**DIS-017 Modelo de Operación de Verificación Post-Producción**

**Anexo Reporte de Evaluación de Solución Tecnológica – Intranet COSDA**

**14/dic/2018**

**Contenido**

[1. Objetivo 3](#_Toc528580693)

[2. Introducción 3](#_Toc528580694)

[3. Alcance 4](#_Toc528580695)

[4. Detalle del Entregable 5](#_Toc528580696)

[4.1. Contexto de la Solución Tecnológica 5](#_Toc528580697)

[4.2. Tipo de Solicitud 5](#_Toc528580698)

[4.3. Evaluación de Código Fuente 5](#_Toc528580699)

[4.4. Alineación Entre Diseño y Construcción de Soluciones Tecnológicas 11](#_Toc528580700)

[4.5. Consistencia entre Ambientes del Ciclo de Vida de Desarrollo 11](#_Toc528580701)

[4.6. Cumplimiento Documental para Liberación en Ambiente Productivo 12](#_Toc528580702)

[4.7. Resultado Integral 13](#_Toc528580703)

[4.8. Propuestas de Acciones de Mejora 18](#_Toc528580704)

[5. Conclusiones 19](#_Toc528580705)

1. Objetivo

El entregable tiene por objetivo especificar las brechas y riesgos identificados derivado de la evaluación realizada sobre la solución tecnológica en los siguientes ámbitos:

**Calidad de código. -** Registrar el resultado del análisis de calidad del código fuente de una solución tecnológica específica con base en diversos estándares aceptados en la industria. La evaluación permite identificar el uso de malas prácticas, brechas de seguridad, *Bugs* que pueden provocar incidencias en la operación, deuda técnica excesiva y oportunidades de mejora en la codificación.

Una vez obtenida la información es posible generar propuestas de mejora que permitan elevar la calidad del código, reduciendo la posibilidad de afectación a la operación del Instituto y facilitando su mantenimiento.

**Verificar alineación entre diseño y construcción de la solución. -** Verificar que exista alineación entre los elementos considerados en la etapa de diseño detallado de la solución tecnológica (Capas de desarrollo, Paquetes, Clases, Métodos, Distribución de Componentes, Tecnologías) y su implementación durante la etapa construcción.

**Validar consistencia entre ambientes del ciclo de vida de desarrollo. -** Validar la consistencia entre los ambientes de desarrollo, pruebas y producción en términos de tecnologías (versiones, *releases* y parches) utilizadas.

**Documentación para liberación en ambiente productivo. -** Verificar que el proyecto cumple con la documentación necesaria para realizar la liberación de la solución tecnológica en ambiente productivo.

1. Introducción

Actualmente existen diversas problemáticas que dificultan la mejora continua de las soluciones tecnológicas desarrolladas dentro del Instituto, las cuales se mencionan a continuación:

* No se cuenta con un repositorio central de código fuente.
* Dificultad en la gestión de cambios realizados al código fuente durante el ciclo de vida de desarrollo.
* Falta de definición de estándares y mejores prácticas que regulen la construcción de soluciones.
* No se realiza la validación de la documentación necesaria para liberar en producción.
* Existen inconsistencias entre el diseño y la construcción de soluciones tecnológicas.
* Se presentan inconsistencias entre los ambientes utilizados durante el ciclo de vida de desarrollo.
* No se evalúa la calidad del código fuente generada.
* El código puesto en producción no cumple con estándares.

El equipo de verificación post-producción es responsable de cubrir las áreas de oportunidad antes mencionadas y proveer al Instituto la siguiente propuesta de valor:



Ilustración Propuesta de Valor

1. Alcance

La evaluación de soluciones tecnológicas considera los siguientes aspectos:

**Calidad de Código**: La evaluación de calidad de código se realiza con apoyo de la herramienta *SonarQube* ya que permite ejecutar una inspección estática sobre la codificación de la solución con base en los siguientes estándares*: Common Weakness Enumeration (CWE), Escal Institute of Advanced Technologies (SANS), Open Web Application Security Project (OWASP), The Motor Industry Software Reliability Association (MISRA) y Code Signing Certifcates CERTM*. La inspección permite identificar diversos errores alojados en el código que aumentan el riesgo de presentar un error en la operación.

**Alineación Entre Diseño y Construcción de Soluciones Tecnológicas:** Verificar el nivel de alineación que existe entre el diseño de soluciones tecnológicas y su construcción. De esta forma se puede identificar discrepancias entre la propuesta inicial de diseño y las decisiones consideradas en la etapa de construcción.

**Consistencia entre Ambientes del Ciclo de Vida de Desarrollo:** Validar que existe consistencia en la configuración de los ambientes (Versionamiento, *releases* y parches) utilizados a lo largo del ciclo de vida de desarrollo (Desarrollo, pruebas y producción). La validación se enfoca en los productos de *Software*, versionamiento y parches instalados en cada ambiente.

**Cumplimiento Documental para Liberación en Ambiente Productivo:** Verificar que el proyecto cumple con la documentación necesaria para realizar la liberación de la solución tecnológica en ambiente productivo.

1. Detalle del Entregable
   1. Contexto de la Solución Tecnológica

El **Sistema de Intranet COSDA** tiene por objetivo proporcionar información sobre las publicaciones internas del instituto, así como el vínculo a varios aplicativos institucionales y que sirve de base para que los asesores proporcionen información a nuestros usuarios finales.

* 1. Tipo de Solicitud

| **Tipo de Solicitud** | Descripción | **Tipo** |
| --- | --- | --- |
| Nuevo | Construcción de una solución tecnológica nueva. |  |
| Mantenimiento Correctivo | Modificación reactiva de una solución tecnológica derivada del descubrimiento de incidencias durante la operación.  Ej. Incidencias reportadas por usuarios. |  |
| Mantenimiento Adaptivo | Modificación realizada para adaptar la solución tecnológica a su entorno.  Ej. Derivado de la actualización de versión de base de datos. |  |
| Mantenimiento Perfectivo | Modificación realizada con el objetivo de agregar funcionalidad nueva, mejorar su desempeño y/o facilitar su mantenimiento.  Ej. Requerimientos que agreguen funcionalidad. |  |
| Mantenimiento Preventivo | Modificación realizada para evitar fallas en la operación.  Ej. Optimización de Código. |  |
| Identificación de Código Fuente | Identificación de código fuente dentro del Instituto. | X |

* **Descripción de la Solicitud**

Se identificó el código fuente del proyecto como parte del esfuerzo realizado por la gerencia de Ingeniería de TI enfocado en centralizar los códigos fuente del INFONAVIT por medio de su almacenamiento en el repositorio institucional.

* 1. Evaluación de Código Fuente

La inspección permite identificar diversas incidencias alojados en el código los cuales aumentan el riesgo de presentar una falla en la operación. La herramienta de evaluación de calidad de código *SonarQube* agrupa las incidencias identificadas por medio de categorías que a su vez se clasifican por su severidad.

* **Categoría**
  + ***Bugs***

Los *Bugs* son errores en el código fuente de un sistema o aplicación que deriva en un comportamiento incorrecto/inesperado generando un resultado indeseado. Es importante identificar y solucionar de manera temprana los *Bugs* de programación para asegurar la calidad en los productos desarrollados, así como su funcionamiento de acuerdo a lo esperado.

* + **Vulnerabilidades de Seguridad**

Las Vulnerabilidades de Seguridad son imperfecciones alojadas en el código fuente las cuales son susceptibles a ser aprovechadas por una atacante con el fin ejecutar código malicioso y/o acceder a información sensible manipulada por una solución tecnológica.

Por lo anterior, es importante identificar de manera temprana las vulnerabilidades de seguridad para así fortalecer el código fuente y, en consecuencia, generar productos de *software* seguros que disminuyan el riesgo de ataques por parte de terceros.

* + **Optimizadores**

Los optimizadores o “*Code* *Smells*” representan un indicador de un problema más profundo. Este tipo de observaciones **NO** representan un *Bug* o falla técnica dentro del código, señalan diversos factores que pueden causar deficiencias a futuro: Aumento de presencia de fallas, bajo desempeño e incluso aumento de deuda técnica.

Por lo anterior, es importante identificar de manera temprana los optimizadores necesarios para aplicar medidas correctivas, de esta forma aumentaría la calidad del producto de *software* y reducir el riesgo de presencia de fallas durante la operación de la solución tecnológica.

* **Severidades**

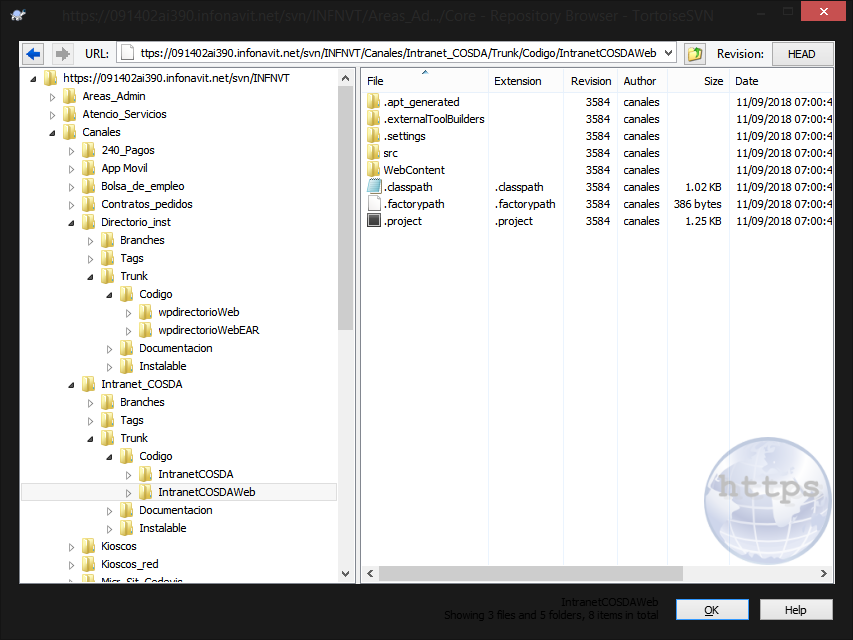
Los errores categorizados anteriormente son clasificados por severidades las cuales son definidas con base en 2 factores: **Impacto** y **Probabilidad**.

|  | **Impacto** | **Probabilidad** | **Descripción** |
| --- | --- | --- | --- |
| ***Blocker*** |  |  | * Alta probabilidad de ocurrencia * En caso de ocurrir puede provocar un error de gran escala que imposibilite la ejecución correcta de la operación del Instituto * Su corrección debe ser inmediata |
| ***Critical*** |  |  | * Baja probabilidad de ocurrencia * En caso de ocurrir puede provocar un error de gran escala que imposibilite la ejecución correcta de la operación del Instituto * Su corrección debe ser inmediata |
| ***Major*** |  |  | * Alta probabilidad de ocurrencia * En caso de ocurrir puede provocar un error de afectación menor |
| ***Minor*** |  |  | * Baja probabilidad de ocurrencia * En caso de ocurrir puede provocar un error de afectación menor |

* + 1. Ubicación en el Repositorio Institucional

Los artefactos evaluados (Código fuente y/o ejecutables) se encuentran en la siguiente ubicación dentro del repositorio:

[*https://091402ai390.infonavit.net/svn/INFNVT/Canales/Intranet\_COSDA*](https://091402ai390.infonavit.net/svn/INFNVT/Canales/Intranet_COSDA)



* + 1. Estructura de Archivo *sonar.properties*

La inspección se realiza por medio de la definición del *archivo sonar.properties* el cual indica a la herramienta *Sonar Scanner* los parámetros necesarios para ejecutar el análisis de calidad de código.

* + 1. Ejecución de Análisis de Código Fuente

El análisis de la calidad del código fuente se realiza por medio de la ejecución del siguiente comando: ***sonar-scaner.bat***



Ilustración Pantalla 1 de ejecución *sonar-scaner.bat*



Ilustración Pantalla 2 de ejecución *sonar-scaner.bat*

* + 1. Resultado del Análisis de Código Fuente

A continuación, se muestran los resultados del análisis de calidad de código categorizado por el tipo de incidencias encontradas:

Ilustración Gráfico de *Bug*s

Ilustración Gráfico de Vulnerabilidades

Ilustración Gráfico de Optimizadores

A continuación se enlistan las incidencias del tipo Vulnerabilidad, Bugs y Optimizadores las cuales fueron identificadas por medio del análisis de calidad de código y que deben ser atendidas a la brevedad posible con el objetivo de reducir el riesgo de fallo durante la operación.

| **Descripción de Incidencias** | | | |
| --- | --- | --- | --- |
| **Tipo** | **Severidad** | **Incidencia** | **Cantidad de Incidencias** |
| *Bug* | *Critical* | **Las llamadas de función no deben pasar argumentos adicionales**  Es posible llamar a una función *JavaScript* con más argumentos de los necesita, pero los argumentos adicionales serán ignorados por la ejecución de la función. | 16 |
| *Bug* | *Major* | **No se debe acceder a las propiedades de las variables con valores "*null*" o "*undefined*"**  Cuando a una variable se le asigna un valor no *undefined* o *null*, no se generan propiedades. Intentar acceder a las propiedades de una variable de este tipo resulta en un *TypeError*, causando una terminación abrupta del *script* si el error no se detecta en un bloque de captura. | 13 |
| *Bug* | *Major* | **Bloques ejecutados condicionalmente deben ser accesibles**  Las expresiones condicionales que son siempre verdaderas o falsas pueden conducir a código muerto. Dicho código siempre es considerado un *bug* y nunca debe usarse en la producción. | 10 |
| *Bug* | *Major* | **No deben utilizarse expresiones idénticas en ambos lados de un operador binario**  No deben utilizarse expresiones idénticas en ambos lados de un operador binario ya que puede provocar un error.  Esta regla ignora \*, +, y = | 9 |
| *Bug* | *Major* | **Las variables no deben asignarse a sí mismas**  No es necesario reasignar una variable a sí misma ya que esta sentencia es redundante y debe eliminarse, o bien la reasignación es un error y en su lugar se ha previsto otro valor o variable para la asignación. | 6 |
| *Bug* | *Major* | **Los nombres de las propiedades no deben ser duplicados dentro de una clase u objeto**  *JavaScript* permite duplicar el nombre de las propiedades en las clases y objetos pero sólo la última instancia de un nombre duplicado determina el valor real que será utilizado. Por lo tanto, cambiar los valores de otras ocurrencias de un nombre duplicado no tendrá ningún efecto y puede causar errores. | 1 |
| *Bug* | *Major* | **No se deben ignorar los valores de retorno de las funciones sin efectos secundarios**  Cuando la llamada a una función no tiene efectos secundarios. En tal caso, o bien la llamada a la función no es viable y debe ser eliminada o el código fuente no se comporta como se esperaba.  Para evitar la generación de falsos positivos, esta regla activa una incidencia sólo en la siguiente lista predefinida de clases inmutables en la *API* de *Java*:   * java.lang.String * java.lang.booleano * Entero java.lang. * java.lang.Doble * java.lang.float * java.lang.byte * java.lang.Character * java.lang.Short * java.lang.stackTraceElement * java.time.DayOfWeek * java.time.Duration * java.time.Instant * java.time.LocalDate * java.time.LocalDateTime * java.time.LocalTime * java.time.Month * java.time.MonthDay * java.time.OffsetDateTime * java.time.OffsetTime * java.time.Period * java.time.Year * java.time.yearMonth * java.time.ZonedDateTime * java.math.BigInteger * java.math.BigDecimal * java.util.Optional | 1 |
| *Bug* | *Major* | **No se deben utilizar operadores de igualdad estricta con tipos diferentes**  Realizar comparaciones de tipos diferentes utilizando operadores de igualdad estricta como === y !== siempre devolverá el mismo valor ya que no se realiza ninguna conversión de tipo antes de la comparación, por lo que se deduce se trata de un bug inmerso en el código. | 1 |
| *Bug* | *Major* | **Todos los *branches* de una estructura condicional no deberían tener exactamente la misma implementación**  Tener todos los *branches* en un *switch* o en una cadena *if* con la misma implementación es un error. Esta regla no se aplica a las cadenas *if* sin hacer uso de *else* o en caso de utilizar *switch* sin cláusulas por defecto. | 1 |
| *Bug* | *Minor* | **No deben utilizarse sentencias “with”**  El uso de la palabra clave “with” produce un error en el código JavaScript.  Usar “with” permite un acceso rápido a las propiedades de un objeto, asumiendo que ya están configuradas. Pero úsalo para acceder a alguna propiedad que no esté ya establecida en el objeto. Dado que los efectos de “with” dependen completamente del objeto que se le ha pasado, “with” puede ser peligrosamente impredecible, y nunca debe ser utilizado. | 33 |
| *Vulnerabilidad* | *Critical* | **El código no debe ser inyectado y ejecutado dinámicamente**  La función de *eval* es una manera de ejecutar código arbitrario en tiempo de ejecución. La evaluación dinámica del código es lenta y un problema potencial de seguridad cuando los argumentos no han sido validados correctamente. | 8 |
| *Vulnerabilidad* | *Critical* | **No se deben utilizar constructores de funciones**  Los constructores de funciones aumentan el riesgo de fallo ya que su ejecución evalúa los argumentos de cadena del constructor de forma similar a la forma en que funciona la evaluación, lo que podría exponer la aplicación programa a un código aleatorio no intencionado que puede ser lento y un riesgo para la seguridad.  En general, es mejor evitarlo por completo, especialmente cuando se utiliza para analizar datos *JSON*. Se sugiere utilizar las funciones *JSON* incorporadas de *ECMAScript* 5 o una biblioteca dedicada. | 3 |
| *Vulnerabilidad* | *Minor* | **No debe utilizarse "*alert(...)"***  *alert(....)* así como *confirm(....)* y *prompt(...)* puede de utilidad para depurar durante la etapa de desarrollo, pero en ambiente de producción este tipo de *pop-up* podría exponer información sensible a los atacantes, por lo que no debe mostrarse. | 14 |
| *Optimizador* | *Blocker* | **Las variables deberán declararse explícitamente**  Debe evitarse la creación de variables globales. Este escenario ocurre cuando se declara una variable dentro de una función o la cláusula *for* de un bucle *for-loop* sin keywords como *let, const* o *var*. | 76 |
| *Optimizador* | *Blocker* | **Los *Switch* *cases* deben terminar con una declaración incondicional "*break*"**  Cuando la ejecución no se termina explícitamente al final de un *switch* *case*, continúa ejecutando las sentencias del siguiente caso. Mientras que esto es a veces intencional, a menudo es un error que lleva a un comportamiento inesperado del aplicativo. | 1 |
| *Optimizador* | *Major* | **Deberían eliminarse los *dead store***  Una *dead* *store* ocurre cuando a una variable local se le asigna un valor, incluyendo nulo, que no es leído por ninguna instrucción posterior. Calcular o recuperar un valor sólo para luego sobre escribirlo o desecharlo, podría indicar un error grave en el código. Aunque no sea un error, representa un desperdicio de recursos.  Incluso la asignación nula a una variable es un almacén muerto si la variable no se utiliza posteriormente. Asignar nulo como una pista al recolector de basura solía ser una práctica común, pero ya no es necesario y ese código debería ser eliminado. | 83 |
| *Optimizador* | *Major* | **Las variables y funciones no deben volver a declararse**  Esta regla verifica que una declaración no utilice un nombre que ya está en uso. Es posible utilizar el mismo símbolo varias veces como variable o como función, pero hacerlo puede confundir al equipo responsable de dar mantenimiento al código. Adicional, es posible que tales reasignaciones se hagan por error, sin que el equipo de desarrollo se dé cuenta de que el valor de la variable es sobrescrito por la nueva asignación.  Esta regla también se aplica a los parámetros de función. | 50 |
| *Optimizador* | *Major* | **Las funciones no deben definirse dentro de los bucles**  Definir una función dentro de un bucle puede producir resultados inesperados. Esta función mantiene referencias a las variables que se definen en los alcances externos. Todas las instancias de función creadas dentro del bucle, por lo tanto, ven los mismos valores para estas variables, lo que probablemente no se espera. | 13 |
| *Optimizador* | *Major* | **Se deben eliminar los pares de paréntesis redundantes**  El uso de paréntesis, incluso los que no son necesarios para hacer cumplir un orden de operaciones deseado, puede aclarar la intención detrás de un fragmento de código. Sin embargo, los pares de paréntesis redundantes pueden inducir a error y deben eliminarse. | 6 |
| *Optimizador* | *Major* | **Los bloques anidados de código no deben dejarse vacíos**  Los bloques de código vacíos tanto deben llenarse o retirarse. | 5 |
| *Optimizador* | *Major* | **Dos *branches* de una estructura condicional no deberían tener exactamente la misma implementación**  Tener dos *branches* en un *switch* o en una cadena *if* con la misma implementación es un error. Esta regla no se aplica a las cadenas *if* sin hacer uso de *else* o en caso de utilizar *switch* sin cláusulas por defecto. | 3 |
| *Optimizador* | *Major* | **Las funciones no deben llamarse con y sin "nuevo"**  Las funciones de constructor, que crean nuevas instancias de objeto, sólo deben llamarse con new. Las funciones no constructivas no deben serlo. La mezcla de estos dos usos podría dar lugar a resultados inesperados en tiempo de ejecución. | 3 |
| Optimizador | *Minor* | **Las variables y funciones no deben volver a declararse**  Esta regla verifica que una declaración no utilice un nombre que ya está en uso. Es posible utilizar el mismo símbolo varias veces como variable o como función, pero hacerlo puede confundir al equipo responsable de dar mantenimiento al código. Adicional, es posible que tales reasignaciones se hagan por error, sin que el equipo de desarrollo se dé cuenta de que el valor de la variable es sobrescrito por la nueva asignación.  Esta regla también se aplica a los parámetros de función. | 34 |
| Optimizador | *Minor* | **Los wrapperss primitivos no deben ser instanciadas sólo para llamadas "*toString*" o "*compareTo*"**  Crear objetos de envoltura primitivos temporales sólo para la conversión *String* o el uso del método *compareTo* es ineficiente. En su lugar, se debe utilizar el método estático *toString* () o comparar el método de la clase de envoltura primitiva. | 6 |

El detalle completo del Análisis de Código Fuente puede ser consultado en la siguiente liga: [*http://10.80.1.146:9000/sonarqube7/projects*](http://10.80.1.146:9000/sonarqube7/projects)

* 1. Alineación Entre Diseño y Construcción de Soluciones Tecnológicas

El nivel de alineación entre la definición de la solución tecnológica generada en la etapa de Diseño de Soluciones y los artefactos generados en la etapa de Construcción es determinado con base en la siguiente tabla de niveles de conformidad:

| **Representación Gráfica de Alineación** | **Descripción de los Niveles de Alineación** | **Valor** |
| --- | --- | --- |
|  | La implementación no tiene características en común con el diseño de la solución | 1 |
|  | La implementación tiene algunas características en común con el diseño de la solución y tales características están implementadas en total conformidad con el diseño. Sin embargo, existen características en la especificación que no fueron implementadas y existen características implementadas que no son cubiertas en la especificación | 2 |
|  | Existe total concordancia entre el diseño de la solución y su implementación. Todas las características están implementadas conforme a la especificación y no existen características implementadas que no sean cubiertas por la especificación | 3 |

**Nota:** No se evaluó la alineación entre Diseño y Construcción de Soluciones Tecnológicas ya que no se generó documentación para la atención de la solicitud.

| **Aspecto Evaluado** | **Diseño** | **Construcción** | **Nivel de Alineación** |
| --- | --- | --- | --- |
| NA | NA | NA | NA |
| **Nivel de Alineación Final** | | | NA |

* 1. Consistencia entre Ambientes del Ciclo de Vida de Desarrollo

El nivel de consistencia entre Ambientes del Ciclo de Vida del Desarrollo es determinado con base en la siguiente tabla de niveles de conformidad:

| **Representación Gráfica de Alineación** | **Descripción de los Niveles de Consistencia** | **Valor** |
| --- | --- | --- |
|  | La tecnología evaluada no es consistente en términos de producto de *Software* utilizado, versión y *release* entre los 3 ambientes considerados en el ciclo de vida del desarrollo | 1 |
|  | La tecnología evaluada es consistente en términos de producto de *Software* utilizado, versión y *release* en 2 de los 3 ambientes considerados en el ciclo de vida del desarrollo | 2 |
|  | La tecnología evaluada es consistente en términos de producto de *Software* utilizado, versión y *release* en los 3 ambientes considerados en el ciclo de vida del desarrollo | 3 |

**Nota:** No se evaluó la alineación entre Diseño y Construcción de Soluciones Tecnológicas ya que no fue posible obtener información detallada sobre los ambientes utilizados.

| **Tecnología** | **Ambiente de Desarrollo** | **Ambiente de Pruebas** | **Ambiente de Producción** | **Nivel de alineación de ambientes** |
| --- | --- | --- | --- | --- |
| NA | NA | NA | NA | NA |
| **Nivel de Alineación Final** | | | | NA |

* 1. Cumplimiento Documental para Liberación en Ambiente Productivo

**Nota:** No se evaluó el cumplimiento Documental para Liberación en Ambiente Productivo ya que no se generó documentación para la atención de la solicitud.

| **Nombre del Documento** | **Entregado** | **No Entregado** |
| --- | --- | --- |
| Plan de trabajo detallado de la solución | NA | NA |
| Diseño detallado de la solución | NA | NA |
| Plan de Implementación (Instalación, pruebas, retorno) | NA | NA |
| Componentes tecnológicos de la solución | NA | NA |
| Acuerdo de Niveles de Servicio | NA | NA |
| Solicitud de formalización del acuerdo de control de atributos de acceso | NA | NA |
| Acuerdo de control de atributos de acceso | NA | NA |
| Solicitud de custodia de cuenta privilegiada | NA | NA |
| Formato de liberación de aplicación | NA | NA |
| Formato de excepción a políticas de seguridad | NA | NA |
| Carta de Aceptación de Pruebas | NA | NA |
| Pruebas de Ancho de Banda | NA | NA |
| Formato de Alta o Modificación de Procesos | NA | NA |
| Atención a Usuario Final, Procesos de Gestión y CASIA | NA | NA |
| Documentos Varios (Memoria Técnica) | NA | NA |
| 01 (Servicio) - Información General | NA | NA |
| 02 (Servicio) - Procedimiento de Operación | NA | NA |
| 03 (Servicio) - Procedimiento de Recuperación | NA | NA |
| 04 (Servicio) - Monitoreo Disponibilidad | NA | NA |
| 05 (Servicio) - Cédula de Respaldos | NA | NA |
| Formatos para Control en CMDB | NA | NA |
| Formato RG1000 | NA | NA |
| Catálogo de Servicios | NA | NA |
| Capacitación a las áreas de Soporte Técnico a la Producción, al usuario final, y a Operaciones | NA | NA |

* 1. Resultado Integral

El resultado global es una vista de alto nivel sobre las observaciones que surgieron a partir de la evaluación de la solución tecnológica. Esta vista es generada por medio de los indicadores generados por cada uno de los rubros evaluados y un resultado integral considerado el porcentaje de valor de cada rubro.

* **Indicadores de Evaluación de Calidad de Código**
  + **Resultado General:** Este indicador muestra la cantidad total de líneas de código del proyecto y la cantidad de incidencias (*Bugs*, Vulnerabilidades y Optimizadores) encontradas por medio del análisis de calidad de código.

Ilustración Resultado General - Código

Ilustración Resultado General - Incidencias

* + **Comparación entre Versiones:** Este indicador muestra la comparativa entre en análisis de calidad de código aplicado a la **última versión desplegada del proyecto** (Versión operando en ambiente productivo y la **versión actualizada más reciente** (Versión con modificaciones recientes que aún no ha sido desplegada en ambiente productivo).

La evaluación de la solución tecnológica consideró únicamente la última versión desplegada en ambiente productivo ya que no se identifica una versión más reciente del proyecto.

Ilustración Gráfico Comparativo - Código

Ilustración Comparativo – Incidencias General

Ilustración Gráfico Comparativo de *Bug*s

Ilustración Gráfico Comparativo de Vulnerabilidades

Ilustración Gráfico Comparativo de Optimizadores

* **Indicadores de Alineación Entre Diseño y Construcción de Soluciones**
  + **Nivel de alineación por aspecto evaluado**: El nivel de alineación determina el grado de apego entre el diseño de soluciones tecnológicas y su construcción. Se utiliza la tabla anterior para definir el nivel de alineación de los aspectos evaluados.
  + **Nivel de alineación final**: Se genera al obtener el promedio de los niveles de alineación obtenidos.

**Nota:** No se especifican Indicadores de Alineación entre Diseño y Construcción de Soluciones ya que no se generó documentación para la atención de la solicitud.

| Aspecto Evaluado | Nivel de Alineación |
| --- | --- |
| Aspecto 1 | NA |
| Aspecto 2 | NA |
| Aspecto 3 | NA |
| Nivel de alineación final | NA |

* **Indicadores de Consistencia entre Ambientes**
  + **Nivel de alineación por aspecto evaluado**: Se obtiene con base en los criterios definidos en la tabla anterior.
  + **Nivel de alineación final**: Se genera al obtener el promedio de los niveles de alineación obtenidos.

**Nota:** No se especifican Indicadores de Consistencia entre Ambientes ya que no fue posible obtener información detallada sobre los ambientes utilizados.

| Tecnología Evaluada | Nivel de Alineación |
| --- | --- |
| Tecnología 1 | NA |
| Tecnología 2 | NA |
| Tecnología 3 | NA |
| Nivel de alineación final | NA |

* **Cumplimiento Documental**
  + **Cumplimiento Documental para Liberación en Ambiente Productivo**: Identificar el nivel de cumplimiento de entrega documental para realizar la liberación de la solución tecnológica en ambiente productivo

**Nota:** No se especifican Indicadores de Cumplimiento Documental para Liberación en Ambiente Productivo ya que no se generó documentación para la atención de la solicitud.

| **Indicador** | **Valor** |
| --- | --- |
| Cumplimiento documental para liberación en ambiente productivo | NA |

* 1. Propuestas de Acciones de Mejora
* **Calidad de Código**

Se sugiere atender de forma inmediata, las incidencias de tipo **Vulnerabilidad** y las de tipo ***Bug*** y **Optimizadores** con severidad ***Blocker***, ***Critical*** con el objetivo de reducir el riesgo de fallo durante la operación. Una vez atendidas las incidencias con severidad más alta, se sugiere establecer un plan de mantenimiento con el objetivo de corregir las incidencias con menor nivel de severidad.

Se recomienda ampliamente el uso de la extensión del IDE, “**Sonar Lint**”. El *plugin* antes mencionado permite identificar incidencias desde la construcción de la solución.

* **Alineación entre Diseño y Construcción de Soluciones Tecnológicas**

No se generan propuestas de acción de mejor debido a que no fue posible evaluar la Alineación entre Diseño y Construcción de Soluciones Tecnológicas.

* **Consistencia entre Ambientes del Ciclo de Vida de Desarrollo**

No se generan propuestas de acción de mejor debido a que no fue posible evaluar la Consistencia entre Ambientes del Ciclo de Vida de Desarrollo.

* **Cumplimiento Documental para Liberación en Ambiente Productivo**

No se generan propuestas de acción de mejor debido a que no fue posible evaluar el Cumplimiento Documental para Liberación en Ambiente Productivo.

1. Conclusiones

Preservar la calidad en el ciclo de vida de las soluciones tecnológicas es un factor de gran relevancia para el Instituto ya que reduce el riesgo de una afectación durante la operación del negocio.

Con base en el resultado de la evaluación de soluciones tecnológicas plasmada en este documento se logran identificar brechas de calidad que deben ser atendidas con el objetivo de incrementar la calidad de las soluciones tecnológicas generadas dentro del Instituto. Las brechas especificadas en este documento sirven de referencia para evitar errores similares en futuros proyectos.