan fallos de funcionamiento del sistema, estos se ponen de manifiesto cuando un conjunto de entradas es ejecutado mediante el código con defectos. El código funciona correctamente para la mayoría de las entradas.

El conjunto de entradas que provocan salidas erróneas durante la ejecución normal del sistema sin hacer distinción del usuario que las efectúa define entonces, la fiabilidad de un programa. No se debe tomar en cuenta para

el cálculo de esta característica, aquellas partes del sistema que se utilizan raramente, dado que aun eliminando estos defectos de software, hace que haya poca diferencia real percibida por los usuarios del sistema; considerando que la mayoría de estos defectos provocarán, probablemente fallos después de meses de utilizar el producto. En términos generales se mejora únicamente un 3 por ciento de la fiabilidad de un sistema mediante la eliminación de un 60 por ciento de errores conocidos en un software.

1.1.2. Fiabilidad del hardware

Los primeros trabajos sobre fiabilidad intentaron extrapolar las matemáticas de la teoría de la fiabilidad del hardware a la predicción de la fiabilidad del software. Las medidas de fiabilidad enfocadas en el concepto de hardware están orientadas a los fallos surgidos debidos al desajuste de los componentes que a los fallos de diseño de un elemento físico. En el hardware son más probables los fallos debidos al desgaste físico, que los fallos relativos al diseño. Para el software lo que ocurre es lo contrario, todos sus fallos se producen por problemas de diseño o de implementación. Todavía se debate sobre la relación entre los conceptos clave de la fiabilidad del hardware y su aplicación al software.

1.1.3. Fiabilidad del operador

Las percepciones y patrones humanos de utilización, también son significativos en la medición de la fiabilidad de un sistema, el operador no tiene en cuenta, ni la gravedad de los fallos, ni las consecuencias de la ausencia de disponibilidad, naturalmente, se preocupan más por los fallos del sistema que tienen consecuencias graves y, la percepción de su fiabilidad se verá influenciada por dichas consecuencias.

Los fallos de funcionamiento del sistema no son provocados por la interacción incorrecta de usuario con la aplicación, los errores humanos no deberían de afectar el funcionamiento, los errores son netamente de desarrollo, estos defectos pueden estar en partes del sistema que nunca han sido usadas.

Cada usuario de un sistema lo usa de diferentes formas los defectos (que resultan de fallos en un sistema) que afectan a la fiabilidad de una aplicación en la operación de un usuario, puede que en ninguna otra ocasión se manifiesten bajo otro modo de trabajo para un nuevo usuario. Los usuarios se adaptan a un software con defectos ya conocidos, estos mismos usuarios pueden compartir información respecto a los errores y sobre cómo esquivar problemas en la ejecución. Evitando problemas, que se sabe se generarán con un conjunto de entradas específicas y de esta manera los fallos de funcionamiento no tendrán lugar en el sistema o no dependerán del factor humano. El mismo caso se aplica a usuarios experimentados que tienden a eludir defectos del sistema que ya saben producirán fallos de funcionamiento.

1.2. Disponibilidad

Aunque los conceptos de fiabilidad y disponibilidad guardan una estrecha relación, no se puede deducir que los sistemas fiables estarán siempre disponibles y viceversa. Por ejemplo, algunos sistemas pueden tener como requisito una disponibilidad alta pero una fiabilidad mucho más baja. Si los usuarios esperan un servicio continuo, entonces los requerimientos de disponibilidad son altos. Los requerimientos de fiabilidad serán bajos en el caso de que las consecuencias de un fallo de funcionamiento sean significativas y el sistema logre recuperarse rápidamente de los fallos.

Una diferencia adicional entre estas características es que la disponibilidad no depende, simplemente de un sistema en sí, sino también del tiempo necesario para reparar los conflictos que hicieron que el sistema dejara de estar disponible.

Además de una medida de fiabilidad, se debe obtener una medida de la disponibilidad. La disponibilidad del software es la probabilidad de que un programa funcione de acuerdo a los requisitos en un momento dado y se define como:

Disponibilidad = [TMDF / (TMDF + TMDR)] x 100.

La medida de fiabilidad TMEF es igualmente sensible al TMDF que al TMDR. La medida de disponibilidad es algo más sensible al TMDR, una medida indirecta de la facilidad del mantenimiento del software.

1.2.1. Métricas de fiabilidad

Las fórmulas presentadas en la definición de fiabilidad y disponibilidad no son directamente aplicables a la especificación de la fiabilidad del software, debido a que los fallos de estos componentes resultan ser a veces transitorios y no permanentes. Los errores solamente se manifiestan con algunas entradas, si los datos no están dañados, a menudo el sistema puede continuar funcionando después de que ocurra un fallo.

Tabla II. **Métricas para especificar la fiabilidad y disponibilidad del software**

Métrica	Descripción
Probabilidad de falla en demanda (POFOD)	La probabilidad de que el sistema falle cuando se solicite una petición de servicio. Un POFOD de 0,001 significa que 1 de cada 1 000 peticiones de servicio conducen a un fallo.
Tasa de ocurrencia de fallos (ROCOF)	La frecuencia de ocurrencia con la que es probable que ocurra un comportamiento inesperado. Un ROCOF de 2/100 significa que es probable que ocurran dos fallos en cada cien unidades de tiempo de funcionamiento.
Tiempo medio entre fallos (MTFF)	El tiempo medio entre fallos de sistema observados. Un MTTF de 500 significa que puede esperarse un fallo cada 500 unidades de tiempo.
Disponibilidad (*AVAIL*)	La probabilidad de que el sistema esté disponible para su utilización en un momento determinado. Una disponibilidad de 0,998 significa que el sistema es probable que esté disponible durante 998 de cada 1 000 unidades de tiempo.

Fuente: elaboración propia.

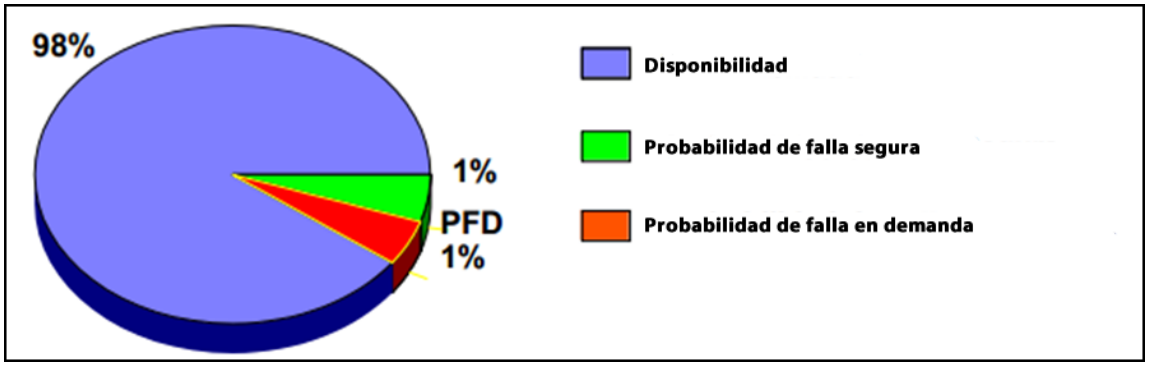
1.2.2. Probabilidad de falla en demanda

Esta técnica busca obtener una unidad de medida que defina el tiempo de respuesta de un sistema y es aplicable a aquellos en los que los servicios se demandan a intervalos de tiempo impredecibles o relativamente largos, y en los que existe una severa consecuencia si el servicio que se solicitó, no se proporciona.

La IEC61508 define la Probabilidad de Falla en Demanda (POFOD, Probability Of Failure on Demand), como la probabilidad estadística de que un sistema falle en forma peligrosa. Las POFOD son utilizadas para la fijación y

determinación de los niveles de integridad, para los cuales corresponderá un factor de reducción de riesgo o FRR (FRR = 1/PFD).

Figura 1. **Implementación probabilidad de falla en demanda**



Fuente: elaboración propia.

1.2.3. Tasa de ocurrencia de fallos

Esta métrica debería utilizarse en donde se hagan peticiones regulares de los servicios del sistema y donde sea importante que estos servicios se proporcionen correctamente.

Se puede definir como el número de acontecimientos inesperados sobre un rato particular de la operación de un sistema o la frecuencia de ocurrencia, con la cual un comportamiento inesperado es probable que ocurra.

1.3. Análisis de riesgo de fallos

El análisis de riesgos es una actividad clave en el proceso de especificación de sistemas críticos. Este análisis requiere identificación de riesgos que puedan provocar accidentes o incidentes en un futuro. Basados en

el análisis de estos riesgos se establecen los requerimientos del sistema, asegurando la menor probabilidad de que estos ocurran y, si en caso ocurren, que no provoquen un incidente grave.

El análisis de riesgos es el proceso de valorar la probabilidad de que un riesgo provoque un accidente.

El primer paso, denominado identificación del riesgo, reconoce la opción de que algo puede ir mal. A partir de esto, cada riesgo se analiza por separado para determinar la probabilidad de que ocurra y el alcance del daño que pueda ocasionar (en caso suceda). Con la definición de probabilidades de cada riesgo se priorizan según impacto. El último paso consiste en desarrollar un plan de gestión para aquellos que tengan la probabilidad de ocurrencia y el impacto más alto.

Los sistemas grandes, generalmente se componen de varios subsistemas y cada uno de estos con diferentes requerimientos de fiabilidad. Dado que un sistema es grande, tiene una fiabilidad alta. El análisis de fiabilidad debe realizarse por cada subsistema para evitar imponer los mismos requerimientos de fiabilidad a todos, es decir, evaluarlos por separado. Esto evita imponer altos requerimientos de fiabilidad en aquellos subsistemas en los que no es necesario.

Los pasos que se requieren para establecer una especificación de la fiabilidad son:

- Para cada subsistema: análisis de riesgos y tipos de fallos de funcionamiento de sistema y análisis de consecuencias que puedan generar los riesgos identificados.

- A partir del análisis de fallos del sistema, dividir los fallos en clases.

- Ya con los fallos divididos, se define el requerimiento de fiabilidad utilizando la métrica adecuada para cada uno. ROCOF se recomienda para el caso de recuperación automática y cuando el efecto del fallo genera molestia para el usuario.

- Según sea el caso en cada clase de fallo, es necesario identificar requerimientos de fiabilidad que definan la funcionalidad del sistema y reducir de esta manera la probabilidad de fallos críticos en la ejecución del sistema.

1.4. Costos de calidad en el desarrollo

La gestión de un proyecto informático empieza con la calificación del proyecto, con la cual se pretende obtener una idea del esfuerzo (trabajo y tiempo) que costará construir la aplicación, denominada estimación y elaborar una planificación del tiempo para las diferentes actividades que son necesarias ejecutar(planificación).

La mayor parte del costo del software se encuentra, actualmente en el costo de las horas de análisis, diseño, programación y prueba que se deben utilizar para obtenerlo.

Debido a esto, al hablar de estimación de costos se hace referencia al esfuerzo humano que se requiere, es decir, las horas de trabajo y esfuerzo necesario para construir el software.

1.4.1. Costos de prevención

Se refiere a todos los costos asociados a esfuerzos realizados para garantizar la calidad del software y prevenir un efecto negativo posterior en todas las fases del desarrollo de software.

Entre estos costos se encuentran involucradas las siguientes etapas:

- Aseguramiento de la calidad: planeación de la calidad, mejora de procesos, definición de procesos, políticas y estándares, obtención, análisis y uso de datos sobre la calidad, análisis de causas raíces.

- Requerimientos: especificaciones y prototipos.

- Administración del proyecto: planeación, capacitación, recopilación de métricas.

- Administración de la configuración: capacitación y herramientas.

1.4.2. Costos de evaluación

Costos realizados para calcular la condición de la calidad final del software por medio de evaluaciones planeadas. Las etapas para esto son:

- Evaluación de proyectos: revisión de especificaciones de requerimientos, diseño y componentes, verificaciones y validaciones en general, inspecciones, pruebas unitarias, de integración y de sistema.

- Auditorías de calidad del producto

- Evaluaciones externas

- Pruebas de productos adquiridos

1.4.3. Costos de fallas internas

Costos que se atribuyen a realizar un escaneo completo para detectar y corregir problemas antes de que el usuario los detecte mediante la ejecución de la aplicación.

El costo final de calidad es medido con los costos totales que podrían desaparecer si no existieran defectos. Se incluyen en este rubro todos los costos asociados a evaluación, prevención y fallas internas y externas.

2. CALIDAD DEL SOFTWARE Y EL CAMPO DE LA REUTILIZACIÓN

El desarrollo de software se ha convertido en uno de los principales problemas que tiene que afrontar la ingeniería del software.

2.1. ¿Qué es la calidad del software?

Se define como el conjunto de cualidades que aportan utilidad y determinan la existencia de un software.

Las características que aportan calidad varían de un sistema a otro según los requerimientos iniciales, por ejemplo: si se va a construir un sistema para el control de semáforos este debe ser confiable a nivel de cero fallas; mientras que si se va a construir un sistema ERP para implementación durante un largo periodo de tiempo, no requiere el mismo nivel de calidad en cuanto a fallas, pero debe ser confiable y mantenible para soportar cambios constantes.

2.2. Modelos de calidad de software

La calidad del Software es un conjunto de cualidades medibles y cada modelo describe diferentes métricas.

2.2.1. Modelo de Boehm

Representa la calidad total mediante una jerarquía de características. Los tres niveles que manipula este modelo están distribuidos en:

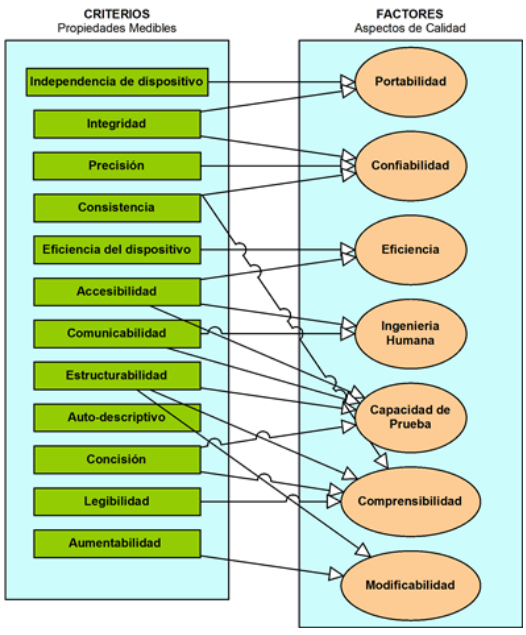
- Características de alto nivel
- Características de nivel intermedio
- Características primitivas

El objetivo de este modelo es que a través de la calidad del software un sistema realice lo que el usuario solicite.

Las características de alto nivel representan requerimientos generales de uso como: cuan usable, confiable y eficiente es el producto; que tan mantenible puede ser y la utilidad general aplicada a diferentes ambientes.

Las características de nivel intermedio (características de calidad) serán medidas a través de las características primitivas (métricas de calidad).

Figura 2. **Modelo de calidad de Boehm**



Fuente: elaboración propia.

2.2.2. Modelo goal question metric (GQM)

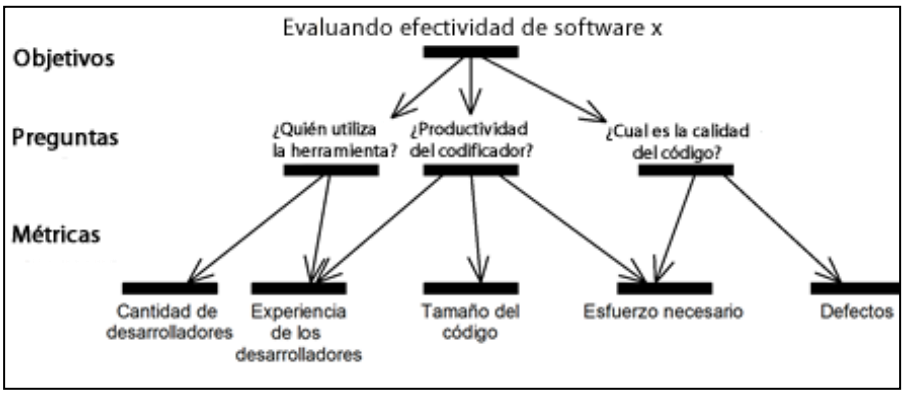
Está basado en el enfoque con el mismo nombre. Este enfoque se basa en la suposición de que una organización necesita, para medir adecuadamente sus resultados, identificar las metas que desea, establecer un modelo cuantificable de objetivos y definir un marco que permita interpretar la información respecto a estos objetivos. El enfoque de GQM basa la mejora en la definición clara de procesos y productos.

El modelo GQM a través de una propuesta define un modelo de calidad hasta obtener las métricas necesarias con el análisis e interpretación de los datos de las mediciones.

La estructura del modelo GQM coloca al objetivo en la parte superior, a partir de este define los efectos de la medición, cómo debe medirse y el punto de vista de donde se toma la medida.

El siguiente diagrama explica el modelo de calidad GQM en donde la organización, dependiendo de sus objetivos, obtendrá el modelo de calidad que requiere.

Figura 3. **Modelo de calidad goal question metric (GQM)**



Fuente: elaboración propia.

Cada objetivo se descompone en varias preguntas para entender los componentes del objetivo y finalmente se obtienen métricas que dan respuesta a cada una de las preguntas.

El modelo de calidad CMMI se fundamente en buena medida en las prácticas que establece el enfoque GQM. Por lo que los elementos de este modelo para calidad de software se pueden definir en tres segmentos:

- Atributos de calidad en operación: rendimiento, confiabilidad, tolerancia a fallas, seguridad, uso.

- Atributos de calidad en desarrollo: eficiencia, mantenimiento, reutilización, verificable.

- Atributos de calidad en implementación: disponibilidad, flexibilidad, interoperabilidad, instalable, portable, recuperable, escalable y seguro.

2.2.3. Modelo de calidad funcionality, usability, reliability, performance, supportability (FURPS)

El modelo FURPS establece únicamente, contrario a los modelos anteriores, cinco características como factores de calidad. Mismas que le dan su nombre:

- Funcionalidad (*Functionality*)
- Usabilidad (*Usability*)
- Confiabilidad (*Realibility*)
- Rendimiento (*Performance*)
- Soporte (*Supportability*)

Estas cinco características son divididas en dos tipos de requerimientos que debe cumplir un sistema:

- Requerimientos funcionales (F): funciones del sistema sin restricciones físicas.

- Requerimientos no funcionales (URPS): atributos del sistema o el ambiente del sistema.

Este modelo está limitado, pues no toma en cuenta la portabilidad de los productos, factor digno de considerar en las exigencias actuales.

2.2.4. Modelo de calidad web quality evaluation method (WebQEM)

Es el modelo utilizado para evaluación y comparación de calidad de una web. Radica en comprender el cumplimiento de un grupo de características respecto a los requerimientos de calidad solicitados. La ventaja de este modelo es que evalúa distintos dominios de aplicación y puntos de vista que se definen al inicio como requerimientos de calidad.

El modelo WebQEM plantea 4 características de calidad distribuidas en subcaracterísticas y atributos.

- Facilidad de uso: comprensibilidad del sitio, mecanismos de ayuda, aspectos de interfaces y estéticos, y misceláneas.

- Funcionalidad: aspectos de búsqueda y recuperación; aspectos de navegación y exploración; aspectos del dominio orientados al estudiante.

- Confiabilidad: no deficiencia.

- Eficiencia: performance y accesibilidad.

2.3. El campo de la reutilización

En el campo de la ingeniería la mayor parte del diseño de sistemas se basa en la reutilización de componentes o módulos existentes. Al referirse a la tecnología de la información e ingeniería de software, un elemento reutilizable define un producto diseñado para ser empleado de manera recurrente en el desarrollo de muchas aplicaciones y sistemas.

Se habla de componentes de software como algoritmos, arquitecturas, diseños de prueba y desarrollo; estructuras de bases de datos, entre otros. La ingeniería del software basada en reutilización es una estrategia en la que el proceso de desarrollo se adapta a la utilización de software ya existente y ha surgido durante los últimos años como resultado de tendencias crecientes en la industria orientadas a reducir costo, riesgo y tiempo de desarrollo de aplicaciones; manteniendo la calidad final del producto y la satisfacción del cliente.

2.3.1. ¿Qué es la reutilización?

La ingeniería del software basada en reutilización, intenta maximizar la reutilización del software existente, desglosando tres clasificaciones para los elementos ya establecidos y las ventajas por las cuales surge esta implementación:

- Componentes ya desarrollados: elementos que han sido aplicados a proyectos anteriores. Estos ya fueron validados y están listos para utilizarse en un proyecto nuevo.

- Componentes ya experimentados: necesarios para la reducción de riesgos en las modificaciones de nuevos sistemas de software, este conjunto incluye datos de prueba, código, especificaciones técnicas, o diseños existentes ya desarrollados para proyectos anteriores que resultan ser similares a una nueva implementación de software.

- Componentes con experiencia parcial: este caso particular incluye componentes que requieren modificaciones sustanciales para la aplicación en el nuevo proyecto, incluyen datos de prueba, código,

especificaciones técnicas, o diseños existentes ya existentes en proyectos anteriores y que se relacionan con el software requerido para el proyecto actual.

Aunque de la reutilización se pueden obtener muchas ventajas una de las características que posee el software es la capacidad de ser específica a una necesidad y ajustarse a requerimientos de otros sistemas. Esta nueva solución puede resultar con costos elevados. Así que, no solo se considera el campo de la reutilización como el uso de componentes de software existentes sino también existe la forma complementaria de reutilización denominada reutilización de conceptos donde no se adapta un en su totalidad como software, sino se abstrae el concepto que posee y se rediseña para ser adaptado a otra variedad de sistemas.

Al utilizar esta técnica se requiere de una actividad de instanciación en donde se adapta la funcionalidad de los conceptos abstractos a un sistema concreto que funciona de manera similar, dando origen a las técnicas de: patrones de diseño y generación de programas.

2.4. Métodos de reutilización

Los sistemas pertenecientes a un mismo dominio de aplicación son similares en comportamiento, estructura y configuración, por lo tanto, tienen potencial para la reutilización. Se han desarrollado muchas técnicas para soportar la reutilización desde hace varios años, la pregunta clave resulta ser ¿cuál es la técnica ideal?, dado que cada una se basa en estándares que ofrecen beneficios diferentes o son aplicables a sistemas específicos. Las técnicas existentes aplican reutilización de componentes desde funciones simples hasta aplicaciones complejas.

Seleccionar la técnica correcta depende de los requerimientos del sistema nuevo requerido, las funciones existentes con las que se cuenta, la tecnología que se aplicará y la experiencia del equipo de desarrollo.

El éxito o fracaso de la aplicación de la reutilización consiste en cuestión más de gestión que de habilidad técnica.

En la mayoría de situaciones existe la posibilidad de aplicar reutilización dada la cantidad de sistemas existentes.

Hay dos caminos posibles en el desarrollo de un sistema de software:

- Los representantes pueden ser renuentes a proteger sus requerimientos para utilizar el uso de componentes ya implementados debido a los riesgos que este conlleva, se aplica el dicho de preferir los riesgos conocidos a los riesgos por conocer.

- El otro camino es que se opte por un desarrollo completamente nuevo, pero este desarrollo de componentes nuevos generará una base de activos software para un sistema futuro.

Es posible que no se conozcan los riesgos existentes a la hora de reutilizar componentes.

A continuación se presenta una lista de aspectos necesarios para considerar la aplicación de reutilizar para generar.

- Cronograma de desarrollo de software: se debe considerar el tiempo necesario para obtener resultados finales; en un sistema de software que

no cuenta con mucho tiempo para la etapa de desarrollo debe firmemente considerar la implementación de componentes reutilizables en lugar de componentes individuales. Se podría generar la desventaja de que los requerimientos no sean perfectos, pero se minimizará el tiempo de desarrollo requerido.

- Tiempo de vida esperado del software: la aplicación de sistemas reutilizables generan una mantenibilidad efectiva para los sistemas. Si se está pensando en construir un software de larga vida, es esencial considerar la reutilización dadas las ventajas que ofrece y las implicaciones que genera a largo plazo. El punto débil seria utilizar únicamente componentes propios y no considerar software de proveedores externos que no sean capaces de proveer mantenimiento a los componentes o brindar el soporte necesario. Para el caso de sistemas sencillos que no sean considerados de larga vida o no requieran alta disponibilidad la opción esta siempre disponible, pero quizá no sea la más económica en costo y riesgo.

- Habilidades del equipo de desarrollo: se requiere de cierta habilidad técnica para desarrollar la reutilización, el cumplimiento de estándares y el proceso de implementación debe ser realizado por personal con experiencia, si el equipo de desarrollo cuenta con esta la reutilización será una técnica efectiva.

- Requerimientos no funcionales y criticidad del software: para este caso existirá un problema si el sistema posee requerimientos no funcionales estrictos que tienen que ser certificados por un regulador externo y que deben tener acceso total al código fuente; los componentes reutilizados que no lo proporcionen no podrán ser aplicados al nuevo sistema y la

técnica de reutilización a través de generadores de programas quedará obsoleta, sin embargo, existen otras más que se pueden aplicar, como se mencionó anteriormente, una de ellas sería la reutilización de conceptos.

2.4.1. Reutilización basada en patrones de diseño

El objetivo de la reutilización de componentes ejecutables ya desarrollados, consiste en adaptarlos a partir de ciertas entradas para obtener salidas específicas. Esto genera un problema cuando los requerimientos de un nuevo sistema de software no se adaptan al funcionamiento que posee un componente.

Los componentes ejecutables podrían no adaptarse a los requerimientos solicitados ya sea desde los algoritmos utilizados para su construcción hasta la abstracción de objetos e interfaces utilizados en la implementación y presentación.

Para solventar esta situación se utiliza la reutilización de diseños abstractos, que dan origen al concepto de patrones de diseño.

El patrón contiene la descripción de un problema y la esencia de su solución, de modo que esta solución puede ser utilizada para aplicarse a diversas situaciones. El patrón no contiene una solución detallada específica sino representa una descripción del conocimiento y experiencia de otras situaciones similares.

El concepto de patrón de diseño se derivó de los documentos presentados por Christopher Alexander en 1987, quien publicó la existencia de ciertas

similitudes en el diseño de edificios con características comunes y que la adopción del mismo patrón para el diseño de estos era confiable y efectiva.

Un patrón se define como la solución efectiva y adecuada a un problema común, orientado a la reutilización, el sitio web hillside.net define el concepto de patrón como:

Los patrones y los lenguajes de patrones son formas de describir las mejores prácticas, buenos diseños y encapsulan la experiencia de tal forma que es posible para los otros realizar dicha experiencia.[1]

Los patrones de diseño son aplicables a cualquier aproximación de diseño de software aunque, generalmente se caiga en el error de que únicamente se aplican a conceptos de desarrollo orientado a objetos, debido a que los patrones a menudo consideran características como polimorfismo y herencia para describir generalidades de los componentes.

Se han definido 4 elementos esenciales para el desarrollo de un patrón de diseño.

- Nombre: referencia significativa y clara del patrón.

- Descripción del área del problema: explica en qué momento puede aplicarse el patrón.

- Descripción de las partes de la solución: muestra gráficamente las relaciones entre objetos del patrón y las clases de los objetos en la

[1] Reutilización del software. http://www.slideshare.net/pto0404/seminario-3-reutilizacin-del-software. Consulta septiembre de 2013.

solución, en resumen, es una breve descripción de sus relaciones y propiedades.

- Declaración de consecuencias: muestra una lista de resultados obtenidos en la aplicación de un patrón que ayuda a los diseñadores a tomar la decisión en la utilización de un patrón.

Esta implementación de reutilización resulta ser muy efectiva con la desventaja de que solo puede ser utilizada por desarrolladores de software con amplia experiencia en el campo con la capacidad para reconocer situaciones genéricas en las que son aplicables los patrones, habilidad que será difícil de aplicar para un desarrollador que no tenga experiencia en el análisis y reutilización de un patrón.

Actualmente existe una gran cantidad de patrones publicados en internet que abarcan varios dominios de aplicaciones y lenguajes. Por lo que su aplicación cada vez se hace más común en diseño de interfaces de usuario y escenarios de interacción.

2.4.2. Reutilización basada en generadores

La reutilización basada en patrones de diseño abstrae el análisis y funcionamiento de un patrón, para que los desarrolladores implementen desde cero una solución efectiva. Con la desventaja de tiempo de desarrollo aun extensos se pensó en la implementación basada en generadores donde al mismo tiempo que se abstraen las funciones lógicas y análisis de un patrón, todo este conocimiento reutilizable se captura en aplicaciones generadoras de programas (realizados por expertos en el domino utilizando una herramienta

CASE o un lenguaje orientado a dominios), para generar un sistema de software operativo.

El generador recibirá la descripción de la aplicación específica y definirá que componentes reutilizables tienen que usarse y la forma en que deben ser parametrizados y combinados.

La ventaja de los generadores radica en que aprovecha la definición de que productos existentes a dominios iguales tienen arquitecturas comunes y realizan funciones comparables. Se presenta el siguiente caso:

Las aplicaciones de procesamiento de datos siguen un dominio similar en el que las operaciones se resumen en entrada, proceso y salida. El proceso consiste en verificación de datos y la salida en generación de informes.

En este caso ideal, pueden crearse componentes genéricos que sean aplicados por una aplicación generadora y forjen soluciones finales de software.

La aplicación generadora tendrá la capacidad de que el desarrollador, únicamente tenga que definir los datos a utilizar, las comprobaciones que debe realizarse y el formato de los informes que requiere para obtener el sistema de software requerido.

Existen muchos productos generadores de aplicaciones en el mercado con la capacidad para generar sistemas de software completos o automatizar ciertas tareas de un sistema, para que el programador las complemente con características específicas.

La mayoría de estos productos están orientados a aplicaciones de negocios aunque existen excepciones tales como:

- Generadores de lenguajes: son generadores que a partir de reglas básicas léxicas, sintácticas y semánticas producen un analizador léxico/sintáctico para en lenguaje descrito, ejemplo de estos generadores son JLex para el lenguaje Java y Antlr para diversos lenguajes como Java, C++ o Phyton.

- Generadores de código en herramienta CASE: este tipo de generadores se han vuelto muy populares por ser capaces de generar el esqueleto de un sistema a partir de su análisis, existen varias herramientas que generan interfaces, clases y relaciones a partir de un modelo UML que recibe de entrada, todo ideado para que el desarrollador únicamente agregue funcionamiento a la estructura final obtenida.

Puede resultar imposible utilizar esta técnica para sistemas que tengan requerimientos de elevado rendimiento, dado que la estructura producida por una aplicación generadora se apega a un dominio de aplicaciones, esta es más rentable de aplicar a soluciones tales como: procesamiento de datos de negocios.

La técnica basada en generadores produce un método de desarrollo denominado programación generativa, que combina técnicas emergentes de desarrollo y generación de programas basados en componentes, convirtiéndose en un elemento clave de la reutilización.

2.4.3. Reutilización basada en marcos de trabajo

Un marco de trabajo (*framework* desde ahora), consiste en un conjunto de clases concretas y abstractas que son utilizadas para generar un nuevo subsistema y la interfaz de comunicación entre desarrollador y *framework*. Al construir un sistema de software, generalmente se hace uso de implementaciones concretas de las clases contenidas en un *framework*. Por sí mismo un *framework* no es considerado como una aplicación sino es considerado parte fundamental de un sistema completo.

Los *framework's* se pueden clasificar en tres tipos distintos:

- De infraestructura de sistemas: proporcionan la base para el desarrollo de sistemas complejos, incluyendo entre sus características interfaces (formularios, formas, ventanas), protocolos de comunicación, compiladores, clases madre, entre otros.

- Para integración de *middleware*: se define como un conjunto de objetos reutilizables que cumplen con ciertas características y estándares para su consumo, los componentes ya están desarrollados y son aplicables a cualquier sistema cubriendo un domino mayor por el resultado de sus operaciones y tamaño de aplicación, ejemplos de ellos seria los denominados Enterprise Java Beans, o la aplicación de librerías *dll* utilizando COM+ de Microsoft.

- De aplicaciones empresariales: estos encapsulan conocimiento de un dominio específico, contienen las características del *framework* de infraestructura aplicado a un área específica.

La estructura de un *framework* puede ser extendida para generar nuevos sistemas y se implementa por medio de clases que heredan características de las clases abstractas que proporciona el *framework* o mediante el uso directo de las clases más concretas.

Los *frameworks* son a menudo instanciaciones de varios patrones de diseño, es decir, surgen a partir de un concepto de reutilización y la mayor ventaja que presenta para construir sistemas basados en esta técnica es que estas mismas aplicaciones se convierten en una base para su posterior reutilización a través del concepto de líneas de producto o familias de aplicaciones. Basado en la característica de que un sistema fue construido con un *framework*, construir nuevas versiones o familias de esta aplicación se simplificará.

Entre las desventajas de la utilización de *frameworks* está la complejidad que requiere aprender a utilizarlos correctamente, no hay duda de que esta es una aproximación efectiva para la reutilización, pero es muy elevado el coste que supone introducirla en los procesos de desarrollo de software.

2.5. Ventajas de la reutilización

Se presentan los beneficios que ofrece la aplicación del concepto de la reutilización.

- Incremento de la confiabilidad: el software reutilizado que ha sido usado y probado en sistemas en funcionamiento, resulta más confiable que el software utilizado en una implementación nueva, los fallos del software reutilizado ya fueron encontrados, analizados nuevamente y corregidos.

- Reducción del riesgo del proceso: se aplica al dicho de que, es mejor conocer riesgos a enfrentarse a nuevos, cuando se realiza la proyección de riesgos para un nuevo sistema, si el sistema va a utilizar componentes ya existentes, estos reducen el margen de error. Es aún más visible cuando se utilizan componentes relativamente grandes para conformar otro sistema.

- Uso efectivo de especialistas: al adoptar el concepto de reutilización los desarrolladores especializados pueden enfocarse en desarrollar software reutilizable que encapsule sus conocimientos.

- Cumplimiento de estándares: genera una ventaja en la presentación de sistemas a los usuarios finales, por ejemplo, cuando se trabaja con un menú que haya surgido de un componente reutilizable, todas las pantallas de la aplicación contarán con un menú similar, lo que facilitará el ambiente a los usuarios y reducirá la probabilidad de errores en el manejo y navegación.

- Desarrollo acelerado: la ventaja más visible es el tiempo requerido para desarrollo de la aplicación, las organizaciones buscan sacar ventaja siempre sobre los competidores o satisfacer lo más rápido posible a sus usuarios. Con el uso de componentes reutilizables se acelera la producción del sistema dado que se evita tiempo de desarrollo y validación.

2.6. Desventajas de la reutilización

Existen también costes y problemas asociados a la reutilización, considerando como primer coste al esfuerzo realizado para determinar si un

componente es apropiado para reutilizarlo en la situación correcta y que demuestre que ese componente asegurará su confiabilidad. Entre otros problemas se encuentran:

- Incremento en costes de mantenimiento: este se genera si el código fuente de un componente no está disponible, considerando un costo alto debido a que este componente se comenzará a tratar como incompatible con las actualizaciones del sistema.

- Falta de soporte de las herramientas: existen herramientas que no soportan reutilización, ejemplo de ellas, las herramientas CASE con las cuales se hace difícil o imposible integrarlas con sistemas de librerías de componentes.

- Síndrome de reinventar la rueda: algunos desarrolladores experimentados pueden preferir desarrollar un nuevo componente desde cero a utilizar los elementos que ya se tienen, esto debido a que desarrollar nuevo código genera más reto que utilizar el software de otra persona.

- Creación y mantenimiento de una librería de componentes: resulta caro construir una librería completa de componentes reutilizables y asegurar que estos pueden ser utilizados por los desarrolladores sin necesidad de realizar cambios o adaptaciones fuertes.

- Búsqueda, comprensión y adaptación de componentes reutilizables: este problema radica en la búsqueda de un componente y la adaptación de este dentro de una biblioteca de componentes reutilizables. Las bibliotecas deben contener una cantidad de recursos que generen al

desarrollador la confianza de que va a encontrar un componente que se adapte a sus necesidades antes de que comience el desarrollo de uno nuevo.

2.7. Atributos que definen la calidad del software

La complejidad en la evaluación y definición de calidad de un software es producto de la gran cantidad de características y atributos que pueden intervenir en los requerimientos de calidad y en las varias relaciones existentes entre los atributos, subcaracterísticas y características, entre otros aspectos.

Debido a esto se presenta el siguiente modelo estándar para la definición de atributos que determinan la calidad de un software.

2.7.1. Modelo ISO/IEC 9126-1:2001

El modelo especifica seis características de calidad interna y externa. La calidad externa evalúa cómo las características satisfacen la necesidad de un usuario, medible en el comportamiento del producto. La calidad interna realiza evaluación total de los atributos que un software debe poseer tomando en cuenta condiciones específicas.

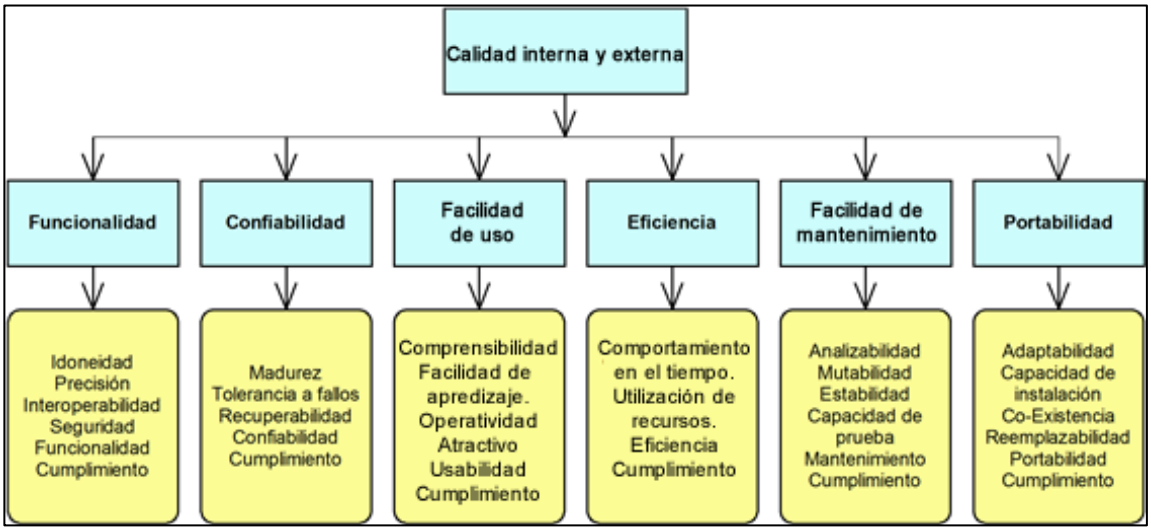
El modelo establece tres niveles:

- Características: que serán las que se compararán al final contra la aplicación de componentes.

- Subcaracteristicas: que describen el comportamiento y medición de las características.

- Métricas: que se utilizarán para la comparación y enlace entre características de calidad de componentes en el último capítulo.

Figura 4. **Modelo de calidad ISO/IEC 9126-1:2001**



Fuente: elaboración propia.

- Funcionalidad: representa la cantidad de posibilidades provistas por un sistema. Un sistema de calidad está basado en un conjunto de funcionalidades que encapsulan las ideas más brillantes sin perder la consistencia. Es necesario que en la carrera por agregar funciones se pierda todo indicio de calidad, por lo que la funcionalidad debe estar reservada para operaciones necesarias.

- Confiabilidad: característica que determina la probabilidad de que un software realice su objetivo satisfactoriamente en un determinado escenario. Considerada característica fundamental, ya que si un programa falla frecuentemente en su funcionamiento, no importa si el

resto de los factores de calidad son aceptables. Posee la ventaja de medición, pues se puede obtener este factor mediante los datos históricos o de desarrollo.

- Facilidad de uso: esta característica describe la simplicidad que posee un software para ser utilizado por el usuario para aplicarlo en resolver problemas, también incluye la facilidad de instalación, operación y monitoreo. Esta característica está más enfocada al usuario final, quien va a interactuar directamente con el sistema desarrollado. La facilidad de uso incluye tareas como: aprendizaje de la utilización del sistema; disminución de la probabilidad de error; efectividad y eficiencia en la realización de dichas tareas y la satisfacción final del usuario.

- Eficiencia: característica enfocada en la evaluación de recursos utilizados por un sistema. La eficiencia es la habilidad que posee el software realizará una función con una utilización equilibrada de recursos, los cuales pueden ser: tiempo de CPU; memoria principal; memoria secundaria; canales de entrada/salida; velocidad de ejecución y tiempo de respuesta. Esta característica está ligada a las propiedades del hardware, por lo que debe estar ligada a aspectos como extensibilidad y reutilización.

- Facilidad de mantenimiento: esta característica está ligada al desarrollador y es parte de la calidad del software debido a que el mantenimiento de sistemas representa más esfuerzo que cualquier otra actividad en la ingeniería del software. Este atributo de calidad define el esfuerzo necesario para corregir un programa si se encuentra un error o adaptarlo si cambia el entorno en el cual se estaba ejecutando o mejorarlo si el cliente presenta un cambio en los requisitos originales. La

facilidad de mantenimiento se resume en la facilidad de comprender, corregir, adaptar y mejorar el software.

- Portabilidad: aunque es la característica de calidad menos relevante, resulta importante en sistemas grandes y complejos. Este factor define la facilidad para transportar productos de software a varios ambientes de hardware y software. Este atributo, también está ligado a la facilidad de instalación y reemplazo de un sistema.

2.8. ¿La reutilización de componentes asegura la calidad?

La calidad del proceso contribuye a mejorar la calidad del producto, y la calidad del producto contribuye a mejorar la calidad en uso. Ahora que ya se sabe cuáles son los atributos necesarios para que un sistema software sea de calidad, se definirán los atributos requeridos por un componente para asegurar su calidad con el objetivo de encontrar cómo repercuten las características individuales en un sistema completo.

Hasta el momento sería incorrecto afirmar que incluir componentes con características de calidad a un software aumentaría su calidad, por lo que en los siguientes capítulos se determinará el impacto de esta implementación.

3. INGENIERÍA DE SOFTWARE BASADA EN COMPONENTES

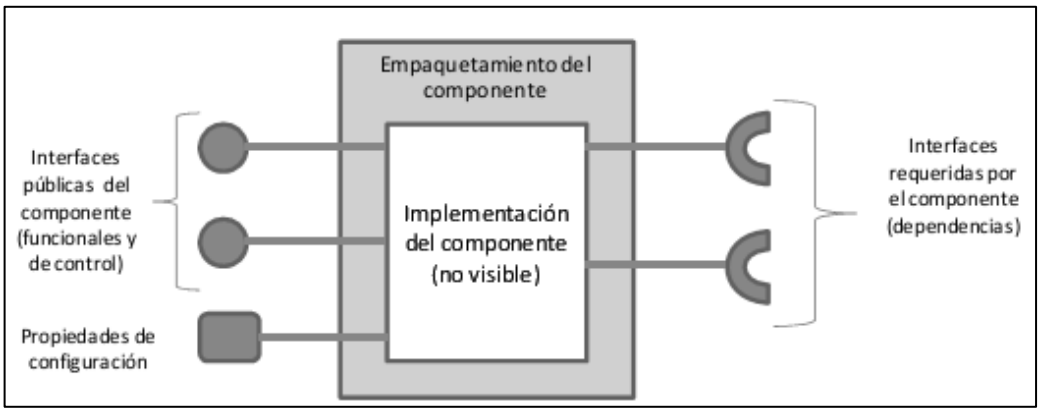
La ISBC es similar a la ingeniería de software basada en objetos, se comienza con un proceso de toma de requerimientos utilizando cualquier técnica convencional, después se procede a efectuar el diseño arquitectónico y antes de comenzar directamente con tareas de diseño detalladas se efectúa una evaluación de requisitos para determinar qué subsistema será capaz de componer el lugar de construir. Esta técnica se ha convertido en una parte importante del desarrollo de aplicaciones robustas debido a que cada vez poseen requerimientos funcionales más complejos y los clientes demandan sistemas más confiables en un tiempo de desarrollo menor.

3.1. Componentes de software

Un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas.

Un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros.

Figura 5. **Representación esquemática de un componente software**



Fuente: elaboración propia.

Los fundamentos de la ingeniería del software basada en componentes son:

- Componentes independientes: son completamente independientes de la interfaz, lo que hace que la sustitución de un componente no afecte el funcionamiento del sistema.

- Estándares de componentes: definen a un bajo nivel cómo las interfaces de cada componente generan comunicación entre ellos. Por definición todo componente debe tener la capacidad de integrarse a un grupo sin importar su lenguaje de desarrollo.

La conformación de estos fundamentos indica que los componentes de software también poseen la propiedad de ser reutilizables, debido a que un sistema basado en componentes podrá, desde ser ensamblado con facilidad hasta presentar cambios de funcionamiento e integración con el mínimo esfuerzo necesario y hacerlo sin alterar la vista del usuario. Para lograr una

eficaz integración y comunicación entre componentes, se necesita de un marco de trabajo sobre la cual se estandaricen operaciones y existan restricciones sobre el diseño.

3.2. Atributos de los componentes

Aun con las definiciones expuestas no se tiene una clara idea de qué es lo que debe integrar a un componente, para garantizar la reutilización de un componente, este debe cumplir con las siguientes características:

- Estandarizado: para que un componente posea esta característica debe pertenecer a algún modelo estandarizado de componentes, donde estén definidas interfaces, metadatos, composición, comunicación y despliegue.

- Independiente: un componente no requiere de otro para su implementación, este debe de poder integrarse o desprenderse de un sistema sin la intervención de otro. Si el componente necesitara de algún otro para su implementación debe ser especificado como un caso de requieres en su documentación.

- Componible: el componente debe proporcionar acceso de su información (atributos y métodos) a entidades externas por medio de interfaces, las interacciones externas, también deben ser reguladas por una interfaz definida públicamente.

- Desplegable: el componente debe de estas listo para usarse sobre cualquier plataforma que implemente el mismo modelo de componentes.

Debe ser un elemento compilado del que solo se hagan uso de sus interfaces sin necesidad de compilarlo.

- Documentado: el componente debe poseer toda la documentación necesaria para que el usuario que desee implementarlo pueda tomar la decisión correcta al usarlo, debe poseer información sobre el desarrollo, interfaces disponibles, sintaxis y semántica que maneja.

Se definen atributos extras que son más específicas según la implementación de los componentes. Para el caso de un componente como proveedor de servicios se presentan:

- Componente como entidad ejecutable: la funcionalidad del componente ya existe para cualquier momento en que desee utilizarse, el código fuente no está disponible, únicamente la versión compilada.

- Componente como proveedor: deben de suministrarse funciones a través de interfaces que no revelen el estado interno de un componente, todo esto debe funcionar en términos de operaciones parametrizadas.

Por la definición de su interfaz se presentan:

- Interfaz proporciona: este tipo de interfaz define todas las operaciones básicas de un componente, define los accesos y métodos que pueden ser invocados por un usuario, en pocas palabras, es el API del componente.

- Interfaz requiere: establece la comunicación entre un componente y otro, define los métodos que deben ser suministrados por componentes

externos, tratados como entradas. Esta propiedad no se considera dentro del contexto de independencia o despliegue.

Por el desarrollo de sus interfaces se presentan:

- Entidad desplegable: un componente será accedido desde una plataforma de ejecución y puede ser manipulado tanto por usuario como por otros componentes, en ningún momento pertenecerá a una plataforma de aplicación donde se pueda acceder a su código fuente y compilarlo a deseo.

- No representación de tipo: un componente es la instancia, no es la plantilla para representar una instancia como en el concepto de clases y objetos.

- Implementación opaca: todas las operaciones internas y atributos específicos de un componente no deben ser accesibles para el usuario que compra o utiliza el componente. Las interfaces proporcionan operaciones, parte de esas operaciones puede retornar datos específicos, pero son parte de la misma implementación.

- Independencia del lenguaje: la interacción entre componentes, generalmente se da dentro de un marco de desarrollo similar, por lo tanto los componentes que interactúan deben estar desarrollados bajo un mismo lenguaje, pero esto no quiere decir que los componentes deben ser desarrollados bajo un lenguaje específico, el lenguaje es independiente de la implementación, sea orientado a objetos, estructurado o cualquier otro.

- Estandarización: los componentes pertenecen a un modelo definido que estandariza su comportamiento.

3.3. Modelos de calidad de componentes

En los modelos de calidad existe un conjunto de características de calidad que se utilizan como base para la evaluación. Basado en este enfoque, el modelo de calidad contiene todos los factores posibles y se usará un subconjunto de estos factores para el desarrollo de un componente.

El punto central serán los modelos de evaluación de calidad externa debido a que para los componentes software disponibles en el mercado, no es factible la evaluación de su calidad interna pues se desconoce el proceso de desarrollo y tampoco la evaluación de calidad en uso, ya que esta depende del uso que requiera darse a cada uno de los componentes seleccionados.

3.3.1. Modelo ISO/IEC 9126-1

El modelo se define como una jerarquía multinivel de factores de calidad, el nivel más alto corresponde a las características de calidad externas (de componentes) y el siguiente nivel presenta atributos medibles cuyo valor se puede obtener aplicando una métrica.

Tabla III. **Modelo ISO/IEC 9126-1**

Funcionalidad	Adecuación, exactitud, interoperabilidad, seguridad, cumplimiento funcional.
Fiabilidad	Madurez, tolerancia a fallos, capacidad de recuperación.
Usabilidad	Capacidad para ser aprendido y administrado.

Continuación de la tabla III.

Eficiencia	Comportamiento temporal, utilización de recursos, cumplimiento de la eficiencia.
Mantenibilidad	Capacidad de análisis, capacidad de cambio, estabilidad.
Portabilidad	Adaptabilidad, instalabilidad, coexistencia, capacidad para reemplazar.

Fuente: elaboración propia

3.3.2. Atributos de calidad de los componentes software

Como se puede observar, el modelo descrito en el párrafo anterior es el mismo que describe la calidad de software, para definir los atributos de calidad de componentes es necesario hacer una separación entre las características que tienen sentido aislado (locales) y las que deben ser valoradas a nivel de arquitectura de software (globales).

Por ejemplo, el atributo tolerancia a fallos es propia de la arquitectura de una aplicación, por el contrario, el atributo madurez es más propio de un componente.

Otro punto importante a considerar es que las características que se van a exponer están dirigidas a los arquitectos de software que van a seleccionar los componentes que integrarán sus aplicaciones.

Como no todas las características descritas en el modelo ISO/IEC 9126-1 son aplicables a componentes, se propone el siguiente cuadro que define el modelo de calidad estrictamente para componentes.

Tabla IV. **Modelo de calidad para componentes**

Característica	**Subcaracterísticas (tiempo de ejecución)**	**Subcaracterísticas (ciclo de vida)**
Funcionalidad	Precisión Seguridad	Idoneidad Interoperabilidad Conformidad Compatibilidad
Fiabilidad	Recuperabilidad	Madurez
Facilidad de uso		Facilidad de aprendizaje Facilidad de comprensión Operatividad Complejidad
Eficiencia	Comportamiento temporal Utilización de recursos	
Mantenibilidad		Cambiabilidad Facilidad de prueba

Fuente: elaboración propia.

Este modelo define varias características que describen la calidad en un componente.

A continuación se presenta una división de los atributos según formen parte del ciclo de vida de un componente.

- Atributos medibles en tiempo de ejecución
    - Sobre la precisión de un componente:
        - Precisión: se encarga de medir la información devuelta por las operaciones de un componente en tiempo real. Medido

por el número de resultados correctos dividido entre el número de resultados imprecisos.

- Exactitud computacional: evalúa el número de resultados incorrectos respecto al número de resultados ejecutados.

- Sobre la seguridad de un componente:

  - Cifrado de datos: evalúa si un componente es capaz de manejar datos de forma cifrada y el método que utiliza para tal actividad.

  - Capacidad de control: evalúa el nivel de seguridad y control de acceso a los datos que maneja un componente.

  - Capacidad de auditar: indica si el componente tiene la capacidad de registrar operaciones en una bitácora de actividades.

- Sobre la recuperabilidad de un componente:

  - Secuencializable: indica si el contenido que manipula un componente es capaz de ser secuenciado y transmitido a otros (incluye la transferencia y recuperación por medio de interfaces).

  - Persistente: indica si un componente es capaz de almacenar un estado de forma persistente.

- Transaccional: indica si un componente suministra funciones o métodos que aseguren que sus operaciones se den por medio de una transacción.

- Recuperación de errores: tratamiento y recuperación de errores de los datos manejados dentro de los componentes, se divide en tres acciones básicas: detección, aviso y tratamiento de errores.

- Sobre el comportamiento temporal:

  - Tiempo de respuesta: evalúa el tiempo transcurrido desde que se recibe una petición hasta que ese cumple con el resultado solicitado.

  - Capacidad de emisión: mide la cantidad de información emitida por un atributo en un lapso del tiempo durante el cual es observado.

  - Capacidad de recepción: mide la cantidad de información que es capaz de procesar un componente en respuesta a una petición en un tiempo dado.

- Sobre la utilización de recursos:

  - Requisitos de memoria: cantidad de memoria que requiere un componente para ejecutarse. Medida por el consumo de recursos dividido la solicitud del componente.

        - Utilización de disco: mide el tamaño de disco que ocupa el componente, de forma estática, sumado al espacio requerido de forma dinámica durante la ejecución de operaciones.

- Atributos medibles durante el ciclo de vida

    - Sobre la idoneidad:

        - Cobertura: se mide respecto al número de interfaces que proporciona un componente comparado contra el número que necesita el usuario del componente.

        - Exceso: esta métrica está relacionada al número de interfaces que se utilizan realmente en la aplicación del componente.

    - Sobre la interoperabilidad:

        - Compatibilidad de los datos: este atributo describe si los datos de entrada o salida que son proporcionados a o por el componentes se apegan a un estándar o especificación que sea compatible con otros componentes o durante el flujo de datos dentro del sistema software.

    - Sobre la conformidad:

        - Conformidad con estándares: indica si el componente cumple con algún estándar internacional en funcionalidad.

- Certificaciones: indica si existe alguna certificación aplicada a este componente.

- Sobre la compatibilidad:

  - Compatibilidad hacia atrás: indica si existe compatibilidad de un componente actual con las versiones anteriores de este.

- Sobre la madurez:

  - Volatilidad: este atributo se encarga de medir el promedio de tiempo transcurrido entre cada versión comercializada de un componente.

  - Evolucionabilidad: atributo que se mide a través de la cantidad de versiones existentes de un componente, lo que da una noción de su madurez.

  - Fallos eliminados: indica el número de errores corregidos en cada versión, aunque el surgimiento de errores y correcciones depende de la implementación.

- Sobre la facilidad de aprendizaje:

  - Periodo para usar correctamente: medido en unidades de tiempo, describe el tiempo necesario para que un usuario nuevo pueda utilizar de forma correcta el componente.

- Periodo para configurar correctamente: mide el tiempo necesario para que un desarrollador pueda configurar de forma adecuada los parámetros requeridos por una función de un componente.

- Periodo para administrar correctamente: mide el tiempo medio para poder realizar las operaciones de administración que requiera el componente de forma correcta.

- Sobre la facilidad de comprensión:

  - Documentación de usuario: calidad de la documentación suministrada junto a un componente.

  - Sistema de ayuda: indica la calidad del sistema de ayuda asociada a un componente.

  - Formación: describe la existencia de un curso o capacitación sobre cómo utilizar el componente proporcionado.

  - Cobertura de la demostración: porcentaje de interfaces disponibles durante una demostración contra el número total de interfaces existentes proporcionadas por un componente.

- Sobre la operatividad:
  - Esfuerzo para operar: grado de esfuerzo requerido por un desarrollador para operar un componente.
  - Esfuerzo para configurar: el mismo caso que el esfuerzo para operar, pero asociado a la configuración.
  - Esfuerzo para administrar: grado de esfuerzo requerido para realizar operaciones administrativas sobre un componente.
- Sobre la complejidad:
  - Interfaces ofrecidas: mide la cantidad de interfaces funcionales que ofrece un componente, cuanto mayor sea este número mayor será la complejidad de uso, y posiblemente, también su complejidad funcional.
  - Interfaces externas utilizadas: cantidad de otros componentes que necesita el componente para operar de manera adecuada o para proporcionar funcionalidad al sistema software.
  - Índice de complejidad: este atributo mide el número medio de operaciones por interfaz ofrecidas por un componente.

- Sobre la cambiabilidad:

  - Modificabilidad: mide el número de parámetros que permiten modificaciones por parte del usuario del componente.

  - Índice de modificabilidad: relación que existe entre el número de parámetros que tiene un componente contra el número de interfaces ofrecidas, de esta manera se obtiene una medida de modificabilidad.

  - Capacidad de control del cambio: capacidad que adquiere el usuario para identificar la versión utilizada por cada componente.

- Sobre la facilidad de prueba:

  - Prueba automática de arranque: determina la capacidad de un componente para determinar su estado y el entorno necesario para ejecutar las funcionabilidad que le corresponde antes de ejecutarse.

  - Batería de pruebas: este atributo define la existencia de pruebas de caja negra, en donde se compruebe la funcionalidad correcta de un componente antes de ser implementado e integrado en un sistema software.

3.4. ¿Cómo evaluar la calidad de un componente?

La calidad, dentro de la ingeniería de sistemas, se define como la totalidad de aspectos y características de un producto o servicio que tiene que ver con su habilidad de satisfacer las necesidades declaradas o implícitas. La calidad de los componentes de software puede ser el factor decisivo para la selección de los componentes que entrarán a ser parte de un nuevo sistema, es decir, si dos componentes son candidatos para hacer parte de una aplicación y muestran la misma funcionalidad, el nivel de calidad que implemente cada uno de ellos será el elemento sobre el cual se base la decisión final. Algunos de los elementos que pueden dar al usuario una idea del nivel de calidad de un componente son:

- Funcionalidad
- Interface
- Usabilidad
- Pruebas
- Seguridad

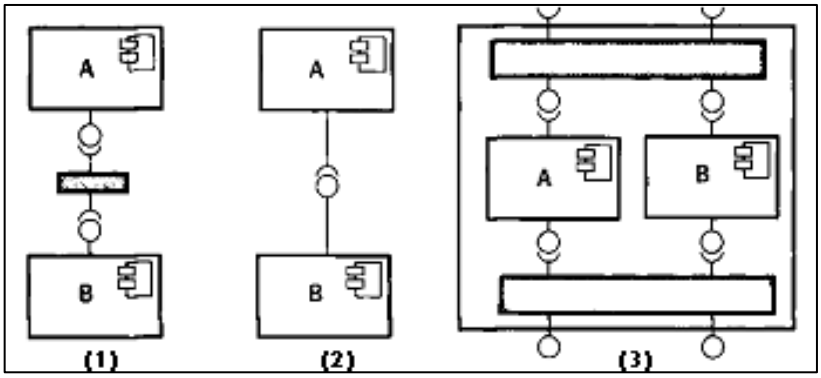
3.5. Composición de componentes

Bajo el moldeo de ingeniería del software basada en componentes, los nuevos sistemas se construyen mediante la integración o composición de componentes. Se define la composición como el proceso de construir aplicaciones mediante la interconexión de componentes de software a través de sus interfaces (de composición).

Haciendo énfasis especial en las interfaces como medio de interacción entre componentes.

La composición puede definirse como una relación cliente-servidor, entre componentes, adaptada a la infraestructura de soporte de los componentes proporcionada por el marco de trabajo del modelo. El componente cliente solicita un servicio del componente servidor el cual ejecuta la operación y envía una respuesta.

Figura 6. **Clasificación de la composición de un componente**



Fuente: SOMMERVILLE, Ian. Ingenieria de Software. p. 253.

- Composición secuencial: se da un par o grupos de componentes que se ejecutan en secuencia, es decir, las interfaces de cada componente al mismo tiempo que proveen funcionalidad, consumen funciones de un componente compañero generando una secuencia de operaciones.

- Composición jerárquica: es la situación más común de composición en donde la interfaz *requiere* de un componente solicita servicios a la interfaz *provides* de otro, generando una dependencia jerárquica.

- Composición aditiva: se da cuando las interfaces de dos o más componente se enlazan para crear un único componente, este proceso

elimina todas las operaciones que puedan estar duplicadas y añade funcionalidad y generalidad al componente nuevo.

Cuando los componentes se diseñan y desarrollan, especialmente para composición, se diseñan las interfaces de estos componentes para que sean compatibles de modo a que puedan ser tratados como una unidad. Cuando los componentes son desarrollados de manera individual se pueden generar problemas de compatibilidad en las interfaces cuando se desea hacer una composición, los problemas más comunes son:

- Incompatibilidad de parámetros: los tipos de datos de los parámetros, o la cantidad, son diferentes para cada interfaz de los componentes que se quieren integrar.

- Incompatibilidad de operaciones: los nombres de las interfaces entre parámetros son distintos.

- Operaciones incompletas: la interfaz que proporciona servicios del lado de un componente es un subconjunto de otra interfaz compuesta que generó un único componente anteriormente.

Para solucionar el problema de la compatibilidad, se escribe un nuevo componente adaptador que reúne las interfaces y garantiza el funcionamiento de la composición. La forma más eficaz de realizar este proceso es identificar la interfaz de un componente y desarrollar el nuevo componente como un adaptador que lo convierte en el otro componente.

La discusión sobre la incompatibilidad de los componentes supone que esta se podría evitar a partir de la documentación de cada interfaz de cada

componente, documentación incluye información sobre operaciones, nombres y tipos de parámetros de cada interfaz, por lo que desde una etapa temprana de diseño se puede definir si existirá la posibilidad de composición, sin embargo, lo que no garantiza la documentación de las interfaces es que si los componentes son semánticamente compatibles unos con otros.

La decisión de crear un sistema con base en la composición de componentes, requiere una ardua evaluación previa, ya que existen conflictos potenciales entre los requerimientos funcionales y no funcionales y la necesidad de entregar un sistema en un periodo de tiempo mas más corto que tenga la capacidad de evolucionar a medida que aparezcan nuevos requerimientos. Se debe de llegar a un equilibrio entre las siguientes decisiones:

- ¿Qué composición de componentes generará la solución óptima para satisfacer los requerimientos?

- ¿Qué composición garantizará que los cambios futuros sean satisfechos de una manera eficiente?

- ¿Cuáles serán las características positivas que tendrá el sistema compuesto? (rendimiento y confiabilidad)

El mejor principio para construir sistemas mediante una composición es dar a cada componente un papel claramente definido y no intercalar funcionamiento con otros componentes más que el que intercambia por medio de su interfaz. Además, se debe usar la mínima cantidad de componentes, ya que a menor escala menor complejidad para el sistema.

4. CALIDAD EN LA INGENIERÍA DEL SOFTWARE BASADA EN COMPONENTES

Hasta ahora se han analizado los atributos de calidad que debe poseer un sistema de software y las características asociadas directamente a los componentes; a continuación se definen los atributos ideales que debe poseer un sistema de software basado en componentes.

4.1. Ingeniería del software basada en componentes

Definir los atributos asociados a calidad, en este tipo de aplicaciones, resulta muy difícil debido a los siguientes aspectos:

- No existe una definición exacta de los atributos de calidad que se deben medir.

- Aun así, teniendo un modelo medible sobre los atributos de calidad que deben poseer los componentes, como es el caso de este documento, la mayoría de componentes no presentan documentación sobre el tipo de atributos que suministra el producto.

Se definirán a continuación características del desarrollo basado en componentes y una propuesta de características de calidad que debe poseer dicho proceso.

El desarrollo basado en componentes se ha considerado como un avance significativo hacia la construcción de sistemas mediante el ensamblado de

componentes prefabricados y trata de sentar las bases para el diseño y desarrollo de aplicaciones distribuidas con base en componentes reutilizables.

Puede considerarse al desarrollo basado en componentes como una extensión natural de la programación orientada a objetos dentro del ámbito de los sistemas distribuidos, tomando como base un componente que es una unidad de composición que ha de poder ser desarrollado e incorporado a otro conjunto de componentes de forma independiente en tiempo y espacio.

Una de las ventajas más significativas en el proceso de desarrollo basado en componentes se basa en la reutilización de estos. Los componentes son diseñados y desarrollados con la visión de ser reutilizados en otras aplicaciones reduciendo así el tiempo de desarrollo en proyectos nuevos y mejora de la fiabilidad (pues se utilizan componentes probados previamente). Aunque, este tipo de ventaja solo es aplicable a nivel interno dentro de las organizaciones que hacen uso de componentes comerciales (COTS), que son componentes con las siguientes características:

- Son vendidos y licenciados a público en general.

- Los mantiene y actualiza el distribuidor original quien conserva los derechos de propiedad intelectual.

- El código no puede ser afectado y se presenta como una caja negra.

El adoptar este tipo de desarrollo conlleva a ciertas modificaciones en los procesos tradicionales de desarrollo, donde enfoque como los llamados *top-down* o *bottom-up* están dando paso a procesos en espiral donde se hace

necesaria la retroalimentación continua entre requisitos del cliente, diseño arquitectónico de aplicaciones y componentes disponibles.

El desarrollo de software basado en componentes se divide en varias tareas para la construcción de aplicaciones:

- Búsqueda de componentes candidatos que satisfagan requisitos del cliente y la arquitectura.

- Evaluación y selección de candidatos.

- Adaptación de componentes a la arquitectura y requisitos.

- Integración, configuración y conexión entre conjuntos de componentes.

Aunque no todo el proceso de desarrollo basado en componentes se presenta como una ventaja, los beneficios aplicados a reducción de esfuerzo de desarrollo, costo de mantenimiento y mejora de producto final son suficientes para extenderse a varios entornos industriales. Debido a esto han surgido diversas metodologías para este tipo de desarrollo, aun así, está lejos de considerarse una verdadera ingeniería debido a que no dispone de métricas adecuadas, procesos, precios, ni una normativa internacional que la respalde.

Por lo anterior descrito, la definición de atributos de calidad asociados a componentes y a la implementación en un sistema de software aparece como una necesidad imperiosa.

4.2. Aspectos de calidad en la ingeniería del software basada en componentes

La calidad de un producto software no es su objetivo funcional, sino la satisfacción de un cliente. Por lo que la calidad de un sistema de software no se refiere únicamente a obtener un producto sin errores sino debe ser definido mediante un modelo estándar que detalle el conjunto de características necesarias para ser considerado de calidad.

4.3. Propuestas de calidad en la ingeniería del software basada en componentes

Basado en el punto de vista de la arquitectura de software y la relación con los componentes que la construyen se propone una clasificación de atributos de calidad en función de si un componente tiene relación con la arquitectura software o no, entonces se podra analizar a partir de estos atributos si un componente existe como entidad independiente o está relacionado con la arquitectura software.

Tabla V. **Modelo de calidad basado en la arquitectura**

Etapa del sistema	Atributo
Arquitectura en ejecución	Rendimiento Usabilidad
Durante el ciclo de vida	Mantenibilidad Portabilidad
Entorno comercial de la arquitectura	Costo Tiempo de vida previsto del sistema
Arquitectura software directamente	Integridad conceptual Construbilidad

Fuente: elaboración propia.

El autor Jan Bosch propone también, un modelo de atributos de calidad basado en la arquitectura, aunque describe la dificultad de especificar con detalle los requisitos de calidad, encuentra que los requisitos más importantes en la mayoría de propuestas se presentan de alguna forma en algo que denomina perfil.

Define un perfil como un conjunto de escenarios con alguna relativa importancia, entre ellos describe los siguientes:

- Perfil de uso: escenario que describe la utilización común de un sistema software.

- Perfil de cambios: es el escenario que se plantea durante el mantenimiento y desarrollo de un sistema.

- Perfil de riesgos: que representa la integridad, seguridad y aspectos propios de operación del sistema software.

Partiendo de la idea de la separación por perfiles, se propone el siguiente modelo de calidad, para el DSBC, el cual se describe en la tabla VI.

Tabla VI. **Modelo de calidad de Jan Bosch**

Rendimiento
Mantenibilidad
Fiabilidad
Seguridad física
Seguridad de acceso

Fuente: elaboración propia.

Este rubro describe la calidad que requiere un sistema de software basado en componentes durante el proceso de construcción, se propone el modelo COCOTS que presenta la división de actividades a medir para obtener los valores óptimos y medir la calidad.

Constructive COTS, este modelo propone cuatro submodelos que describen las cuatro principales fuentes de costo durante el proceso de integración de componentes:

- Valoración: proceso de selección e integración de componentes.

- Adaptación: actividades previas a la integración de un componente. Ejemplo: inicializar parámetros y configuración de protocolos.

- Código de integración: se refiere a la cantidad de código nuevo necesario para integrar componentes en un sistema.

- Volatilidad: frecuencia con la que cambian las versiones de los componentes utilizados dentro del sistema.

4.4. Reutilización de componentes

La reutilización de componentes es un proceso en el cual la construcción de una nueva aplicación involucra la reutilización de un conjunto de componentes ya desarrollados. De esta manera, se logra maximizar la reutilización de componentes de software. Para que esta se pueda llevar a cabo es necesario un repositorio o biblioteca de componentes que cumplan con estándares que garanticen su confiabilidad y especificaciones solicitadas por un nuevo sistema. De aquí la importancia de un modelo de componentes, definido en el inciso anterior.

La construcción de componentes reutilizables se basa en métodos de ingeniería de software. La construcción se puede efectuar empleando lenguajes convencionales de programación, ya sea lenguajes de tercera o cuarta generación, incluso generadores de código, técnicas de programación visual o incluso otros métodos más avanzados.

La mayoría de componentes existentes en la actualidad fueron desarrollados por la misma organización que los utiliza y se basan en los requerimientos solicitados para una aplicación, en vez de haber sido desarrollados de manera genérica para atender diversas peticiones. Es necesario, por lo tanto, realizar cambios sobre estos componentes para que sean adaptables a cualquier sistema, lógicamente, esto conlleva costos por lo que surge la necesidad de evaluar entre costos asociados a reutilizar el componente y si los ahorros asociados a la reutilización de este componente son favorables contra el costo para hacer que ese componente sea reutilizable.

Para evaluar la primera opción tiene que verificarse si el componente implementa una o más acciones del dominio estable, estas acciones de dominio

indican las funciones genéricas que pueden ser implementadas sobre el mismo dominio de aplicación. Si el componente contiene varias de estas funciones de dominio quiere decir que su reutilización generara una ventaja.

Cuando se evalúan costos entre ahorro en la implementación y costos asociados a hacer un componente reutilizable, se tienen que evaluar todos los cambios requeridos, estos incluyen, cambios en documentación, certificación del componente y costes involucrados para garantizar que un componente sea genérico.

Que un componente sea o no reutilizable, depende del dominio de aplicación al que pertenece y el alcance en funcionalidad.

Mientras un componente sea más abstracto más amplia será su reutilización, pero viene acompañado de consecuencia como que si el componente sea más complejo, esté más cargado de operación y por lo tanto sea más difícil de entender.

Debido a esto, para que un componente tenga un alto nivel de reutilización debe contar con interfaces genéricas que se adapten a distintos sistemas en los que se planea usar, sin embargo, la reusabilidad añade complejidad.

Cuando se genera un componente reutilizable es necesario encontrar un punto de equilibrio entre generalidad y comprensibilidad.

4.5. Análisis y diseño para la reutilización

Para describir los requerimientos de una determinada aplicación se crean modelos de datos funcionales y de comportamiento, para continuar se utilizan

especificaciones por escrito, que describan los modelos para obtener una descripción precisa de los requerimientos.

Cuando se analiza un sistema basado en la idea de reutilizar se evalúa el modelo de análisis para determinar aquellos elementos del modelo que pueden ser sustituidos por componentes ya existentes. El problema de este proceso consiste en el análisis que se haga para alcanzar correspondencias de especificaciones, es decir, hacer coincidir requerimientos con funciones proporcionadas por los componentes.

Para incluir un componente dentro del análisis de un sistema no basta con que coincidan los requerimientos con las funciones, hay que tener en cuenta también, cuán objetiva resulta la reutilización y centrarse en los principios básicos de un buen diseño, de aquí, que el análisis debe ser desarrollado por ingenieros de software con sólidos conceptos en análisis, diseño y reutilización de componentes. Para el diseño basado en reutilización deben considerarse ciertos aspectos:

- Datos estándar: incluye conocer el domino de aplicación del nuevo sistema y de los componentes que se estén considerando reutilizar.

- Protocolos de interfaz: deben establecerse tres niveles de interfaz de los componentes considerados:

    - La naturaleza de las interfaces intramedulares.

    - Las interfaces técnicas de cada componente (no humano).

- La interfaz hombre-máquina que define el uso directo que cada usuario tendrá con el componente.

Cuando el diseñador tiene estos elementos definidos, ya poseerá un marco de referencia con el cual puede llevar a cabo el diseño.

4.6. Modelos de desarrollo basado en componentes

Se han propuesto muchos modelos de componentes, se pueden mencionar CORBA de OMG, el modelo Enterprise Java Beans de Oracle y el modelo COM+ de Microsoft.

Dado que el impacto futuro de la reutilización y de la ingeniería de software basada en componentes en la industria es enorme, grandes compañías de software han propuesto los estándares de componentes mencionados:

- OMG/CORBA: grupo de gestión de objetos (Object Management Group) publica una arquitectura común de distribución de objetos. Estos proporcionan toda una gama de servicios que hace posible que los componentes reutilizables tengan comunicación con otros independientes de su ubicación dentro del sistema, para tal función los componentes usan un lenguaje de definición de interfaces (LDI) para cada componente.

- COM de Microsoft: modelo de objetos para componentes que proporciona una plataforma para utilizar los componentes elaborados por varios fabricantes dentro de una única aplicación bajo el sistema operativo Windows. Cuánta con dos elementos esenciales, las interfaces

y un conjunto de mecanismos para pasar mensajes entre interfaces de componentes. Las interfaces se registran con el sistema lo que hace posible sean utilizables por cualquier usuario y tengan comunicación con cualquier componente.

- Componentes Java Bean de Oracle: es una infraestructura portátil e independiente de la plataforma que utiliza el lenguaje de programación Java. El sistema JavaBean amplia el applet de Java para acoplar los componentes de software más sofisticados, necesarios para el desarrollo basado en componentes.

Entre las características que deben ofrecer los modelos, son identificadores únicos para cada componente:

- Definir la forma de empaquetado para el despliegue de componentes.

- La forma en que podrán ser reemplazados o modificados los componentes existentes.

- Contener la documentación necesaria que permita decidir a un desarrollador sobre la implementación de un componente.

Los modelos no solo definen estándares, también proporcionan el middleware que proporciona el soporte para los componentes ejecutables. Un modelo de componentes proporciona los siguientes servicios para la implementación de componentes:

- Servicios de plataforma: permiten a los componentes la comunicación entre sí.

- Servicios horizontales: definen estándares de comunicación y gestión, entre ellos gestión de componentes, gestión de transacciones, gestión de recursos, concurrencia, persistencia y protección. La disponibilidad de estos servicios reduce los costes de desarrollo de componentes.

4.7. Especialización de productos software

La reutilización de componentes software presenta agilidad en la construcción de sistemas software empresarial como los mencionados a continuación.

4.7.1. Herramientas enterprise resource planning (ERP)

El sistema ERP genérico integra un gran número de módulos y componentes que se relacionan de varias formas para adaptarse a los requerimientos de un cliente y crear un sistema específico.

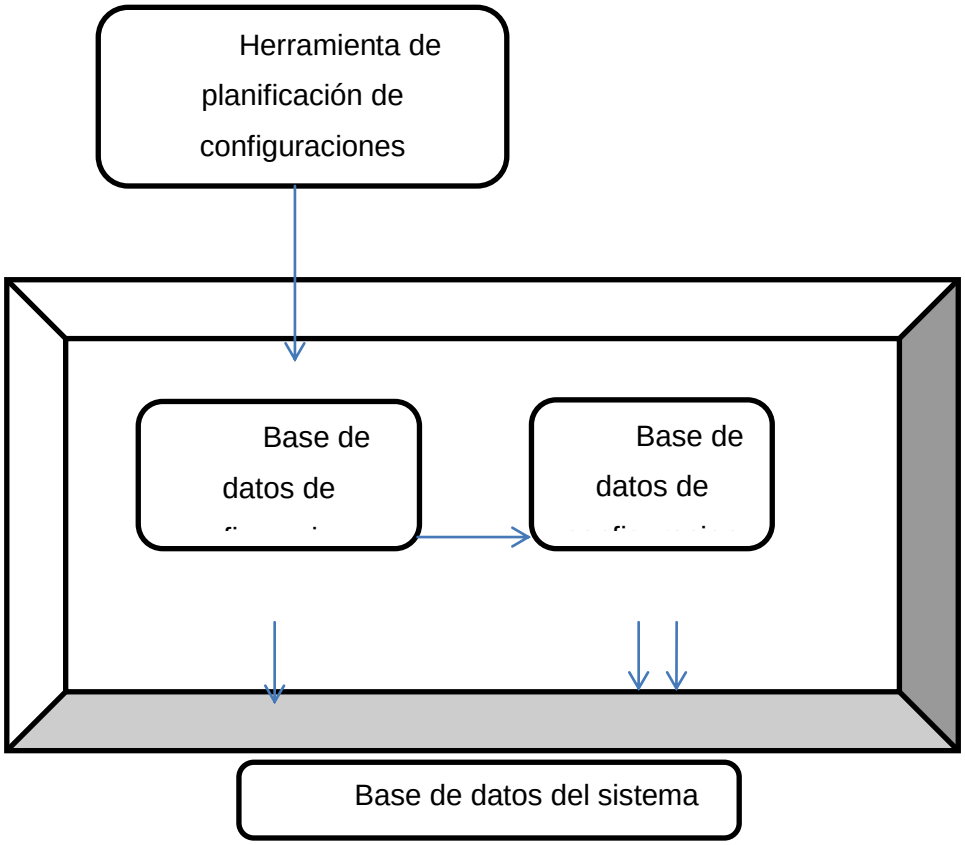
El proceso de configuración durante el diseño implica seleccionar qué módulos se van a requerir configurar estos de modo individual, definir la integración con otros módulos, añadir reglas de negocio y definir la estructura de la base de datos del sistema.

Los sistemas ERP son el ejemplo más extendido sobre la reutilización de componentes, este tipo de sistema es utilizado por la mayoría de las organizaciones grandes para gestionar sus operaciones, aun cuando la funcionalidad se ve bloqueada muchas veces por la funcionalidad del núcleo genérico.

Es en este proceso donde se ven reflejadas las ventajas de un sistema basado en componentes, el sistema comienza con una plataforma genérica y a continuación, se desarrolla modificando, integrando y extendiendo módulos para crear un sistema específico que proporciona las funcionalidades requeridas por el cliente.

Esta operación implica cambiar el código núcleo del sistema base, para que sea posible una mayor flexibilidad.

Figura 7. **Configuración de un sistema ERP**



Fuente: elaboración propia.

4.7.2. Herramientas customer relationship management (CRM)

Es una estrategia de negocio que busca la fidelización de clientes, su objetivo principal es optimizar la relación entre empleados y clientes. A través de ciertos módulos genéricos que integran la información actualizada sobre las necesidades de una persona.

Módulos de un CRM:

- Ventas
- Servicios
- *Marketing*

Se considera a este tipo de herramientas dentro del grupo de las denominadas de colaboración, donde se integra un conjunto de componentes informáticos en un solo sistema con varios usuarios concurrentes en distintas estaciones de trabajo, brindan un conjunto de herramientas integradas que satisfacen una amplia gama de necesidades.

Entre sus características se encuentran:

- Gestiona la documentación en archivos para su potencial reutilización
- Colabora en el proceso de toma de decisiones
- Soporta funciones de gestión de proyectos
- Manejo de perfiles de usuario
- Facilita el trabajo de un equipo en distintas estaciones de trabajo

Para construir un sistema CRM se ejecuta una planeación, modelado, diseño e implementación de procesos de negocio basado en un enfoque de negocio y orientado a las necesidades específicas de un cliente, mediante el uso de componentes reutilizables que facilitan la integración entre varios módulos y sistemas informáticos ya existentes, lo que agiliza el proceso de implementación y adaptación de la organización al sistema de software.

Este tipo de soluciones están creadas teniendo en cuenta particularidades de cada sector que requiera el servicio, haciendo uso de los componentes adecuados y código fuente reutilizable se pueden implementar soluciones personalizadas optimizando el costo de creación e implementación.

Entre las ventajas de utilizar sistemas CRM se encuentran:

- Independencia de la plataforma: se puede extender con componentes y programas en cualquier plataforma y sistema operativo.

- Código reutilizable: pueden integrarse y extenderse mediante estándares abiertos.

- Componentes reutilizables: se pueden escribir componentes reutilizables y utilizar componentes de terceros.

4.7.3. Herramientas business process management (BPM)

BPM es una metodología de mejoramiento continuo del rendimiento de los procesos. Es sobre todo, una disciplina de gestión basada en la mejora iterativa de los procesos de negocio. Es una tecnología que impulsa el trabajo

en la organización, soportado en el monitoreo, la optimización y la trazabilidad de los procesos de negocio.

Un sistema BPM se encarga de describir cómo van a interactuar sistemas y personas para realizar una actividad.

La mejor manera de integrar el proceso con los varios sistemas existentes de la empresa es a través de la definición de un servicio reutilizable, principio central de la filosofía SOA. Además, los analistas funcionales que diseñan los procesos con las herramientas BPM son algunas veces los que expresan, por primera vez, la necesidad de un nuevo servicio y tienen un papel importante que jugar en la fase de definición del servicio.

Al igual que las herramientas mencionadas anteriormente, un sistema BPM maneja una arquitectura de componentes reutilizables lo que significa que tiene muchas piezas móviles que integran un solo sistema de software.

Este tipo de herramientas fomentan la reutilización como mecanismo para aumentar la agilidad y la flexibilidad empresarial. Si se diseñan adecuadamente, los sistemas de negocio se puede adaptar de acuerdo con las necesidades cambiantes del entorno en lugar de estar limitado por aplicaciones frágiles y altamente integradas que restringen la agilidad.

5. IMPACTO EN LA CALIDAD DE LOS SISTEMAS BASADOS EN COMPONENTES

El presente capítulo recoge la exploración estadística de los resultados en una modalidad de encuesta usada para medir el impacto en la calidad de un sistema de software desarrollado a partir de componentes que cumplen con un modelo de calidad.

En la ingeniería del software basada en componentes, la calidad es un aspecto que tiene que ver especialmente con la satisfacción de los usuarios finales respecto a los sistemas de software construidos. Frecuentemente, la medida de la percepción de clientes se establece como parámetro principal que da noticia a la organización sobre la calidad del servicio prestado.

La exploración de la información recogida en la encuesta muestra, sobre todo, cuáles son las características de calidad de un componente que agregan calidad a un sistema de software y cuáles características de calidad de componentes que no influyen en la calidad del sistema de software que integran.

El resultado obtenido se relaciona mediante el análisis estadístico de variables consideradas hipotéticamente discriminantes y variables que miden el nivel de impacto en la calidad de un sistema. Las variables discriminantes describen algunos rasgos principales del público encuestado y permiten establecer en qué medida dichos rasgos condicionan la valoración de los aspectos de calidad evaluados.

5.1. Análisis estadístico de resultados sobre el impacto en la calidad de la ingeniería del software basada en componentes

El cuestionario de la encuesta distingue dos tipos principales de variables: discriminantes y valorativas.

Las primeras sirven para perfilar sociológica y actitudinalmente a la población encuestada en virtud de su sexo, edad y área de trabajo dividida en dos segmentos genéricos.

Las segundas sirven para registrar el impacto en la calidad que los encuestados consideran tienen las características de calidad de un componente individual sobre un sistema software completo. La medición de estas variables dentro de la encuesta se basó en el modelo de calidad 9126-1 que define características de calidad de software y componentes.

También es importante indicar que la población encuestada fue limitada a personas que han usado o desarrollado un sistema software corporativo o empresarial.

Se ofrecen a continuación los resultados meramente descriptivos de cada ítem del cuestionario. En primer lugar, los estadísticos discriminantes y en segundo, el análisis estadístico de datos. Para finalizar se realiza una prueba de correlación entre las variables y una determinación de fiabilidad de los datos evaluados.

El análisis estadístico de los datos de la encuesta ha sido realizado con el programa informático SPSS versión 19.0.0.

5.1.1. Estadísticos descriptivos

Las variables discriminativas representan segmentos sociales en la población encuestada.

Tabla VII. **Análisis estadístico para variables discriminativas**

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	Masculino	47	87,0	87,0	87,0
	Femenino	7	13,0	13,0	100,0
	Total	54	100,0	100,0	

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válidos	Tecnología	32	59,3	59,3	59,3
	Otra	22	40,7	40,7	100,0
	Total	54	100,0	100,0	

Fuente: elaboración propia, con base al programa SPSS.

De la población entrevistada 87 por ciento resulta ser de sexo masculino y 13 por ciento femenino, asignada como (1) y (2) para el análisis de valores nominales.

El área de trabajo, dividida en dos segmentos genéricos, indica que el 59,3 por ciento de las personas encuestadas tienen experiencia en el campo tecnológico y el 40,7 por ciento pertenecen a otra área donde son usuarios de un sistema de software.

La frecuencia de la edad está distribuida en varios segmentos que van desde 19 hasta 35 años, donde el rango más alto indica un promedio de 24 o 25 años.

Se establecieron como variables valorativas todos aquellos aspectos de calidad de componentes que impactan sobre la calidad de un sistema software, la valoración de estos aspectos se ha medido con una escala numérica graduada de 0 (mínimo impacto) al 5 (impacto alto) en la idea de que es un tipo gradiente al que las personas se sienten habituados y pueden recoger con alta fiabilidad la opinión como una escala de intervalo.

Actuando de esta manera, es posible emplear estadísticos como la media o la desviación estándar que contribuyen notablemente a aumentar y mejorar la comprensibilidad y la interpretación de los resultados.

Tabla VIII. **Estadísticos descriptivos para todas las variables**

		Funcionalidad	Fiabilidad	Eficiencia	Facilidad de uso	Mantenibilidad	Portabilidad
N	Válidos	54	54	54	54	54	54
	Perdidos	0	0	0	0	0	0
Media		**4,092593**	3,8750	**4,141975**	3,864198	3,604938	3,85
Mediana		4,200000	4,0000	4,166667	4,000000	3,666667	4,00
Moda		4,6000	4,25	5,0000	5,0000	4,3333	4
Desv. típ.		**0,6188400**	0,70502	**0,8916144**	0,9461796	0,9800399	1,071
Asimetría		-1,128	-1,352	-1,189	-0,540	-0,630	-0,842
Error típ. de asimetría		0,325	0,325	0,325	0,325	0,325	0,325
Curtosis		1,453	3,940	1,710	-0,474	-0,079	0,268
Error típ. de curtosis		0,639	0,639	0,639	0,639	0,639	0,639

Fuente: elaboración propia, con base al programa SPSS.

A partir de los resultados se puede observar que la característica de calidad de los componentes que más impacta de manera positiva sobre la calidad de un sistema software, es la funcionalidad con un valor de 4,09; ofrece, además, una desviación típica discreta, indicativa de una considerable homogeneidad en las valoraciones, seguido de la eficiencia (4,14) también, con una desviación menor a 1, aunque mayor a la desviación indicada por la funcionalidad.

Inversamente, las características de componentes que tienen un menor impacto sobre un sistema software son, en orden de impacto: fiabilidad (3,87), facilidad de uso (3,86), mantenibilidad (3,6) y portabilidad (3,85). La portabilidad se encuentra en último lugar debido a la desviación estándar presentada.

Con esta primera aproximación de resultados se puede decir que, si los componentes utilizados para una aplicación de software cuentan con un alto índice de funcionalidad y eficiencia, el sistema tendrá un impacto positivo en ambas características y por lo tanto mejorará su calidad final.

De igual manera, se deduce que si los componentes de software tienen altos índices de fiabilidad, facilidad de uso, mantenibilidad y portabilidad, no impactan en la calidad final de sistema. Es decir, un componente podría no poseer la característica de facilidad de uso y, aun así el sistema construido puede ser de calidad o podría ser un componente sencillo de usar y poseer la documentación adecuada, pero estos factores no repercuten en la calidad del sistema construido.

Pero afirmar esto es demasiado pronto, ya que se está evaluando la totalidad de la población encuestada sin distinguir los posibles perfiles

valorativos, que tal vez ofrecerían los encuestados en virtud de sus características individuales.

La hipótesis es: variables como sexo, área de trabajo y edad pueden influir en las puntuaciones otorgadas a los diferentes aspectos de calidad evaluados, se analizará a continuación, si se cumple esta hipótesis para la deducción de conclusiones.

5.1.2. Relaciones entre variables discriminantes y valorativas

Se ha procedido a cruzar todas las variables discriminantes con las valorativas, expresando los cruces mediante el coeficiente de asociación de Spearman con el objetivo de observar cuáles arrojan valores relevantes y estadísticamente significativos y determinar qué variables descriptivas resultan discriminantes en la valoración de calidad presentada.

La matriz siguiente representa los cruces entre variables y muestra el índice de correlación entre cada una, es importante destacar que se utilizó el coeficiente de asociación de Spearman debido a que las variables no presentan una distribución normal.

Tabla IX. **Matriz de correlaciones**

		Funcionalidad	Fiabilidad	Eficiencia	Facilidad de uso	Mantenibilidad	Portabilidad
SEXO	Coeficiente de correlación	0,105	0,098	0,043	-0,137	-0,135	-0,052
	Sig. (bilateral)	0,449	0,481	0,755	0,321	0,329	0,709
	N	54	54	54	54	54	54
EDAD	Coeficiente de correlación	0,068	-0,243	0,014	-0,146	-0,133	-0,176
	Sig. (bilateral)	0,625	0,077	0,918	0,292	0,336	0,204
	N	54	54	54	54	54	54
AREA	Coeficiente de correlación	-0,046	-0,241	0,175	-0,114	-0,206	-0,098
	Sig. (bilateral)	0,739	0,079	0,207	0,414	0,136	0,482
	N	54	54	54	54	54	54

Fuente: elaboración propia, con base al programa SPSS.

Se puede observar que, dadas las relaciones de variables discriminantes con valorativas, ningún coeficiente de correlación se aproxima a 1, y de igual manera ningún valor de P (Sig. Bilateral) es menor a 0,5, por lo que la hipótesis nula se rechaza y se acepta la hipótesis inversa que, dice que las variables discriminativas no afectan en las puntuaciones otorgadas.

Como las variables discriminantes no afectan los rangos medidos se procederá a realizar una prueba de fiabilidad para la evaluación de los datos, para esto se utilizará el índice Alpha de Cronbach que evalúa el promedio de correlaciones entre variables.

Mientras el valor de Alpha de Cronbach se aproxime más a 1, más fiable será la evaluación de datos.

5.1.3. Determinación de Alpha de Cronbach

Para la determinación del índice se ha excluido la variable edad generando el siguiente resultado:

Tabla X. **Alpha de Cronbach sin la variable edad**

Alfa de Cronbach	Alfa de Cronbach basada en los elementos tipificados	N de elementos
0,666	0,565	8

Fuente: elaboración propia, con base al programa SPSS.

El índice de fiabilidad Alpha de Cronbach no viene acompañado de ningún valor que permita rechazar la hipótesis de fiabilidad en la escala. No obstante, cuanto más se aproxime a su valor máximo, 1, mayor es la fiabilidad de la escala.

En la primera evaluación el índice indicaba 0,131, lo que mostraba poca fiabilidad en la evaluación del modelo si se consideraban todas las variables de la encuesta.

Si se descarta la variable edad del análisis de respuestas se obtiene un valor de 0,666 lo que incrementa la fiabilidad de los resultados. El incremento en la fiabilidad se debe a que la correlación entre la variable edad contra el resto de variables no es significativa para el modelo.

Ahora que se ha eliminado la variable edad del análisis del modelo se procederá a evaluar el nivel de impacto relacionado a cada variable discriminativa buscando obtener los mismos resultados que la evaluación individual de frecuencias realizada al inicio.

Tabla XI. **Fiabilidad de variables contra variable sexo**

	SEXO		Estadístico	Error típ.
FUNCIONALIDAD	Masculino	Media	4,102128	0,0792827
		Varianza	0,295	
		Desv. típ.	0,5435349	
	Femenino	Media	4,028571	0,3986371
		Varianza	1,112	
		Desv. típ.	1,0546947	
FIABILIDAD	Masculino	Media	3,8457	0,10524
		Varianza	0,521	
		Desv. típ.	0,72147	
	Femenino	Media	4,0714	0,22304
		Varianza	0,348	
		Desv. típ.	0,59010	
EFICIENCIA	Masculino	Media	4,134752	0,1285048
		Varianza	0,776	
		Desv. típ.	0,8809842	
	Femenino	Media	4,190476	0,3907471
		Varianza	1,069	
		Desv. típ.	1,0338197	
FACILIDAD DE USO	Masculino	Media	3,929078	0,1302039
		Varianza	0,797	
		Desv. típ.	0,8926327	
	Femenino	Media	3,428571	0,4697984
		Varianza	1,545	
		Desv. típ.	1,2429696	

Continuación de la tabla XI.

MANTENIBILIDAD	Masculino	Media	3,645390	0,1428043
		Varianza	0,958	
		Desv. típ.	0,9790172	
	Femenino	Media	3,333333	0,3849002
		Varianza	1,037	
		Desv. típ.	1,0183502	
PORTABILIDAD	Masculino	Media	3,87	0,154
		Varianza	1,114	
		Desv. típ.	1,055	
	Femenino	Media	3,71	0,474
		Varianza	1,571	
		Desv. típ.	1,254	

Fuente: elaboración propia, con base al programa SPSS.

Cuando se relaciona la variable discriminante sexo contra los rangos obtenidos, se llega a la conclusión que, las características de componentes que tienen un impacto alto sobre la calidad de un sistema software son: funcionalidad, fiabilidad y eficiencia. Aunque la fiabilidad solo aporta una medida alta evaluada contra la variable sexo = femenino y no contra sexo = masculino.

Como el resultado no es el mismo en ambos casos, se descarta la eficiencia como característica que aporta impacto positivo en la calidad final de un sistema software.

Tabla XII. **Fiabilidad de variables contra variable área de trabajo**

	ÁREA DE TRABAJO		Estadístico	Error típ.
FUNCIONALIDAD	Tecnología	Media	4,112500	0,1121410
		Varianza	0,402	
		Desv. Típ	0,6343653	
	Otra	Media	4,063636	0,1298593
		Varianza	0,371	
		Desv. típ.	0,6090941	
FIABILIDAD	Tecnología	Media	4,0391	0,08408
		Varianza	0,226	
		Desv. típ.	0,47565	
	Otra	Media	3,6364	0,19304
		Varianza	0,820	
		Desv. típ.	0,90543	
EFICIENCIA	Tecnología	Media	4,031250	0,1593553
		Varianza	0,813	
		Desv. típ.	0,9014499	
	Otra	Media	4,303030	0,1859793
		Varianza	0,761	
		Desv. típ.	0,8723203	
FACILIDAD DE USO	Tecnología	Media	3,958333	0,1567979
		Varianza	0,787	
		Desv. típ.	0,8869827	
	Otra	Media	3,727273	0,2200140
		Varianza	1,065	
		Desv. típ.	1,0319569	
MANTENIBILIDAD	Tecnología	Media	3,812500	0,1338176
		Varianza	0,573	
		Desv. típ.	0,7569866	
	Otra	Media	3,303030	0,2537919
		Varianza	1,417	
		Desv. típ.	1,1903896	

Continuación de la tabla XII.

PORTABILIDAD	Tecnología	Media	3,94	0,185
		Varianza	1,093	
		Desv. típ.	1,045	
	Otra	Media	3,73	0,239
		Varianza	1,255	
		Desv. típ.	1,120	

Fuente: elaboración propia, con base al programa SPSS.

Cuando se relaciona la variable área de trabajo contra los rangos obtenidos se puede observar el mismo comportamiento que en la evaluación anterior donde se considera que la fiabilidad tiene un impacto positivo, solo en uno de los casos, con la variable Área de Trabajo = Tecnológico y no se considera con impacto sobresaliente en otras áreas de trabajo.

El resto de características analizadas se acoplan a la tabla de frecuencia presentada al inicio del capítulo, donde las características de componentes que realmente tienen un impacto positivo sobre la calidad de un sistema software son: funcionalidad y eficiencia. Mientras el resto: fiabilidad, facilidad de uso, mantenibilidad y portabilidad no tienen impacto considerable sobre un software basado en componentes.

5.2. Aproximación a un modelo de calidad para ISBC

En el capítulo anterior se presentaron dos propuestas de modelos de calidad para la ingeniería de software basada en componentes, ambos modelos definen distintas características que debe poseer un software basado en componentes durante las etapas de su ciclo de vida.

Con los resultados medidos en la encuesta se va a definir el modelo de calidad que mejor se adapta a la arquitectura de software basada en componentes, describiendo las características de calidad de un sistema software que tienen un impacto positivo y las características de calidad individuales de componentes que generan este impacto.

Según los resultados obtenidos, las características de calidad de componentes que impactan positivamente la calidad de un software son: funcionalidad y eficiencia; estas se dividen en:

- Precisión
- Exactitud computacional
- Cifrado de datos y seguridad
- Capacidad de auditoria
- Conformidad de estándares
- Certificaciones
- Compatibilidad hacia atrás
- Tiempo de respuesta

El modelo de calidad propuesto por el autor Len Bass en el libro Software Architecture in Practice, divide las características como se muestra a continuación:

- Rendimiento
- Usabilidad
- Mantenibilidad
- Portabilidad
- Costo
- Tiempo de vida previsto del sistema

- Integridad conceptual
- Construbilidad

Si se comparan los resultados obtenidos con el modelo de calidad propuesto por Len Bass se puede observar que los únicos atributos que empatan contra los resultados obtenidos son rendimiento (en la etapa de arquitectura y ejecución) e integridad conceptual (durante la etapa de arquitectura software general).

Atributos como: mantenibilidad, portabilidad, usabilidad y construbilidad quedaron descartados durante la evaluación de resultados, el costo y tiempo de vida previsto del sistema, ni siquiera eran factores considerados en atributos ni sistemas software.

El modelo de calidad propuesto por el autor Jan Bosch en el libro Design & Use of Software Architectures, divide las características como se muestra a continuación:

- Rendimiento
- Mantenibilidad
- Fiabilidad
- Seguridad física
- Seguridad de acceso

Si se comparan los resultados obtenidos con el modelo de calidad propuesto por Jan Bosch se obtiene un mejor resultado al empatar varios de los atributos propuesto por el autor.

El atributo rendimiento propuesto en el modelo se ajusta al atributo eficiencia devuelto por los resultados, distribuidos en características como tiempos de respuesta y requisitos de memoria y disco.

Los atributos seguridad física y seguridad de acceso se ajustan al atributo funcionalidad devuelto por los resultados, distribuidos en características como cifrado de datos y seguridad y capacidad de auditoría.

Mantenibilidad resulta ser uno de los atributos descartados durante la evaluación de resultados, por lo tanto no se empata con esta parte del modelo. Finalmente fiabilidad que, aunque se descartó de los resultados, era una de las características con mayor rango de impacto entre las menos significativas.

Es necesario indicar que la evaluación de resultados, por medio de estadística descriptiva, no permite conocer el nivel de impacto en la calidad del software construido con componentes de calidad, tampoco indica qué características de calidad de componentes tienen un impacto positivo y cuáles no resultan significativos. Para obtener estos resultados se va a definir un modelo lineal multivariable y una prueba de confianza que valide dicho modelo.

5.3. Medición del impacto en la calidad

Actualmente, no existe una técnica que sea capaz de probar cuál es el nivel de impacto sobre un sistema software si este se construye utilizando componentes que posean características de calidad. Para dicho fin, se definirá un modelo a base de un conjunto de relaciones buscando representar de forma sencilla una realidad empírica.

Se presenta la predicción de un modelo lineal multivariable, el cual se centra en estudiar conjuntos de variables representadas por características de componentes y factores discriminantes de una población que determinarán cuál es el impacto en la calidad de un sistema software basado en componentes.

Por medio de un proceso de regresión lineal se establecerán las variables que colaboran de manera significativa, con el objetivo de obtener un modelo que explique el comportamiento de una variable (variable endógena, explicada o dependiente), que se indica por:

Y: nivel de impacto que tienen las características de calidad de componentes sobre la calidad del software.

Medido por un rango de números enteros finitos que van desde N_0 hasta N_1. Donde N_1 representa el máximo impacto posible y N_0 representa el valor mínimo o valor sin impacto.

Utilizando la información proporcionada por los valores tomados por un conjunto de variables (explicativas, exógenas o independientes), que se denotarán por:

Variables poblacionales discriminantes:

- X1: sexo
- X2: área de trabajo
- X3: edad

Variables valorativas (características de calidad de componentes software):

- X4: funcionalidad
- X5: fiabilidad
- X6: eficiencia
- X7: facilidad de uso
- X8: mantenibildad
- X7: portabilidad

Dada la robustez de la regresión, las variables independientes valorativas representan una escala de impacto en la calidad del software construido a base de componentes y las variables discriminantes se han transformado en variables nominales ficticias. Donde sexo está definida como 0=masculino y 1=femenino; área de trabajo como 0=tecnológica y 1=otra.

El modelo lineal esperado estará representado por:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + ... b_k X_k + u, N_0 <= Y <= N_1$$

Donde los coeficientes b_1 , b_2 , ... , b_k denotan la magnitud del efecto de las variables explicativas.

Tabla XIII. **Validación del modelo**

	Impacto	Sexo	Área de trabajo	Edad	Funciona-bilidad	Fiabilidad	Eficiencia	Facilidad de uso	Manteni-bilidad	Porta-bilidad
Impacto	1	-0,019	0,019	0,066	0,58	0,474	0,542	0,734	0,548	0,477
Sexo	**-0,019**	1	-0,096	-0,074	-0,04	0,109	0,021	-0,179	-0,108	-0,05
Área de trabajo	**0,019**	-0,096	1	0,231	-0,039	-0,283	0,151	-0,121	-0,258	-0,097
Edad	**0,066**	-0,074	0,231	1	0,025	-0,321	0,012	-0,18	-0,306	-0,247
Funcionalidad	0,58	-0,04	-0,039	0,025	1	0,254	0,325	0,303	0,263	0,163
Fiabilidad	0,474	0,109	-0,283	-0,321	0,254	1	0,124	0,396	0,341	0,281
Eficiencia	0,542	0,021	0,151	0,012	0,325	0,124	1	0,401	0,058	0,259
Facilidad de uso	0,734	-0,179	-0,121	-0,18	0,303	0,396	0,401	1	0,572	0,563
Mantenibilidad	0,548	-0,108	-0,258	-0,306	0,263	0,341	0,058	0,572	1	0,602
Portabilidad	0,477	-0,05	-0,097	-0,247	0,163	0,281	0,259	0,563	0,602	1

Fuente: elaboración propia, con base al programa SPSS.

El coeficiente de Pearson indica la medida de correlación lineal entre pares de variables, de aquí se puede definir cuál será la relación entre las variables independientes y la variable dependiente Y (nivel de impacto).

La tabla de correlaciones indica que la variable independiente sexo tiene una correlación negativa sobre la variable dependiente Y, esto indica que cuando una de estas variables aumenta, la otra disminuye en proporción constante. Se descarta entonces la variable sexo de la definición del modelo.

Las variables área de trabajo y edad presentan una correlación entre 0 y 1, aunque el valor sea pequeño no se puede descartar, pues no llega a ser negativo, por lo que no descarta que las variables sean independientes.

El nivel de colinealidad representa la correlación entre variables explicativas del modelo. Se toleran valores entre -0,7 y 0,7. De ser mayor este

factor es necesario eliminar una de las variables, pues indicaría que una explica a la otra y distorsionaría el error estándar llevando a conclusiones incorrectas.

De la matriz expuesta anteriormente, se puede observar que no existen variables independientes que posean colinealidad entre ellas, por lo tanto hasta ahora, se aceptan todas las variables, excepto sexo, que ya se había rechazado en el proceso anterior.

5.3.1. Regresión lineal multivariable

Para la deducción del modelo lineal se ha aplicado la técnica de Regresión Stepwise o regresión hacia adelante, donde se empieza a deducir el modelo con una sola variable independiente y se van añadiendo aquellas variables e interacciones que mejoran significativamente el modelo.

Tabla XIV. **Modelo lineal**

Modelo	Coeficientes no estandarizados		Coeficientes tipificados	Sig.	Intervalo de confianza de 95,0 para B	
	B	Error típ.	Beta		Límite inferior	Límite superior
(Constante)	-1,206	9,357		0,898	-20,029	17,617
Facilidad de uso	**4,276**	1,034	0,352	**0,000**	2,197	6,355
Funcionalidad	**4,589**	1,261	0,247	**0,001**	2,053	7,126
Eficiencia	**3,520**	0,909	0,273	**0,000**	1,692	5,349
Mantenibilidad	**3,182**	0,935	0,271	**0,001**	1,302	5,062
Edad	**1,050**	0,251	0,278	**0,000**	0,545	1,555
Fiabilidad	**3,832**	1,141	0,235	**0,002**	1,536	6,128

Fuente: elaboración propia, con base al programa SPSS.

Con el método utilizado para la deducción del modelo se ha llegado a la sexta iteración que incluye únicamente las variables mostradas, omitiendo: sexo, área de trabajo y portabilidad, cumpliendo el punto anterior donde la variable sexo quedaba descartada por tener una correlación negativa sobre la variable dependiente. Se han descartado, además, otras dos variables que no presentaban significancia en la evaluación.

5.3.2. Explicación del modelo lineal

Cuando el nivel (índice) de calidad en la característica facilidad de uso de un componente varía una unidad y los índices de calidad en el resto de atributos se mantienen constantes, el impacto en la calidad del software construido se incrementará en 4,27.

Cuando el nivel (índice) de calidad en la característica funcionalidad de un componente varía una unidad y los índices de calidad en el resto de atributos se mantienen constantes, el impacto en la calidad del software construido se incrementará en 4,59.

Y así sucesivamente con todas las características de componentes incluidas en el modelo.

La variable edad incrementa 1,05 el impacto en la calidad cuando el índice de todas las características se mantienen constantes, por lo que el impacto no se ve afectado considerablemente con este valor.

Finalmente la ecuación quedaría definida como:

$$Y = -1{,}206 + 4{,}276X_7 + 4{,}589X_4 + 3{,}52X_6 + 3{,}182X_8 + 1{,}05X_3 + 3{,}832X_5$$

5.3.3. Error estándar de la estimación

En la tabla XV se muestran los errores típicos de cada uno de los modelos antes de llegar al más exacto el cual fue el que dedujo la ecuación lineal (6).

Tabla XV. **Pruebas de hipótesis del modelo lineal**

Modelo	R	R cuadrado	R cuadrado corregida	Error típ. de la estimación
1	0,734	0,539	0,530	7,88262
2	0,824	0,680	0,667	6,63421
3	0,846	0,715	0,698	6,31704
4	0,863	0,744	0,723	6,04792
5	0,887	0,787	0,764	5,58112
6	0,910	**0,828**	0,806	**5,06547**

Fuente: elaboración propia, con base al programa SPSS.

Del error estándar presentado por el modelo 6 se puede concluir que la distancia promedio de los valores alrededor de la ecuación de regresión y por tanto la dispersión de los valores observados es de 5,06.

5.3.4. Coeficiente de determinación del modelo

Haciendo uso de la misma tabla de resumen de modelos se obtiene el valor de R^2 que representa el coeficiente de determinación menos optimista para una muestra pequeña. El modelo 6 muestra un coeficiente de R^2 =0,828 y un valor de 0.806 en el valor R cuadrado corregida para valor de estimación, con lo que se puede concluir que aproximadamente el 82,8 por ciento del nivel

de impacto en la calidad de un software basado en componentes es explicado por el índice de calidad en los atributos individuales de cada componente utilizado.

5.3.5. Pruebas de hipótesis global e individual

Las pruebas de hipótesis se utilizan para investigar si las variables independientes tienen coeficientes significativos dentro del modelo. Para esto se determinarán las siguientes hipótesis

H_0 = B0 = B1 = B2… Bx = 0 (los coeficientes de regresión son igual a cero)

H_1 = al menos uno de los coeficientes de regresión no es cero.

Para este análisis se utilizará la tabla Anova que proporciona la variación de la variable dependiente.

Tabla XVI. **Anova del modelo**

Modelo		Suma de cuadrados	gl	Media cuadrática	F	Sig.
6	Regresión	5803,359	6	967,227	37,695	**0,000**
	Residual	1205,974	47	25,659		
	Total	7009,333	53			

Fuente: elaboración propia con base al programa SPSS.

La tabla ANOVA muestra un nivel de significancia de 0,000, por lo tanto p = 0,000 < 0,05, por lo que se rechaza la hipótesis nula H_0, aceptando H_1, lo que ratifica la relación entre las variables.

Para la prueba de hipótesis individual, se analizará el valor de significancia de cada uno de los coeficientes encontrados en la tabla utilizada para deducir el modelo. Los valores que se pueden observar varían entre 0,000 y 0,02, por lo que de la misma manera se rechaza la hipótesis nula y se acepta H_1.

5.3.6. Media poblacional y diagnósticos por caso

Como la distribución de datos mostrados en los resultados no presentan una distribución normal, la media poblacional se define como la sumatoria de los valores obtenidos de Y dividido el total de la población.

De aquí se obtiene que el valor de la media poblacional será:

$Y = 101,5555556$

El impacto máximo N_1 determinado por el modelo será:

$N_1 = 120$

Y el valor sin impacto N_0 determinado por el modelo será:

$N_0 = 71$

Se muestra en la tabla XVII el diagnóstico de los casos, donde se puede observar que de 54 casos evaluados, 28 sobrepasan la media poblacional.

Tabla XVII. **Diagnósticos por caso**

Número de casos	Residuo típ.	IMPACTO	Valor pronosticado	Residual
1	-0,111	114,00	114,5617	-0,56174
2	0,017	108,00	107,9162	0,08385
3	0,381	118,00	116,0711	1,92892
4	-0,021	111,00	111,1051	-0,10508
5	-0,050	117,00	117,2528	-0,25283
6	-0,064	105,00	105,3253	-0,32526
7	0,351	102,00	100,2196	1,78041
8	-0,036	106,00	106,1844	-0,18437
9	0,390	93,00	91,0255	1,97453
10	0,650	98,00	94,7064	3,29363
11	-0,079	109,00	109,4019	-0,40192
12	0,761	74,00	70,1431	3,85689
13	0,344	95,00	93,2578	1,74225
14	0,216	107,00	105,9084	1,09163
15	0,000	117,00	117,0002	-0,00022
16	0,421	112,00	109,8683	2,13169
17	-0,204	97,00	98,0344	-1,03436
18	0,713	102,00	98,3894	3,61057
19	0,279	98,00	96,5889	1,41110
20	0,013	120,00	119,9362	0,06377
21	0,359	93,00	91,1827	1,81728
22	-0,087	103,00	103,4409	-0,44094
23	0,299	98,00	96,4843	1,51572
24	-4,562	71,00	94,1079	-23,10786
25	-4,562	71,00	94,1079	-23,10786
26	-0,220	108,00	109,1166	-1,11656
27	0,008	117,00	116,9596	0,04043
28	-0,122	100,00	100,6161	-0,61609
29	0,509	95,00	92,4221	2;57794
30	-0,082	106,00	106,4142	-0,41422
31	-0,012	117,00	117,0623	-0,06228
32	0,569	114,00	111,1174	2,88264
33	0,348	106,00	104,2376	1,76235
34	0,333	96,00	94,3112	1,68878

Continuación de la tabla XVII.

35	-0,012	100,00	100,0600	-0,06003
36	-0,311	86,00	87,5729	-1,57290
37	-0,242	116,00	117,2235	-1,22352
38	0,504	101,00	98,4454	2,55464
39	0,136	96,00	95,3112	0,68877
40	0,165	117,00	116,1629	0,83709
41	-0,013	109,00	109,0661	-0,06615
42	0,089	94,00	93,5488	0,45121
43	0,123	107,00	106,3756	0,62442
44	-0,172	103,00	103,8706	-0,87055
45	0,428	88,00	85,8336	2,16636
46	0,418	92,00	89,8842	2,11577
47	0,441	94,00	91,7656	2,23444
48	0,137	90,00	89,3050	0,69503
49	0,503	103,00	100,4519	2,54810
50	-0,206	97,00	98,0429	-1,04292
51	0,403	88,00	85,9597	2,04030
52	0,320	108,00	106,3780	1,62197
53	0,091	107,00	106,5381	0,46185
54	0,449	90,00	87,7267	2,27334

Fuente: elaboración propia con base al programa SPSS.

CONCLUSIONES

1. El impacto de utilizar componentes de calidad durante la ingeniería basada en componentes está determinado por el modelo: Y = -1,206 + 4,276 X_7+4.589X_4+3.52X_6+3.182X_8+1.05X_3+3.832X_5. Donde Y representa el impacto en la calidad del software con un valor entero positivo en un rango de 71 a 120. La descripción del modelo y variables se ha documentado en el capitulo 5.

2. Basado en el modelo ISO/IEC 9126-1, las métricas que definen las características de calidad de un componente software son: precisión, exactitud computacional, seguridad, tiempo de respuesta, requisitos de memoria y disco; conformidad de estándares, compatibilidad hacia atrás, madurez, facilidad de aprendizaje, comprensión, operatividad y cambiabilidad.

3. Basado en el modelo ISO/IEC 9126-1, un sistema software se considera de calidad si posee las siguientes características: funcionalidad, confiabilidad, facilidad de uso, eficiencia, facilidad de mantenimiento y portabilidad. Cada una de estas características se adapta a un requerimiento de la calidad del software según la descripción mostrada en el capítulo 2.

RECOMENDACIONES

1. El presente documento deja abierta numerosas vías para proseguir la investigación realizada hasta ahora. El modelo presentado muestra a la facilidad de uso y funcionalidad como dos de las características que tienen un impacto positivo en la ingeniería del software basado en componentes. Es necesario realizar una investigación sobre la correlación entre características que no impactan directamente a la calidad y las dos características significativas mencionadas, para extender el conocimiento y obtener mejores resultados en la medición de calidad final de un producto software.

2. El modelo lineal descrito se adapta a la teoría de sistemas de información denominada Modelo de Aceptación de Tecnologías (TAM: Technology acceptance model). Se insta al lector a investigar sobre el tema para ampliar las conclusiones presentadas sobre cómo el modelo describe el impacto en la calidad durante el proceso de ingeniera basada en componentes.

3. Exhortar a las organizaciones que quieran realizar un análisis de calidad para los componentes, definir y consensuar modelos de calidad para productos específicos en diferentes ramas de aplicación.

4. Sin métricas para evaluar componentes no se puede conseguir una verdadera ingeniería del software. Por lo que es importante que la evaluación de la calidad de los componentes debe realizarse por entidades independientes, las organizaciones que consumen un

componente en este caso, al menos hasta que los fabricantes de componentes software adquieran la madurez necesaria para definir los índices de calidad en los productos que distribuyen.

BIBLIOGRAFÍA

1. AYALA, Claudia. *Construcción de una taxonómica de componentes COTS orientada a la gestión de requisitos.* [en línea] <www.researchgate.net/deen/32bfed640cd33b77.pdf> [Consulta: agosto de 2012].

2. BASS, Len. *Software architecture in practice.* 3a ed. Estados Unidos: Addison-Wesley Professional, 2003. 528 p.

3. DÍAZ RÍOS, Luis Javier. *Reutilización del software.* [en línea] <http://www.slideshare.net/pto0404/seminario-3-reutilizacin-del-software> [Consulta: agosto de 2012].

4. PEARSON, Craig Larman. *UML y patrones, introducción al análisis y diseño orientado a objetos.* Mexico: Prentice-Hall, 2001. 690 p.

5. PRESSMAN, Roger. *Ingeniería del software: un enfoque práctico.* 5a ed. Mexico: McGraw-Hill, 2002. 602 p.

6. ROSELO GARCÍA, Emilio. *Reutilización de COTS diseño de una solución para mejorar la reusabilidad de un entorno CAE.* [en línea] <http://dialnet.unirioja.es/servlet/tesis?codigo=22533> [Consulta: septiembre de 2013].

7. SOMMERVILLE, Ian. *Ingeniería del software.* 5a ed. España: Pearson Addison Wesley, 2005. 687 p.

APÉNDICES

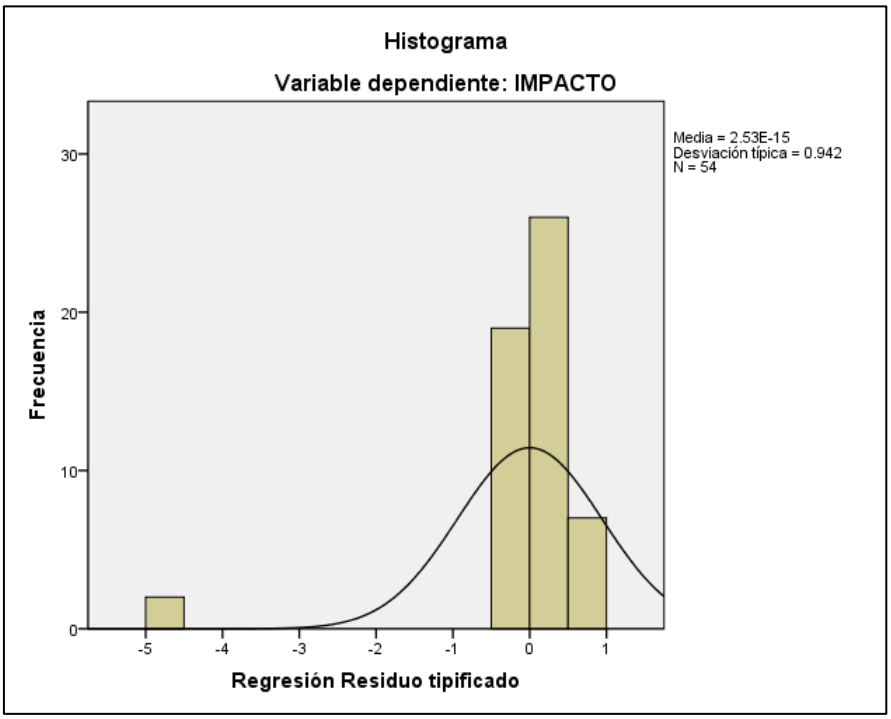
Para determinar el impacto en la calidad de un sistema software basado en componentes al utilizar componentes que posean un modelo de calidad consistente, se realizó la siguiente encuesta aplicada a una población relacionada con desarrollo de software y uso o mantenimiento de sistemas organizacionales.

La encuesta realizada se aloja temporalmente en:

https://docs.google.com/forms/d/1gmncUM0twpe1v7XMaJ2sRL-3vU9nr9JQacTdnIOiJtA/viewform

Los resultados de la encuesta se desarrollan en el capítulo 5, junto con pruebas de confianza y determinación de un modelo lineal que describe impacto mencionado.

A continuación se describen las pruebas de normalidad del modelo deducido.



Fuente: elaboración propia, con base al programa SPSS.