# Lineamientos y Estándares de Desarrollo de Software para el rol Desarrollador

# Lineamientos y estándares de desarrollo de software para el rol desarrollador

Este documento ha sido elaborado por los miembros del Consejo para las Tecnologías de Información y Comunicación del Estado Plurinacional de Bolivia (CTIC-EPB).

**Coordinación Secretaria Técnica del CTIC-EPB**: Juan Pablo Poma Chuquimia, Rodrigo Martinez Flores, Khantuta Muruchi Escobar e Ivan Rios Benitez.

**Grupo de Trabajo**: Pavel Sergio Mollinedo Grandi, Alexandra Aguilar Soria, Monica Vasquez Jaldin, Junior Manuel Muñoz Cuiza, Waldo Panozo Ramirez, Félix Nina Cruz, Nestor Poma Callizaya, Wilder Galarza Villarroel, Miguel Angel Medina Berdeja, Marco Antonio Hinojosa Fernandez, Juan Victor Serrudo Chavez, Wilson Campero Merlo, Johnny Henry Vasco Calle, Luis Alfredo Mamani Titirico, Juan Alberto Chacon Mercado, Felix Humberto Mercado Baspineiro, Wilson Charlie Seoane Sanchez, Luis Riveros Bothelo, Leonardo Yujra Mamani, Silverio Condo Huanca, Luis Roque Quispe, Daniel Dorado Cors, Ricardo Miguel De la Barra Bacarreza, Lionel Cardenas, Roland Catacora Huanca, Luis Alberto Quispe Pinto, Juan Carlos Paredes Mamani, Cristian Alvarado Paredes, Ramiro Ernesto Velasco Chambi, Victor Hugo Espinoza Balboa, Freddy Yussep Yuca Muñoz, Dilma Yugar Zegarra, Marco Antonio Franz Coca Barrientos, Maria Roxana Carvajal Aguirre, Pilar del Rosario Quenta Choque, Pablo Mendieta, Monica Torrez Apaza, Raquel Condori Mamani, Erwin Romeo Lucia Lazo, Juan Carlos Argote Orozco, Grover Velasquez Colque, Freddy Andres Veliz Miranda, Adrián Rojas Arévalo, Tito Andres Murillo Anibarro, Diego Eddy Rosa Mamani, Michael Castillo Ocampo, Nayra Nina Conde, Jose Marcial Flores Guerrero, Álvaro Zenon Quispe Segales, Wilson Mamani Condori, Jorge Ticona Colque, Vladimir Lopez Castro, Juan Marcelo Bayon Medina, Carlos Monroy Gutiérrez, Edwin Oscar Balderrama, Edwin Coronel Chicasaca, Juan Carlos Quispe Hinojosa, Marcelo Alvarado Laura, Ariel Espinoza Sanjinez, René Jorge Rocha Pérez, Nelson Luna Maidana, Edwin Coro Aricoma, Roberto Carlos Gorena Fernández, Juan Carlos Chipana Alvarado, Mauricio Alejandro Calla Marin, Jose Camilo Tapia Barrientos, Luis Solari Arias, Ronald Gerardo Chávez Gonzales, Rocio Mabel Alberto Patty, Leonardo Chuquimia Monzon, Fidel Angel Conde Lecoña, Veronica Chavez Espinoza, Víctor Hugo Argandoña Terán, Vladimir Callisaya Coaquira, Epifanio Inosente Ramirez, Abel Montoya Lopez, Edwin Yujra, Yuri Vladimir Laura Queteguari, Humberto Condarco Alejo, Alexeis Vladimir Carrillo Pinaya, Jhenrry Alvaro Mamani Javier, Julqui Israel Quispe Mamani, Susan Cinthia Donaire Apala, Mario Betancourt Lafuente, Ximena Gladys Alejandro Aquino, Roger Espinoza Vargas, Alfredo Callizaya Gutierrez, Jhonny Parra Mamani, Wilmer Rivero Nina, Joaquín Gonzalo Barrios Chacón, Delina Amdahid Delgado Alvarez, Marcelo Davis Ferrufino Tapia, Marcelo, Alan Valenzuela Andrade, Renato Apaza Tito, Ramiro Loza Herrera, Octavio Jesus Torrico Alvarez, Roxana Beatriz Cruz Mamani, Alvaro Diego Daza Alcaraz, David Samuel Canabire Garrado, Telassim Ginnola Gomez Jimenez, Edwin Carlos Chura Loza, Marcelo Rundo Davila, Ricardo Anyelo Flores Tilcara, Reynaldo Rodrigo Choque Vicente, Ever Luis Aruquipa Machaca, Christian Cuellar Ramírez, Carlos Max Tarqui Saenz, Primo Wenceslao Mamani Gutierrez, Isrrael Alvarez Rua, Armando Quispe Mamani, Angela Rocío Cáceres Coria, Franz Santiago Mamani Laruta, Ylsen Cardozo Cerezo, Luis Miguel Aguilar Cabezas, Benjo Huallpa Martinez, Diana Arlerh Muriel Soto, Alvaro Valentino Chavez Uzquiano, Juan Rosendo Llusco Catacora, Erick Rodrigo Romero Sanchez, Gladys Patty Quispe, Marcelo Tito Copa, David Gabriel Montaño Rocha, Daniel Saguez Tezanos Pinto, Erick Quispe Ajno, Victor Hugo Fuentes Nacho, Juan Marcelo Albis Ortiz, Mauricio Andres

Laurel Corrales, Policarpio Carata Flores, Daniel Chalco Barrera, Roger Navia Gutierrez, Leoncio Ivan Fuertes Delgadillo, Luis Alberto Fernandez Orellana, Armin Mesa Sanchez, Yamil Eduardo Reyna Kinlock, Luis Eduardo Rejas Alurralde, Rolando Aguilar Ninahuanca, Jhony Hernan Saavedra Gutierrez, Yajaira Peñarrieta Rodriguez, Juan Carlos Alvarez Villca, Erick Sierra Caballero, Alan Rodrigo Corini Guarachi, Alicia Heydy Estrada Cava, Jorge Antonio Nava Amador, Alejandro Salamanca, Marco Huanca Layme, Danitza Velka Vega Quispe, Luis Angel Pabon Cuentas, Edems Charles Vargas Quispe, Augusto Velasquez Pairumani, Ivan Marcelino Contreras Tamayo, Roman Franco Valero Calle, Luis Fernando Villca Valeros, Rosmery Jauregui Ticona, Elias Condori Mamani, Darío Vallejos Claros, Ivan Marcelo Chacolla, Yuli Richard Guaman Alvarez, Victor Hugo Gutierrez Carvajal, Leonardo Esquias Esquibel, Oscar Alex Villegas González, Hernan Rodrigo Miranda Dick, Erick Gomez, Remmy Alanoca, Alberto Encinas, Daniza Maritza López Mendoza, Oscar Parrado Ugarte. Victor Fuentes Nacho, Jose L Aguilar Mamani, Guiver Aymuro.

Diseño:

Diagramación:

Depósito Legal:

Impreso:



La Paz, Bolivia

2021

**CONTENIDO**

## 1. INTRODUCCIÓN

La adopción de buenas prácticas es fundamental para todas las etapas de desarrollo de un sistema o aplicación informática permitiendo la mitigación de problemas de software, ayudando al usuario final a tener un recurso eficiente, confiable, seguro y privado. Asimismo, se minimizan los riesgos y costos de mantención del sistema en horas hombre.

El presente documento pretende establecer lineamientos generales y recomendaciones específicas para la construcción de software en el Sector Público del Estado Plurinacional de Bolivia, que debe ser seguido por los equipos de desarrollo que trabajen en el Estado.

El documento cuenta con las siguientes secciones:

- **Lineamientos generales sobre desarrollo de software**. - Aborda las características del código fuente, gestión del repositorio de código y despliegue de aplicaciones. En ese sentido, se tratan temas como: guías de codificación, principios de diseño, *test*, versión de código fuente y automatización de la puesta en producción.
- **Lineamientos específicos al desarrollo según el ámbito de uso**. - Describe los lineamientos que deben seguir las aplicaciones, según dónde se utilizan. Existen tres ámbitos de uso: desarrollo web, desarrollo de aplicaciones de escritorio y para dispositivos móviles (celulares o tablets). Adicionalmente, se aborda el código heredado también llamado (*legacy)*, que por diferentes motivos no puede ser reemplazado, y que requiere el uso de buenas prácticas para su mantenimiento.

Sobre la aplicabilidad de los lineamientos, deben tenerse en cuenta las siguientes consideraciones:

- **Obligatorio,**- donde es imprescindible que se opere del modo señalado. Indicado por los términos “debe”, “requerido”, u “obligatorio”.
- **Recomendado,** donde es altamente aconsejable que se opere de dicho modo. Indicado por los términos “debería” o “recomendado”.
- **Opcional,** donde se puede optar por alternativas que consideren más convenientes. Indicado por los términos “opcional” o “puede”.
- **No permitido,**- donde no es recomendable operar de la forma señalada. Indicado por los términos “no debe” o “no permitido”.

## 2. ANTECEDENTES

El Artículo 2 del Decreto Supremo N° 2514 establece la creación de la Agencia de Gobierno Electrónico y Tecnologías de Información y Comunicación (AGETIC), como “una institución pública descentralizada de derecho público, con personalidad jurídica, autonomía de gestión administrativa, financiera, legal y técnica y patrimonio propio, bajo tuición del Ministerio de la Presidencia”.

EL Artículo 9 del Decreto Supremo N.º 2514 crea el Consejo para las Tecnologías de Información y Comunicación del Estado Plurinacional de Bolivia (CTIC-EPB), como instancia de coordinación técnica para la implementación de Gobierno Electrónico y el uso y desarrollo de Tecnologías de Información y Comunicación en el país.

El artículo 10 del Decreto Supremo N.º 2514 señala que la presidencia del CTIC-EPB estará a cargo del Director(a) General Ejecutivo(a) de la AGETIC, cuyas atribuciones serán:

- Convocar a sesiones ordinarias y extraordinarias del CTIC-EPB.
- Velar por el cumplimiento de la normativa aplicable en las resoluciones emanadas del plenario del CTIC-EPB.
- Organizar y coordinar grupos de trabajo entre los miembros del CTIC-EPB
- Aprobar y gestionar la normativa elaborada y/o recomendada por el plenario o grupos de trabajo del CTIC-EPB.
- Gestionar y administrar la página web del CTIC-EPB.
- Elaborar y aprobar el reglamento de funcionamiento interno del CTIC-EPB.
- Convocar a sesiones ordinarias y extraordinarias del CTIC-EPB.
- Velar por el cumplimiento de la normativa aplicable en las resoluciones emanadas del plenario del CTIC-EPB.
- Organizar y coordinar grupos de trabajo entre los miembros del CTIC-EPB
- Aprobar y gestionar la normativa elaborada y/o recomendada por el plenario o grupos de trabajo del CTIC-EPB.
- Gestionar y administrar la página web del CTIC-EPB.
- Elaborar y aprobar el reglamento de funcionamiento interno del CTIC-EPB.

El artículo 11 del Decreto Supremo N.º 2514 lista las funciones del CTIC-EPB:

- Formular propuestas de políticas y normativa relacionada con Gobierno Electrónico, a ser presentadas a la AGETIC.
- Presentar proyectos y programas de Gobierno Electrónico y Tecnologías de Información y Comunicación en el ámbito gubernamental a la AGETIC para su gestión.

- Generar mecanismos de participación de instituciones y organizaciones de la sociedad civil en la proposición y formulación de políticas y acciones relacionadas con Gobierno Electrónico y Tecnologías de Información y Comunicación en el ámbito gubernamental.
- Establecer espacios de coordinación entre las entidades del sector público para el desarrollo conjunto de programas, proyectos o acciones de Gobierno Electrónico y Tecnologías de Información y Comunicación en el ámbito gubernamental.
- Desarrollar y proponer estándares abiertos oficiales del Estado Plurinacional de Bolivia en materia de Gobierno Electrónico y Tecnologías de Información y Comunicación aplicables a las entidades del sector público.
- Establecer espacios de coordinación de comunidades de desarrollo informático, dentro del Estado, con la ciudadanía y a nivel internacional.

El presente documento fue parcialmente trabajado en la gestión 2019, retomando este año el trabajo del Ctic y concluyendo los lineamientos para desarrollo de software que se presentan en este documento, los lineamientos fueron construidos por los representantes de las siguientes entidades y de la sociedad civil

- Administración de Aeropuertos y Servicios Auxiliares a la Navegación Aérea
- Administración de Servicios Portuarios - Bolivia
- Administradora Boliviana de Carreteras
- Aduana Nacional
- Agencia Boliviana Espacial
- Agencia de Infraestructura en Salud y Equipamiento Médico
- Agencia Estatal de Medicamentos y Tecnologías en Salud
- Agencia Estatal de Vivienda
- Agencia Nacional de Hidrocarburos
- Autoridad de Fiscalización de Electricidad y Tecnología Nuclear
- Autoridad de Fiscalización Del Juego
- Autoridad de Fiscalización y Control de Cooperativas
- Autoridad de Fiscalización y Control de Pensiones Y Seguros
- Autoridad de Impugnación y Tributaria
- Autoridad de Regulación Y Fiscalización De Telecomunicaciones Y Transportes
- Autoridad de Supervisión del Sistema Financiero
- Banco Central de Bolivia
- Caja de Salud Cordes
- Caja de Salud De Caminos
- Caja Nacional de Salud
- Cámara de Diputados
- Cámara de Senadores
- Central de Abastecimiento y Suministros De Salud
- Comando en Jefe de las Fuerzas Armadas del Estado
- Comunidad de Software Libre
- Comunidad Software Libre Bolivia
- Consejo Nacional de Vivienda Policial
- Corporación de Las Fuerzas Armadas Para El Desarrollo Nacional

- Corporación del Seguro Social Militar
- Defensoría del Pueblo
- Depósitos Aduaneros Bolivianos
- Dirección General de Aeronáutica Civil
- Dirección General de Migración
- El Fondo Nacional de Desarrollo Forestal
- Empresa Boliviana de Alimentos y Derivados
- Empresa Estatal de Televisión "Bolivia Tv"
- Empresa Estatal de Transporte Por Cable Mi Teleférico,
- Empresa Publica Productiva Cartones de Bolivia
- Fondo de Desarrollo Indígena
- Fondo Nacional de Inversión Productiva y Social
- Fuerza Especial de Lucha Contra el Narcotráfico
- Fundación Cultural del Banco Central de Bolivia
- Gaceta Oficial de Bolivia
- Gestora Publica de la Seguridad Social de Largo Plazo
- Institución Publica Desconcentrada Soberanía Alimentaria
- Instituto Boliviano de Metrología
- Instituto Nacional de Laboratorios De Salud
- Instituto Nacional de Reforma Agraria
- Instituto Nacional de Salud Ocupacional
- Instituto Plurinacional De Estudio De Lenguas Y Culturas
- Lotería Nacional de Beneficencia y Salubridad
- Más y Mejor Internet Para Bolivia
- Ministerio de Defensa
- Ministerio de Economía y Finanzas Publicas
- Ministerio de Justicia y Transparencia Institucional
- Ministerio de La Presidencia
- Ministerio de Medio Ambiente y Agua
- Ministerio de Minería y Metalurgia
- Ministerio de Obras Públicas, Servicios y Vivienda
- Mutúal de Servicios al Policía
- Policía Boliviana
- Pro-Bolivia
- Procuraduría General de Estado
- Registro Único para la Administración Tributaria Municipal
- Servicio General de Identificación Personal
- Servicio Geológico Minero
- Servicio Nacional de Áreas Protegidas La Paz
- Servicio Nacional de Patrimonio Del Estado
- Servicio Nacional de Riego
- Servicio Nacional de Verificación de Exportaciones
- Servicio Nacional del Sistema De Reparto
- Servicio Plurinacional de Defensa Publica
- Servicio Plurinacional de La Mujer y de la Despatriarcalizacion "Ana Maria Romero"
- Transporte Aéreos Bolivianos
- Unibol Guaraní y Pueblos De Tierras Bajas Apiaguaiki Tüpa
- Unidad de Investigaciones Financieras

- Unidad de Proyectos Especiales
- Universidad Autónoma Tomás Frías
- Universidad Mayor de San Simón
- Vías Bolivia
- Viceministerio de Defensa Social y Sustancias Controladas
- Vicepresidencia Del Estado Plurinacional
- Yacimiento Petrolíferos Fiscales Bolivianos
- Yacimientos de Litio Bolivianos

Cabe mencionar que el desarrollo de esta mesa de trabajo se enmarca en el Reglamento de Funcionamiento del CTIC-EPB, aprobado mediante la Resolución Administrativa N° 024/2016 de la AGETIC, de fecha 31 de mayo de 2016.

## 3. OBJETIVOS.

### 3.1. Objetivo Principal:

Establecer lineamientos generales que deben ser tomados en cuenta en el momento de desarrollar software para el Estado por parte de los desarrolladores.

### 3.2 Objetivos Secundarios.

- Homogeneizar y normalizar los desarrollos de software.
- Mejorar la calidad y facilitar el mantenimiento del software.
- Servir de documento base para guías de codificación más específicas.
- Permitir implementar herramientas que puedan dar soporte al desarrollo del Software y verificar que se cumplan las reglas descritas en este documento.
- Desarrollar código fuente legible, seguro, eficiente, robusto y cohesionado.
- Mejorar la coordinación entre los equipos de trabajo evitando discusiones innecesarias.
- Facilitar los procesos de revisión de código y la creación de *scripts* para *testing*.

## 4. ALCANCE

Este documento es aplicable al rol de desarrollador en el interior de la administración pública y empresas proveedoras de software para el Estado.

## 5. LINEAMIENTOS GENERALES PARA EL DESARROLLO DE SOFTWARE.

### 5.1 Calidad de código fuente

Aunque una aplicación funcione perfectamente, el esfuerzo de mantenimiento o de atender el cambio de requerimientos puede ser más costoso que la reconstrucción

completa del software . Estos esfuerzos están directamente relacionados con varios factores, uno de ellos es la claridad y formato del código fuente. En esta sección están descritas las buenas prácticas con respecto a la calidad del código fuente.

#### 5.1.1 Nombres adecuados

Los nombres de variables*, funciones* y clases* deben ser entendibles (evitar el uso de acrónimos) y deben ser lo suficientemente descriptivos, de tal forma que revele el uso que se le quiere dar.

Se recomienda que el nombre de las funciones contenga un verbo de acción y el nombre de las variables contenga sustantivos y/o adjetivos.

No está permitido asignar valores directamente en el código fuente, estos deben obtenerse de un archivo de configuración, parámetros o constantes.

Es recomendable no usar variables globales*, sino que las mismas sean encapsuladas dentro de una estructura (objetos* o clases).

#### 5.1.2 Funciones y métodos

Una función debería tener máximo 3 valores de entrada, si son más, es recomendado encapsular estos parámetros dentro de un objeto o estructura de datos.

Es obligatorio que una función realice una única tarea y que generalmente no supere las 10-20 líneas de código.

#### 5.1.3 Manejo de errores

Los errores son condiciones críticas que no pueden ser manejadas por el código, y las excepciones son errores detectados durante la ejecución.

Es recomendable que las excepciones, si es que no pueden ser solucionadas, sean identificadas y gestionadas mediante mecanismos del lenguaje (por ejemplo *throw**, *try and catch**), y que se evite códigos de error internos y salidas del sistema.

Los mensajes de la excepción deben describir el contexto, ser específicos y fácilmente identificables.

### 5.1.4 Archivos de configuración

Las variables de configuración (nombre de la base de datos, rutas, etc.) deben centralizarse en archivos de configuración o variables de entorno* claramente identificados. La configuración debe permitir cambiar el entorno en el cual se ejecuta la aplicación (*development*, *test* y *production*).

### 5.1.5 Credenciales y secretos

Para el manejo de las credenciales o secretos, como ser contraseñas, tokens* de acceso y claves, en entornos de producción, es recomendable usar software especializado que permita:

- Centralizar las credenciales o secretos.
- Gestionar listas de acceso (ACL*) que permitan definir quienes tienen acceso a estos recursos.
- Generar dinámicamente contraseñas, tokens* y claves.
- Tener registros de auditoría, que indiquen quien tuvo acceso y en qué momento a las credenciales o secretos.

### 5.1.6 Creación de *logs*

Es obligatorio que el software genere un *log** de eventos, y este pueda responder a las siguientes preguntas:

- ¿Cuándo pasó? (*timestamp**).
- ¿Qué pasó? (códigos de error, niveles de impacto, etc.).
- ¿Dónde pasó? (por ejemplo, hostnames, gateway, módulos, etc.).
- ¿Quién estuvo involucrado? (por ejemplo, usernames, identificadores internos).
- ¿De dónde vino? (por ejemplo, el source IP*).

Los *logs* en entorno de producción deben registrar eventos importantes. En los entornos de *test* y de desarrollo es recomendable registrar eventos a detalle.

### 5.1.7 Comentarios

Los comentarios sólo deben ser usados en los siguientes casos:

- Documentación de API’s (JSDoc, javadoc, Pydoc, apiDoc, openAPI).
- Explicación de “POR QUÉ” se realizó cierta parte del código, si esta no queda clara en el código.

- Modificaciones específicas de ciertas librerías, frameworks o procedimientos (*hacks*).
- Comentarios tipo “TODO*” para indicar tareas que deban realizarse.
- Referencia (*link*) a un Código que es copiado de alguna fuente de Internet como ser *github*, *gitlab* o *stackoverflow.*

Los comentarios que no deben estar en el código fuente son:

- Marcadores posicionales.
- Código fuente comentado.
- *Comentarios de cambios realizados.*

### 5.1.8 *Tests*

Es recomendable que un sistema tenga *tests* unitarios* o de integración*.

Los *tests* unitarios deben cumplir con las siguientes características:

- Ejecución rápida
- Independientes
- Repetibles
- Mocks para dependencia externas.
- Evita el acceso a base de datos, peticiones por red, y acceso a sistema de archivos
- Son capaces de ejecutarse en paralelo
- Cubren al menos el 50% de las funciones codificadas

Considerar que los test unitarios y de integración, deben realizarse durante la fase de desarrollo, y no es una fase separada antes de entrar a producción.

### 5.1.9 Mecanismos de control

Para poder validar la calidad de código es recomendable utilizar:

- *Linters**.
- Analizador de código estático*.
- Ejecución de pruebas unitarias o de integración.
- Sesiones de “*code review*”*.
- Test de mutación y *fuzzy test**.

### 5.2 Calidad del repositorio de código

Un repositorio de código es útil para múltiples propósitos en el ciclo de vida de una aplicación. La función más importante que cumple es la de versionar el código, es decir, que este repositorio registra los cambios en los archivos a través del tiempo, esto permite sincronizar e integrar el código desarrollado por un equipo en un producto o componente software, pero también permite albergar toda la documentación relativa al desarrollo, o gestionar las distintas entregas y la publicación del código.

#### 5.2.1 Configuración Inicial

En el directorio raíz de cada proyecto deben existir los siguientes archivos:

- README.md Que contiene la descripción general del proyecto.
- INSTALL.md Describe los pasos para la instalación.
- UPDATE.md Describe los pasos para la actualización.
- LICENSE.md Licencia GPL Bolivia.
- CHANGELOG.md Para registrar los cambios entre versiones (opcional)
- .gitignore para no versionar archivos de configuración, compilados de la aplicación y binarios grandes.

No está permitido versionar claves, contraseñas, tokens u otra información sensible. En su lugar se puede poner datos de ejemplo.

No está permitido versionar las dependencias de código externos al software, debiéndose usar el gestor de paquetes. Tampoco se debe versionar el software compilado o empaquetado.

Es obligatorio que la estructura de base de datos (tablas, funciones y procedimientos almacenados) sea versionada mediante *scripts* de *migrations**.

#### 5.2.2 Gestión de proyectos comunitarios

En el documento *“Lineamientos para la organización del desarrollo, publicación, documentación y licenciamiento del software del Estado Plurinacional de Bolivia”*[9] se puede consultar los lineamientos acerca del manejo de repositorios de código de proyectos comunitarios.

#### 5.2.3 Mecanismos de control

Para poder validar la calidad del repositorio es recomendable utilizar:

- Herramientas de integración continua que verifiquen las reglas mínimas.
- *Scripts* de validación de estas reglas.

### 5.3 Calidad en el despliegue de aplicaciones

La automatización juega un papel fundamental a la hora de asegurar la calidad en los procesos de despliegue del software. Si el proceso de construcción, pruebas y despliegue no están automatizados, y existen problemas en la aplicación o en la configuración, no podremos reproducir las mismas. Es decir, al trabajar de forma manual, cada vez que se lleve a cabo el despliegue de una nueva versión, se realizará de forma distinta. No hay control en el proceso de d*espliegue* y, por lo tanto, no hay forma de asegurar la calidad final del producto. Es por eso que listamos las buenas prácticas para el despliegue de aplicaciones, con el objetivo de automatizar este proceso.

#### 5.3.1 Uso del repositorio para el despliegue

Es obligatorio que para la solicitud de despliegue se indique la versión que se actualizará, misma que estará en la rama *master* con el *tag* correspondiente; también se debe mantener la versión anterior en caso de *rollback**.

#### 5.3.2 Automatización

Es recomendable realizar la automatización a tres niveles:

- Integración continua, permite integrar los avances en desarrollo de forma frecuente.
- Entrega continua, permite enviar todos los cambios exitosos a un servidor de pruebas.
- Despliegue continuo, es colocar los cambios en ambientes de producción, a escala completa.

Es obligatorio que las aplicaciones tengan definido en el código los entornos de desarrollo, *test* y producción. Estos deben permanecer separados para reducir riesgos de acceso o cambios no autorizados.

#### 5.3.3 Escalamiento y alta disponibilidad

Se debe considerar escalamiento vertical y horizontal.

- Escalamiento vertical, adicionando más recursos (RAM, procesador y velocidad del disco).

- Escalamiento horizontal, añadiendo más instancias de la aplicación ya sea en máquinas virtuales o contenedores.
- Tomar en cuenta la existencia de software que está mejor preparado para escalamiento vertical* que para escalamiento horizontal* y viceversa.

Las herramientas que pueden usarse para la gestión de escalamiento horizontal son *reverse proxy, caching server, forward proxy, load balancer, message broquers o firewalls*.

Para identificar los puntos críticos de la aplicación es recomendable seguir con los procesos que indica el documento *Foundation Level Performance Testing* de la ISTQB[3] (load testing, stress testing, endurance testing, spike testing, volume testing, scalability testing).

### 5.3.4 Bases de datos

En el despliegue de bases de datos es obligatorio tener en cuenta las gestión de *Backups.* Los backups pueden ser de 4 tipos según se requiera:

- *Backup* completo
- *Backup* incremental
- *Backup* diferencial
- *Backup* del log de transacciones

La “estrategia de *backup*” se realiza según la cantidad de registros almacenados por unidad de tiempo, espacio para los *backups y* el tipo de *backup* requerido.

Es recomendable tener un entorno de integración continua que verifique la integridad y confiabilidad de los diferentes *backups*.

### 5.3.5 Mecanismos de control

Para poder validar la calidad de despliegue se puede utilizar los siguientes procesos:

- Calcular los tiempos necesarios para realizar un despliegue en los diferentes entornos. Menos tiempo significa que se tienen los procesos más automatizados.
- La frecuencia de generación de *backups**.
- Despliegue automático de la aplicación mediante el uso de migraciones.
- Revisar los tipos de *tests* mencionados en el Documento del ISTQB. [2]

## 6. LINEAMIENTOS ESPECÍFICOS PARA EL DESARROLLO SEGÚN EL ÁMBITO DE USO.

### 6.1 Calidad en desarrollo web

Las aplicaciones web tienen una serie de ventajas que han permitido ser el tipo de software que más se usa. Una de las ventajas más importantes es que no dependen de la computadora ni del sistema operativo en el que se ejecuta, y que puede ser utilizadas en cualquier parte del planeta a cualquier hora y desde diferentes dispositivos. Entre los aspectos negativos se encuentra que no es posible usar todas las funciones que pueden usarse en aplicaciones nativas. En tal sentido, listamos las mejores prácticas.

#### 6.1.1 HTML

Es obligado el uso de los *tags* de html5, dejando de lado los *tags* de estándares antiguos como ser el xhtml, así mismo, definir el *encoding** como utf8 en las cabeceras.

Es obligatorio el uso de "*tags* semánticos"* y metadatos, de tal forma que mejore la indexación y ordene la estructura de la página.

Es recomendable que no exista código javascript y css en el html, y que los archivos css sean llamados antes que los de javascript.

Es obligatorio que los recursos (imágenes, archivos css y archivos javascript) sean optimizados para la web mediante procesos de minificación* o uglificación*.

#### 6.1.2 CSS

Es recomendable usar una convención de nombres para elementos css o usar una ya existente como BEM*, OOCSS* y SMACSS*.

Es recomendable usar preprocesadores de css que extienden sus funcionalidades, por ejemplo: sass*, less* y stylus*.

Es recomendable que exista un único archivo css donde se puedan configurar los colores primarios, fuentes e iconos de la aplicación.

#### 6.1.3 JAVASCRIPT

Es recomendado que como mínimo se use javascript basado en el estándar ecma6 durante el desarrollo y en producción. Se requiere convertirlo a ecma5 para la compatibilidad con navegadores antiguos.

### 6.1.4 Seguridad

Es obligatorio tomar en cuenta las recomendaciones de la OWASP*, reflejadas en el documento OWASP TOP 10. [5]

### 6.1.5 Accesibilidad

Es recomendable que las aplicaciones web sigan las recomendaciones WCAG v2.1 nivel A [1].

### 6.1.6 Mecanismos de control

Para poder validar la calidad de software web se pueden utilizar las siguientes herramientas:

- Analizadores de código estático
- Herramientas de análisis Web (Rendimiento, seguridad y validación de html)
- Herramientas que verifiquen la compatibilidad de navegadores
- Actividades de "*code review*"*
- Revisiones QA y Seguridad

## 6.2 Calidad en desarrollo móvil

Una aplicación móvil es una aplicación desarrollada con herramientas específicas, para que éstas se ejecuten en el sistema operativo de dispositivos como celulares y tablets. Estas aplicaciones pueden acceder a funcionalidades de los Sistemas Operativos del equipo móvil para facilitar el uso de dispositivos como brújula, cámara, correo, GPS, etc. Estas aplicaciones se instalan en el dispositivo y normalmente hacen un uso óptimo de la funcionalidad del móvil. En esta sección se listarán las mejores prácticas para aplicaciones móviles considerando la diversidad de dispositivos móviles existentes.

### 6.2.1 Compatibilidad

Es obligatorio tener identificadas las versiones de los sistemas operativos soportados y los tamaños de pantalla (vertical u horizontal) donde la aplicación móvil se ejecutará.

### 6.2.2 Publicación

Es obligatorio que la aplicación sea preparada para su publicación en las tiendas de aplicaciones oficiales para cada sistema operativo*.

Es recomendable que se tenga un flujo para capturar información y notificar en caso de que la aplicación se detenga.

Está prohibido colocar vídeos como contenido estático que haga que los instaladores sean más pesados.

Solo se debe pedir al usuario los permisos mínimos y estrictamente necesarios.

### 6.2.3 Rendimiento

Es recomendable medir el uso de memoria, CPU y batería de la aplicación, además de la temperatura del dispositivo y tener estas estadísticas disponibles.

Es recomendable realizar procesos de revisión de la calidad indicados en la *Foundation Level Mobile Application Testing* de la ISTQB [4]

### 6.2.4 Seguridad

Es obligatorio tomar en cuenta las recomendaciones de la OWASP*, reflejadas en el documento OWASP TOP 10 mobile. [6]

Es obligatorio utilizar mecanismos de seguridad [4] para el almacenamiento y manejo de datos sensibles, como ser contraseñas o tokens.

La comunicación con *web services* debe ser encriptada usando un certificado SSL/TLS.

### 6.2.5 Accesibilidad

Es recomendable que las aplicaciones web sigan las recomendaciones WCAG v2.1 nivel A [1].

### 6.2.6 Mecanismos de control

Para poder validar la calidad de código se pueden utilizar las siguientes herramientas:

- Analizador de código estático.
- Revisiones QA y Seguridad.

## 6.3 Calidad en desarrollo de escritorio

Estas aplicaciones están totalmente vinculadas al sistema operativo de un computador, solo se puede trabajar con ellas en un equipo que tenga instalada la misma, por lo que si no existe acceso a él, dejamos de tener acceso a la aplicación. Este tipo de aplicaciones generalmente se desarrollan cuando las mismas deben funcionar en entornos de desconexión, uso de hardware especializado o uso intensivo de recursos como ser procesador, RAM o disco de almacenamiento. Enumeramos las mejores prácticas para este tipo de aplicaciones.

### 6.3.1 Compatibilidad

Es obligatorio que la aplicación funcione en al menos un sistema operativo libre y con diferentes resoluciones de pantalla.

Las aplicaciones de escritorio deben tener un mecanismo de actualización y notificación de nuevas versiones.

### 6.3.2 Publicación

Es recomendable que las aplicaciones se publiquen usando los sistemas de empaquetado propios de cada sistema operativo.

### 6.3.3 Gestión de errores

Se deben gestionar los errores mediante logs y es recomendable que se tengan mecanismos tipo “Reportar un problema” en caso de que la aplicación capture errores o la aplicación se detenga.

### 6.3.4 Mecanismos de control

Para poder validar la calidad de la aplicación de escritorio se pueden utilizar las siguientes herramientas:

- Analizador de código estático
- Revisiones QA

## 6.4 Calidad en sistemas heredados (LEGACY)

Un sistema heredado (o sistema *legacy*) es una aplicación de software, anticuada que recibe actualizaciones periódicamente, y que en muchos casos resultan tareas críticas. Los riesgos operacionales de una actualización o cambio a un nuevo software pueden ser muy altos, especialmente si hay riesgo de que los datos vitales se pierdan o corrompan. En estos casos se deben seguir lineamientos que permitan su mantenimiento en el tiempo.

### 6.4.1 Pasos Iniciales

- Es obligatorio que el código *legacy* se versione si es que aún no lo está.
- En el caso de código *legacy* es aceptable colocar dependencias en el repositorio.

- Es obligatorio seguir con el punto **5.2.1** sobre calidad del repositorio del presente documento.

### 6.4.2 Modificaciones en sistemas heredados

Es recomendable evaluar el objetivo al que se quiere llegar:

Si el objetivo es reducir la cantidad de errores, reducir la deuda técnica o cubrir fallos de seguridad del software se debería tomar en cuenta:

- Realizar *tests* unitarios antes de la “refactorización”.
- Seguir las recomendaciones del capítulo de calidad de código.
- Usar patrones de refactorización [8]

Si el objetivo es desarrollar nuevas funcionalidades o modificar existentes se debería tomar en cuenta:

- Realizar *test* de integración, en caso de que el cambio sea complejo.
- Ir mejorando la calidad de código existente a medida que se realicen modificaciones en el mismo.

Para ambos casos es recomendable:

- Comenzar con cambios que puedan realizarse mediante el IDE* de forma automática. Por ejemplo, cambiar nombres de variables y funciones o usar estilo de codificación.
- Preparar la aplicación para que tenga un flujo de integración continua.

### 6.4.3. Mecanismos de control.

Para poder validar la calidad de código se pueden utilizar las siguientes herramientas:

- *Linters**
- Analizador de código estático
- Ejecución de pruebas unitarias
- Sesiones de “*code review*”*
- *Test* de mutación

**GLOSARIO** [7]

**ACL** (Lista de control de acceso): Lista los derechos de acceso de los sujetos al objeto requerido de acuerdo con un modelo de seguridad. Opcionalmente, el proceso puede registrar en una pista de auditoría cualquier intento de acceso ilegal.

**Analizador de código estático:** Es la herramienta que lleva a cabo el análisis estático de código. La herramienta comprueba el código fuente para determinadas propiedades tales como la conformidad con estándares de codificación, métricas de calidad o anomalías en el flujo de datos.

**ApiDoc:** Es una sintaxis para agregar documentación de la API al código fuente de Api Rest.

**Backup:** (Copia de seguridad) Un recurso que es, o puede usarse, como un sustituto cuando falla un recurso principal o cuando un archivo se ha dañado. La palabra también se usa como verbo, respaldar, es decir, hacer una copia anticipando fallas o daños futuros. Así, un volcado forma una copia de seguridad que se utilizará en los casos en que el archivo de un usuario se haya vuelto inservible; la realización del volcado se puede considerar como una copia de seguridad de la versión en el disco.

**BEM** es un sistema de nomenclatura de clases ideado por el equipo de desarrollo de Google Rusia. La idea que hay detrás de BEM es diferenciar clases CSS que cumplen diferentes funciones. Esto se hace nombrando las clases CSS de una manera especial, de tal forma que su nombre describe su función

**Certificado** (certificado digital, certificado de clave pública), Para un medio de autenticación de claves públicas un certificado es un archivo emitido por un tercero de confianza, una autoridad de certificación.El certificado contiene tanto una clave pública como los detalles de la persona u organización a la que pertenece, que el tercero declara ser correcto.

**Clase:** La clase proporciona una forma de tipo de datos abstracto. También es la base del concepto de objeto que subyace en lenguajes orientados a objetos.

**Code Review:** También conocido como Peer Code Review, es el acto de reunirse consciente y sistemáticamente con los compañeros programadores para verificar el código de los demás en busca de errores y se ha demostrado repetidamente que acelera y agiliza el proceso de desarrollo de software como pocas otras prácticas pueden hacerlo.

**Código fuente:** La forma de un programa que se ingresa a un compilador o transpiler para conversión en código objeto equivalente.

**ECMA:** EcmaScript es el lenguaje de scripting estandarizado por Ecma International en la especificación ECMA-262 e ISO / IEC 16262. El lenguaje se usa ampliamente para scripting del lado del cliente en la web, en forma de varias implementaciones conocidas como JavaScript, JScript y ActionScript.

**Encoding:** La representación de símbolos en algún alfabeto por símbolos o cadenas de símbolos en algún otro alfabeto.

**Entorno:** Es un sistema de software que brinda soporte para desarrollo, reparación y mejora de software, y para la gestión y control de estas actividades. Un sistema típico contiene una base de datos central y un conjunto de herramientas de software. La base de datos central actúa como un depósito de toda la información relacionada con un proyecto a lo largo de la vida útil de ese proyecto. Las herramientas de software ofrecen soporte para las diversas actividades, tanto técnicas como de gestión, que se deben realizar en el proyecto.
Los diferentes entornos varían en la naturaleza general de sus bases de datos y en la cobertura que brinda el conjunto de herramientas. En particular, algunos fomentan (o incluso hacen cumplir) una metodología de ingeniería de software específica, mientras que otros brindan solo apoyo general y, por lo tanto, permiten la adopción de una variedad de metodologías. Sin embargo, todos los entornos reflejan la preocupación por todo el ciclo de vida del software.

**Función**: Una unidad de programa donde: dados valores para los parámetros de entrada calcula un valor. Ejemplos incluyen las funciones estándar como sin (x), cos (x), exp (x); además, la mayoría de los lenguajes permiten funciones definidas por el usuario. Una función es una "caja negra" que se puede utilizar sin ningún conocimiento o comprensión de los detalles de su funcionamiento interno. En algunos lenguajes de programación, una función puede tener efectos secundarios.

**Fuzzing o fuzz testing:** Es una técnica de prueba de software automatizada que implica proporcionar datos no válidos, inesperados o aleatorios como entradas a un programa de computadora. Luego, el programa se monitorea en busca de excepciones como fallas, afirmaciones de código incorporadas fallidas o posibles fugas de memoria.

**Gestor de paquetes:** Es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.

**Hack:** Una solución alternativa eludiendo un problema o una limitación reconocidos en un sistema o política. Suele ser una solución temporal que implica que se necesita una solución genuina al problema. Pero las soluciones provisionales son con frecuencia tan creativas como las verdaderas soluciones, e implican un pensamiento innovador en su creación.

**Hostnames** El nombre del host o hostname es un nombre único que se asigna a un ordenador o dispositivo (impresora, router, etc.) para poder identificarlo dentro de una red, entre los distintos dispositivos conectados a la misma. En una red local, el nombre del ordenador que se asigna desde el sistema operativo es el que se utiliza para su identificación. Los nombres de host pueden utilizar caracteres ASCII hasta un máximo de 255

**html 5** ( hypertext markup language version 5): Una forma de SGML destinada a su uso en la World Wide Web. Una característica particular de HTML es el uso de etiquetas que incluyen información sobre hipervínculos, ya sea a otros lugares del documento u otros documentos en la Web. El texto rodeado por estas etiquetas suele estar subrayado y en un color diferente cuando se muestra en el navegador, y se

puede hacer clic para saltar al enlace. HTML tiene un conjunto de etiquetas básicas, que en su mayoría definen formato en lugar de contenido, aunque algunos navegadores usan etiquetas no estándar. La versión actual es HTML5 y se está trabajando en HTML6.

**IDE** (Interactive development environment) Entorno de desarrollo interactivo Un conjunto de programas para el desarrollo de programas o aplicaciones con un usuario común que interactúa, a menudo usa una interfaz gráfica de usuario e incluye herramientas de software para escribir y editar código, compilar, ejecutar y depurar con una manera fácil y consistente de moverse entre las diversas funciones.

**Javadoc:** Es una sintaxis para agregar documentación de la API al código fuente de Java.

**JSDoc:** Es una sintaxis para agregar documentación de la API al código fuente de JavaScript**.**

**Linter:** Linting es la verificación automática de código fuente en busca de errores programáticos y estilísticos. Esto se hace usando una herramienta de linting (también conocida como linter). Un linter es un analizador de código estático básico.

**Log:** Registro generado por procedimiento que implica registrar todos los datos e interacciones que pasan por un punto particular de un sistema. El punto elegido suele ser parte de un bucle de comunicación o una ruta de datos hacia o desde un dispositivo como un teclado y una pantalla en la que los datos son transitorios. Si ocurre una falla del sistema o un resultado inesperado, es posible reconstruir la situación que existía. Por lo general, estos registros no se archivan y se pueden sobrescribir una vez que se haya completado el trabajo asociado.

**Migrations**: También conocidas como schemes migrations, migraciones de esquemas de bases de datos o simplemente migraciones, son conjuntos controlados de cambios desarrollados para modificar la estructura de los objetos dentro de una base de datos relacional. Las migraciones ayudan a hacer la transición de los esquemas de la base de datos de su estado actual a un nuevo estado deseado, ya sea que eso implique agregar tablas y columnas, eliminar elementos, dividir campos o cambiar tipos y restricciones. Las migraciones gestionan cambios incrementales, a menudo reversibles, en las estructuras de datos de forma programática. Los objetivos del software de migración de bases de datos son hacer que los cambios en la base de datos sean repetibles, compartibles y comprobables sin pérdida de datos. Generalmente, el software de migración produce artefactos que describen el conjunto exacto de operaciones necesarias para transformar una base de datos de un estado conocido al nuevo. Estos se pueden registrar y administrar mediante un software de control de versiones normal para realizar un seguimiento de los cambios y compartirlos entre los miembros del equipo.

**Minificación:** Elimina los espacios en blanco innecesarios y los comentarios y puntos y comas redundantes en el código fuente. Se puede revertir cuando sea necesario.

**Módulo:** Es una construcción de programación o especificación que define un componente de software. A menudo, un módulo es una unidad de software que

proporciona a los usuarios algunos tipos de datos y operaciones sobre esos tipos de datos, y puede en algunos casos compilarse por separado.

**Niveles de prioridad:** Los niveles actúan a modo de filtro en un listado de logs. Normalmente podremos configurar el nivel mínimo que deseamos para mostrar estos mensajes a través de un archivo de configuración o un flag del compilador. Normalmente los niveles son los siguientes, en un orden de importancia descendente: Fatal, Error, Warning, Info, Debug, Trace

**Objeto:** Un objeto es una instancia de un componente que comprende estructuras de datos y procedimientos (llamados métodos) para manipular las estructuras. Estos métodos se activan mediante mensajes enviados al objeto, y la estructura interior del objeto está completamente oculta a cualquier otro objeto (un propiedad llamada encapsulación). Los objetos se derivan de una plantilla y la colección de los objetos que son instancias de una plantilla particular se dice que forman una clase.

**OOCSS** (Object-Oriented CSS) Fue desarrollada por Nicole Sullivan en 2008 y se basa en dos principios básicos: Separar la estructura del diseño (en inglés lo describen como skin, piel). Separar el contenedor del contenido.

**OpenAPI:** Es una sintaxis para agregar documentación de la API al código fuente de Api Rest.

**Pydoc:** Es una sintaxis para agregar documentación de la API al código fuente de Python.

**Rollback:** Un proceso que inicia un proceso o programa en ejecución a un punto de control. La palabra también se usa como verbo para reiniciar en el punto de control.

**Sass, less, stylus:** Son preprocesadores de código css usados para extender la funcionalidad básica de css a través de su propio lenguaje de script. Ayuda a simplificar la sintaxis para complejos elemento como variables funciones mixins y herencia

**SGML:** Un metalenguaje bajo estándar internacional (ISO 8859) que se utiliza para definir la sintaxis de los lenguajes de marcado textual. Esto permite que tanto el remitente como el receptor del texto identifiquen su estructura (por ejemplo, título, autor, encabezado, párrafo, etc.)

**SMACSS** (Scalable and Modular Architecture for CSS): Desarrollado en 2011 por Jonathan Snook, SMACSS funciona mediante organización de las reglas CSS en 5 categorías. (Base, Maquetación, Módulo, Estado y tema)

**Source IP**: Es la dirección IP de origen que normalmente es la dirección desde la que se envió el paquete, pero la dirección del remitente en el encabezado puede modificarse.

**Timestamp:** La hora y fecha de una operación o evento cuando se agrega automáticamente a una visualización de pantalla, archivo de registro o archivo de salida de un procedimiento informático. Es una ayuda valiosa para rastrear errores y

puede usarse como parte de un proceso de auditoría. La hora y la fecha se derivan de los datos internos de la computadora.

**TODO:** Es un tag informal en los comentarios que los programadores usan para ayudar a indexar problemas comunes, este tag significa que algo debe ser realizado

**Token** (token de seguridad): Un dispositivo que, empleado junto con algo que el usuario conoce (por ejemplo, una contraseña), permitirá el acceso autorizado a un sistema informático o red. El dispositivo puede ser físico, en cuyo caso es un hard token (por ejemplo, una tarjeta inteligente); o puede ser lógico, en cuyo caso es un soft token.

**Try and Catch:** Son palabras clave que representan el manejo de excepciones debido a errores de codificación o datos durante la ejecución del programa. Un bloque Try es el bloque de código en el que ocurren excepciones. Un bloque de captura Catch detecta y maneja excepciones

**Test de mutación:** La herramienta que testea los test unitarios son los test de mutaciones. Su funcionamiento es relativamente sencillo: esta herramienta genera pequeños cambios en el ćodigo, conocidos como mutantes, y a continuación pasa los test unitarios. Si los test unitarios fallan, es que han sido capaces de detectar ese cambio de código, y el mutante es eliminado. Si, por el contrario, los test unitarios pasan, el mutante sobrevive y la fiabilidad del test queda en entredicho.

**Tests unitarios** Las pruebas unitarias consisten en verificar el comportamiento de las unidades más pequeñas de una aplicación. Técnicamente, eso sería una clase o incluso un método de clase en los lenguajes orientados a objetos, y un procedimiento o función en los lenguajes procedimentales y funcionales.

**Uglificación:** Transforma el código en una forma "ilegible" cambiando los nombres de las variables, los nombres de las funciones, etc., para ocultar el contenido original. Una vez que se usa, no hay forma de revertirlo.

**Username:** Cada cuenta incluye un nombre de usuario o username y una contraseña, al cual se le asigna niveles de acceso de seguridad,. El administrador del sistema es responsable de configurar y supervisar las cuentas de usuario.

**Utf8** (Formato de transformación Unicode de 8 bits): Un método para codificar puntos de código Unicode utilizando enteros sin signo de un byte; de uno a cuatro de estos números enteros se utilizan según el punto de código. UTF-8 es el esquema de codificación Unicode más utilizado y es la codificación predeterminada para documentos XML.

**Variable:** Una unidad de almacenamiento que se puede modificar durante la ejecución del programa, generalmente por operaciones de asignación o lectura. Una variable generalmente se denota con un identificador o con un nombre.

**Variables de entorno:** Es una variable dinámica que puede afectar al comportamiento de los procesos en ejecución en un ordenador.

**Variables globales**: Las variables globales son accesibles desde todas las partes de un programa. Por el contrario, las variables locales son accesibles solo en el programa módulo dentro del cual se definen.

**Xhtml**: Una reformulación de HTML como una aplicación de XML en lugar de SGML. La versión 1 era compatible con la versión 4 de HTML, pero las siguientes versiones no lo fueron.

## BIBLIOGRAFÍA

[1] World Wide Web Consortium. (2008). Web content accessibility guidelines (WCAG) 2.1. [Online]. Disponible: https://www.w3.org/TR/WCAG21/
[2] ISTQB: Foundation Level Syllabus, (2018) v3.1. [Online], Disponible https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html
[3] ISTQB: ISTQB CTFL-PT Syllabus (2018) GA performance [Online], Disponible https://www.istqb.org/downloads/send/59-performance-testing/239-istqb-ctfl-pt-syllabus-2018-ga.html
[4] ISTQB: Mobile Application Testing Specialist Syllabus (2018) [Online] Disponible: https://www.istqb.org/downloads/send/61-mobile-application-testing/251-mobile-application-testing-specialist-syllabus.html
[5] OWASP, F. (2020). OWASP Top Ten. [Online] Disponible: https://owasp.org/www-project-top-ten/
[6] OWASP Mobile Security Project: Top 10 Mobile Risks," 2014. [Online]. Disponible: https://www.owasp.org/index.php/OWASP_Mobile_Security_ Project#tab=Top_10_Mobile_Risks.
[7] Butterfield, A., Ngondi, G. E., & Kerr, A. (Eds.). (2016). *A dictionary of computer science*. Oxford University Press.
[8] Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

[9] AGETIC, Lineamientos para la organización del desarrollo, publicación, documentación y licenciamiento del software del Estado Plurinacional de Bolivia (2017)