¿Cómo evaluar y comparar software?

Versión 1.7 - 2019

Contenido

1. Resumen

Este documento sirve de guía para la comparación de soluciones de software tanto de propietario como *open source*. Ofrece al lector recomendaciones para considerar los distintos factores que influyen en la selección de un producto de software y facilita el análisis, proveyendo herramientas para dicha tarea.

2. Identificar los candidatos

El primer paso para encontrar una solución de software para la organización es identificar los distintos candidatos que podrían, eventualmente, ser implementados. Para ello, basta con una simple búsqueda en internet; para el caso de productos de propietarios, es suficiente consultar a los proveedores habituales si incluyen dentro de sus portfolios de productos una solución para el problema a resolver.

Una vez realizada esta búsqueda y enumerados los candidatos, se puede comenzar a realizar el análisis en base a los factores de la siguiente sección.

3. Primera comparación de atributos de los candidatos en función de las necesidades de la organización

El **objetivo** de esta etapa es **seleccionar** entre los **potenciales candidatos** unos pocos que, en primera instancia, parecen más adecuados para la solución. Los seleccionados serán luego comparados en profundidad.

Para realizar esta tarea, es recomendable obtener información de las fuentes disponibles, por ejemplo:

- **Experiencias** de otras organizaciones y/o profesionales del medio. Muchas veces, consultar a otras organizaciones (tarea que puede realizarse a través de Agesic) si han realizado un análisis similar o si tienen experiencia en algunos de los productos candidatos, puede dar una guía para la primera comparación.
- Información de *reviews* y comparativos disponibles en **internet**.
- Información de **consultoras** como Gartner, Deloitte, etc. Muchas veces, esta información se puede encontrar en internet o mediante una suscripción a estas empresas. Agesic puede facilitar el acceso a esta información.
- Información suministrada por los **fabricantes** en los casos de software licenciado y documentación suministrada por las **comunidades** en los casos de software *open source*. También se podrán realizar consultas específicas.

Como guía, se presentan algunos **atributos** que se recomienda considerar en esta etapa de la evaluación:

- Funcionalidades.

- Costos.
- Tamaño del despliegue o del mercado.
- Soporte.
- Dependencias.
- Mantenimiento.
- Confiabilidad.
- Performance.
- Escalabilidad.
- Usabilidad.
- Seguridad.
- Flexibilidad.
- Interoperabilidad.
- Tipos de licencia.
- Aspectos legales.

Naturalmente, estos atributos sirven para comparar tanto software licenciado como de código abierto; sin embargo, veremos que para la comparación de software de código abierto se pueden considerar otros aspectos, principalmente, porque muchas veces se puede obtener más información debido a la apertura de su código.

Evidentemente, si hay atributos fuera de esta lista que son específicamente importantes para la solución o para su organización, estos deben ser incorporados en la evaluación.

a. Funcionalidades

Este atributo es fundamental en el proceso de evaluación. El producto debe tener, al menos, las funcionalidades básicas necesarias requeridas por la organización.

Muchas veces es útil realizar una pequeña **lista de requerimientos fundamentales**, los que no deben faltar, y validar que cada producto cumpla con estos requerimientos.

Generalmente, los sitios web de los distintos productos/proyectos presentan una lista resumida de sus funcionalidades y estas pueden servir para realizar esta validación.

b. Costos y beneficios

Los costos son un aspecto fundamental en el proceso de selección. Si bien, por lo general, el software libre no presenta costos de adquisición, es importante analizar los **costos operativos y de implementación** que implican.

Por lo tanto, es importante considerar **todos** los costos relacionados a una determinada solución, sea libre o propietaria.

Este modelo está compuesto de una **guía** y una **planilla** de carga de datos. Esta permite comparar los resultados de distintos proyectos, funcionalidad que podrá ser utilizada para comparar las distintas alternativas de este análisis.

El modelo se divide en la estimación de **dos variables** de un proyecto: por un lado, los **costos;** y, por otro, los **beneficios** en un plazo de **10 años**. En este documento analizaremos cuáles son los costos y beneficios específicos que se deben considerar para realizar el análisis de un producto de software. El objetivo es obtener los valores de entrada para la planilla de análisis y comparación.

i. Análisis de costos de una solución de software

El **plan de cuentas** para el análisis de proyectos propuesto en el modelo está dividido en tres categorías: "componentes **técnicos"**, "componentes **profesionales**" y "componentes de **mantenimiento".** El resultado del análisis deberá generar los resultados de cada factor en base a estas categorías.

Los componentes **técnicos** se refieren a los costos de hardware y software directos, los componentes **profesionales** se refieren a costos de mano de obra y los componentes de **mantenimiento** se refieren a los costos **operativos** de un proyecto (componente muchas veces olvidado en el análisis de costos, pero fundamental para obtener un resultado correcto del análisis).

No es trivial estimar el **costo de los recursos humanos internos** para determinada tarea. Una posible forma de hacerlo es utilizando una fórmula como la siguiente: "*costo promedio anual de un ingeniero de soporte * número de ingenieros * porcentaje de tiempo dedicado al soporte de la solución".*

Se recomienda revisar el plan de cuentas del trabajo de análisis de proyectos y verificar que todos los componentes de costos del proyecto están siendo considerados y estimados.

Como guía particularmente orientada a proyectos de software, consideremos los costos de acuerdo a cada una de las etapas del proyecto. En la planilla adjunta a este documento se encuentra la relación de cada uno de estos componentes, con el plan de cuentas del "Modelo de análisis de costos y beneficios de proyectos".

1. Selección y búsqueda

    a. Costo de la **evaluación**: El proceso de evaluación de un producto de software trae consigo costos; en general, estos costos provienen de horas de consultoría o recursos humanos que trabajan en el proceso. Si este proceso es subcontratado se podrá tener un costo exacto de este componente. De lo contrario, se podrá estimar en base a un costo promedio de los recursos dedicados dentro de la organización a dicha tarea.
    b. Costo de **prueba de concepto**: La prueba de concepto puede traer varias componentes de costos
        i. **Recursos humanos / Consultoría**: En general, el análisis implica

costos en recursos humanos que realizan la implementación del producto y su posterior prueba. Estos costos pueden ser comparables a los de implementación final, dependiendo del alcance del piloto.

ii. **Hardware**: Ya sea por el costo de alquiler o de compra de equipamiento para realizar la prueba, el uso de hardware traerá algún costo asociado.

iii. **Licencias de software**: En algunos casos es necesario adquirir licencias para realizar el piloto.

2. Adquisición e implementación

    a. Costo de **adquisición de las licencias**: En los casos de software libre, este costo será nulo. En los casos de productos licenciados, si se adquiere una licencia, esta tendrá un costo. Muchas veces, los nuevos productos se venden en modalidad servicio y no presentarán costos de inversión de adquisición (ya que se compra de un servicio con derecho a uso y no se adquiere una licencia).
    b. Costo de **instalación**: El costo de implementación estará ligado a los recursos humanos necesarios para la instalación del producto en los servidores de la organización. En los casos en que el producto es adquirido en forma de servicio, este componente no existirá. Debe considerarse que muchas veces las soluciones *open source* presentan mayores esfuerzos de instalación que los productos propietarios. Esto puede estimarse en base a referencias que puedan encontrarse en foros y a la cantidad de componentes y dependencias que pueda tener el producto.
    c. Costo de **adaptación**: Es posible que el producto deba adaptarse a las necesidades de la organización o, al menos, configurarse para ello. Este esfuerzo implica costos en recursos para realizar dicha tarea.
    d. Costo de **integración**: En muchos casos es necesario integrar el nuevo producto a otros sistemas de la organización. Esta integración implica la elaboración o configuración de interfaces.

3. Gestión del cambio organizacional

    a. Costo de **migración de datos** de uso: Muchas veces, existen datos de usuarios que deben ser movidos al nuevo producto. En estos casos, es necesario realizar un esfuerzo para extraer los datos del sistema de origen e insertarlos en el sistema destino.
    b. Costo de entrenamiento y **capacitación**: El nuevo producto implicará realizar capacitación a los usuarios para que incorporen el producto.
    c. Costo de **cambios en procesos internos**: Muchas veces, incorporar un producto de software implica cambios en los procesos de la organización, ya sea para aplicar una buena práctica o porque el producto solo soporta determinado tipo de proceso. Esto implica una inversión en tiempo y, en general, en una menor eficiencia cuando el proceso recién se adopta.

4. Mantenimiento

d. Costos de **soporte**: Se deben considerar todos los costos de soporte, desde el soporte a usuarios hasta el soporte al sistema y su base de datos.
e. Costo de **actualizaciones** y **mantenimiento**: Las aplicaciones deben ser mantenidas, deben realizarse los respaldos de datos correspondientes y actualizarse a las nuevas versiones.
f. Costos de mantenimiento de **infraestructura**: El hardware tiene vida útil y a veces es necesario reemplazarlo luego de que se convierte en obsoleto. En ocasiones, las aplicaciones se montan sobre hardware en alquiler y se considera un costo operativo.

i. Beneficios

Se recomienda considerar los **beneficios** de la implantación de cada producto, de acuerdo a la guía del "Modelo de análisis de costos y beneficios de proyectos".

b. Popularidad, tamaño del despliegue o del mercado

La **popularidad** de un producto de software es un factor importante a considerar. Para los productos licenciados, una forma de conseguir la información es solicitar casos de éxito que estén vinculados al tipo de organización o a la aplicación que se esté buscando.

En el caso de los productos de software libre, este factor es aún más importante. El análisis se debe realizar mediante internet, por ejemplo, midiendo las interacciones en los foros y medios de la comunidad, la cantidad de integrantes activos en la comunidad y también los casos de éxito conocidos en el medio.

c. Soporte

El desarrollo del modelo de software libre no limita la presencia de distribuciones comerciales, revendedores de valor añadido (VAR) y una oferta de soporte. Por el contrario, el acceso ilimitado al código y a la información del producto fomenta a terceros con intereses comerciales a participar en los proyectos. La existencia de estos elementos es, en general, un buen indicador, ya que implica que el producto tiene una viabilidad a largo plazo. Otra forma de identificar productos con una comunidad de soporte es evaluar la disponibilidad de cursos, conferencias, libros y revistas especializadas.

La carencia de actividad comercial alrededor del producto puede significar que el producto es aún inmaduro o que es muy "de nicho". En estos casos, es importante identificar cuáles pueden ser las fuentes de soporte y si estas son sustentables.

d. Mantenimiento / continuidad del producto

La realidad indica que los programas no son completamente estáticos. Los cambios son necesarios, las necesidades se van modificando y ningún programa es completamente perfecto. El mantenimiento de un software es importante y está muy vinculado al punto anterior: “Soporte”. En la medida en que un producto es mantenido, es más fácil que se generen *fixes* en base a solicitudes de soporte.

Este punto también está relacionado a la popularidad del producto. En la medida en que la popularidad sea alta, mayor será la participación en las comunidades para el software libre y mayor será la demanda del producto en los licenciados.

El punto fundamental aquí es el **nivel de inversión en esfuerzo** para el producto y la **estabilidad** de la organización que mantiene la aplicación. En los productos libres, algunas de los factores que podemos analizar son los siguientes:

- ¿Quién se siente dueño del producto y lo lleva adelante? ¿Quiénes son los principales contribuyentes y cuál es su interacción con la comunidad?
- ¿Qué tan grande es la base de usuarios? ¿Qué tan rápido está creciendo? Las opiniones de los usuarios son muy importantes para que se pueda mantener el producto.
- ¿Cuál es la frecuencia de nuevas versiones y qué tanto se parece al *roadmap*?
- ¿Qué tan grande es la base de desarrolladores? ¿Está creciendo o decreciendo?
- ¿Hay algún proceso bien definido (por ejemplo, votación) para adoptar decisiones importantes en la comunidad?
- ¿Hay una guía de estilo de programación y documentación definida explícitamente? ¿Se sigue esa guía?

e. Dependencia

Hace referencia al grado de dependencia que tiene una organización respecto del software o hardware ofrecido por un proveedor. Por ejemplo, supongamos que la organización cuenta con recursos de hardware disponibles con sistema operativo Linux y la organización, además, posee personal calificado para operar esta tecnología y no así tecnología Microsoft. Entonces, un producto que pueda funcionar sobre Linux podría ser más atractivo o eficiente para la organización. Así también podrá ocurrir con el tipo de base de datos o servidor de aplicación (*middleware*).

f. Confiabilidad

La confiabilidad mide qué tanto una aplicación funciona y produce el resultado esperado. La confiabilidad es difícil de medir y depende mucho de cómo se utiliza la aplicación. Los reportes de problemas de otros usuarios no necesariamente muestran una mala confiabilidad; es común que las personas se quejen de programas muy confiables cuando

las expectativas son altas tanto para el que genera el producto como para el que lo consume.

Sin duda, la mejor forma de medir la confiabilidad es probarlo con **carga real** de trabajo; sin embargo, muchas veces se puede encontrar información que nos permitirá tener una idea de la confiabilidad de una aplicación

En particular, es más probable que una aplicación **madura** sea confiable. El sitio web del proyecto ya nos puede ofrecer una descripción de la madurez del producto. Si el proyecto declara que la aplicación no está lista para usuarios finales, entonces, en general, es así. Sin embargo, algunas veces ocurre que los desarrolladores son muy perfeccionistas y nunca admiten que la aplicación ya tiene madurez suficiente.

Los proyectos de software libre que se utilizan por una comunidad amplia tienden, en general, a ser confiables; si no lo fueran, seguramente, no serían utilizados por tantos usuarios. Los distintos repositorios de código libre incluyen información que pueden ayudarnos a validar la estabilidad; por ejemplo, **Freecode** incluye una medida de madurez de los proyectos y **SourceForge** incluye el *Development Status*.

g. Performance

Es importante saber que el producto a utilizar tendrá una performance aceptable para el uso que queremos darle, que los recursos necesarios de infraestructura son razonables y que disponemos de ellos. Los foros y listas de correo de los proyectos pueden ayudar a conseguir información más detallada. La mejor manera de testear la performance es probarlo también en condiciones de carga de trabajo real y específicas al tipo de uso que se le dará.

h. Escalabilidad

La escalabilidad en este contexto se refiere a la capacidad de la aplicación de manejar un cierto **tamaño de problema** o de **cantidad de datos**. Si la expectativa es que la aplicación pueda manejar un volumen importante de datos o que pueda ser ejecutado en instancias distribuidas en forma paralela, este punto es importante. Lo mejor es buscar evidencia previa de que el programa se utilizó de esta manera y que fue satisfactorio. Una referencia que a veces ayuda es saber si hay una herramienta de *testing* que pruebe la escalabilidad en los niveles buscados.

i. Usabilidad

La usabilidad mide la **calidad de la interfaz aplicación-usuario**. Una aplicación con buena usabilidad permite aprender a usarla en forma rápida y natural.

Si bien no tienen una interfaz de usuario, se puede medir que tan fácil o naturales les resulta a los programadores usar las librerías o API. Generalmente, una API debería hacer

las cosas fáciles de usar; y las cosas difíciles, al menos posibles. Una forma de medir esto es analizar algunas muestras de código y verificar qué tan fáciles son de usar.

En los casos de aplicaciones con interfaz de usuario, una buena manera de validar es utilizar alguna demo provista por el proyecto o ver alguna captura de pantalla, para intentar medir la naturalidad y la facilidad de uso. Agesic publica **guías de usabilidad** que pueden ser aplicadas para medir si la aplicación está alineada. A veces, los proyectos intentan aplicar ciertas guías de usabilidad; esto puede ser un buen indicador de que es un tema considerado en el desarrollo de la aplicación.

j. Seguridad

Evaluar la seguridad de un producto no es nada sencillo, en gran medida, porque muchas veces depende del uso, del ambiente donde se implementa y de los requerimientos necesarios para asegurar el nivel de seguridad buscado.

Una manera de atacar este problema es identificar los **requerimientos de seguridad aceptables**. En el sitio web de Agesic se pueden encontrar recomendaciones actualizadas que podrían servir de base para la identificación. Luego, habría que buscar evidencia de que el producto cumple con estos requerimientos y que, a nivel general, el equipo de desarrollo trabaja en forma activa para eliminar las vulnerabilidades detectadas.

La evaluación de consultoras independientes puede servir para realizar este análisis. **Coverty** (http://www.coverity.com/), en particular, presenta en forma gratuita análisis de seguridad para aplicaciones de código abierto. En particular, además, los reportes de Coverty identifican *rungs*. *Rung 1* significa que los desarrolladores del proyecto revisan activamente los reportes de seguridad y *Rung 2* significa que han eliminado todos los posibles defectos identificados en los reportes anteriores.

Algunos productos propietarios se certifican según los **Criterios Comunes** (CC). Es raro que productos de código abierto lo hagan debido a los costos que implica. Si algún producto a analizar tuviera la evaluación según CC, es importante revisar el *Security Target*. Este documento, que debe ser público, especifica información importante sobre la evaluación. Si el documenta especifica un escenario de prueba conceptualmente distinto del que se aplicaría, entonces puede que los requerimientos importantes de seguridad que son de interés no fueran validados y que la prueba no aplicara al análisis.

Por supuesto, si la seguridad es un punto muy importante para la solución, uno debería luego realizar un análisis con mayor profundidad sobre este tema.

k. Flexibilidad

La flexibilidad del producto se refiere a su **capacidad de adaptarse o ser adaptado** a requerimientos que hoy no son contemplados, pero que puede ser necesario incorporar en el futuro. Los productos de código abierto tienen una clara ventaja en este punto, pues

pueden ser modificados en un futuro. Sin embargo, realizar estos cambios puede ser más o menos sencillo de acuerdo a cómo está desarrollado el producto.

Se puede buscar algunos indicadores de qué tan flexible es una aplicación, por ejemplo, buscando la disponibilidad de plantillas, *plug-ins* o API. La disponibilidad de documentación y su calidad son otros indicadores, ya sea de cómo está creado el producto o de cómo realizar agregados o cambios.

La disponibilidad de herramientas de testeo permite realizar cambios con menores riesgos, al poder realizar pruebas de los cambios con bajo costo.

l. Interoperabilidad

Es importante prever que la aplicación pueda, al menos, interoperar con otros sistemas que estén actualmente relacionados. La aplicación de estándares para la interoperabilidad es un buen indicador. Se pueden utilizar también requerimientos de interoperabilidad recomendados por Agesic.

m. Tipos de licencia y aspectos legales

Los aspectos legales de un producto de software son importantes y deben considerase en una comparación. Es recomendable que antes de incorporar un producto, se lean las **condiciones legales**, generalmente, definidas en la licencia.

A diferencia del resto de los atributos, la parte legal muchas veces es desestimada en los productos propietarios, suponiéndose que no es algo que se deba considerar cuando se está pagando por el producto. Sin embargo, es recomendable que se analicen los términos de la licencia, especialmente, el End User License Agreement (EULA). En algunos casos, estos documentos pueden tener cláusulas que no son aceptables para la organización, como por ejemplo, habilitar al proveedor a deshabilitar el software sin aprobación judicial, habilitar el envío de información al proveedor, prohibir la publicación de información sobre el producto, etc.

Por supuesto que también es recomendable analizar las condiciones de licencia de los productos *open source*. Si bien los términos de las licencias son, por lo general, muchos más abiertos que para los productos propietarios, existen muchas discusiones acerca del licenciamiento *open source*. Distintos desarrolladores y comunidades aplican distintos tipos de licencia, de acuerdo a las motivaciones que lo llevan a realizar su aporte. En particular, algunas licencias exigen que si el código es modificado, este debe ser publicado.

i. Garantías legales

Algunos usuarios piensan que al comprar un producto propietario están protegidos legalmente en caso de errores o funcionamiento distinto al esperado. Sin embargo, prácticamente todas las licencias limitan la responsabilidad legal sobre fallas o errores de los productos.

La mayoría de las licencias *open source* aclaran específicamente que no ofrecen garantías. El tipo de licencia más común, GPL, específica que **por defecto** no se dará garantía sobre el producto.

De cualquier manera, este factor es de relativa importancia, dado que nunca es práctico llegar a una instancia legal, sino que es recomendable verificar la calidad del producto antes de incorporarlo.

ii. Licencias *open source*

El primer paso es determinar el tipo de licencia del producto y verificar que, efectivamente, es una licencia *open source*. La enorme mayoría de las licencias están basadas en GPL y LGPL, la licencia estilo BSD y la licencia estilo MIT. Se pueden revisar las plantillas de estas licencias en los siguientes links:

- http://opensource.org/licenses/bsd-license.php
- http://opensource.org/licenses/mit-license.php

El tipo de licencia *open source* no es demasiado importante si solo se va a usar el producto y no se va a modificar. Cuando se va a modificar el producto cambiando parte de su código, es muy importante entender el tipo de licencia que tiene.

Existen dos tipos de licencia fundamentales: licencias ***copylefting*** y licencias ***non-copylefting***. Las aplicaciones que son publicadas con licencia *copylefting* pueden ser modificadas, pero las modificaciones deben ser publicadas bajo las mismas condiciones en que está publicado el original. La mayoría de las aplicaciones/licencias utilizan *copylefting*, como por ejemplo, GPL.

n. Tabla de comparación

La tabla de comparación presenta de una manera clara y resumida los resultados para cada uno de los factores de comparación. En algunos casos es razonable presentar resultados cualitativos y en otros, cuantitativos.

Es recomendable utilizar al menos una **escala de tres niveles** como resultado: “A”, “B” o “C”, dependiendo de si el resultado es “Muy bueno”, “Bueno” o “Malo”.

En otros casos, cuando se tiene buena información y se realiza un análisis de mayor profundidad, se podrá establecer puntajes en base 10 y se podrán ponderar cada uno de los factores de acuerdo al peso que tienen para la organización. Un ejemplo de tabla podría ser el siguiente:

Factor	Peso	Puntaje	Total (Peso X Puntaje)
Funcionalidades			
Costos			
Popularidad			
Soporte			
Mantenimiento			
Dependencias			
Confiabilidad			
Performance			
Escalabilidad			
Usabilidad			
Seguridad			
Flexibilidad			
Interoperabilidad			
Tipos de licencias			
TOTAL			

Es importante considerar que si se trata de una licitación y se utilizan ponderadores en el análisis, es recomendable **publicar** los ponderadores que se utilizarán para la comparación en el pliego. Así, se presentará una comparación más transparente.

5. Análisis a fondo del o los candidatos (preselección)

Una vez completada la selección, es recomendable realizar pruebas para validar la información incluida en la primera parte del análisis y verificar que el producto se adapta a las necesidades de la organización.

Este paso en particular es semejante tanto para un producto propietario como para uno *open source*. El objetivo es analizar los mismos atributos que se consideraron en el paso anterior, pero con mucha mayor profundidad y con acceso a la realización de una prueba real sobre el producto, validándose así los distintos factores considerados. Principalmente, el ejercicio puede validar que el funcionamiento del producto es el esperado, probando directamente los procesos de interés.

La performance y la escalabilidad se pueden verificar realizando una prueba de carga. En muchos casos, este punto no es tan importante como lo era hace unos años, principalmente, debido a las mejoras en las capacidades de cómputo actuales. De cualquier manera, existen en la actualidad muchas aplicaciones en las que este factor es importante. Se debe analizar caso a caso si este punto es relevante o no.

Es importante realizar la evaluación sobre la versión estable que se implementaría en la organización. Las diferentes versiones pueden tener comportamientos muy distintos.

Cuando es posible implementar primero el producto en procesos o ambientes de baja criticidad, se puede realizar una prueba sobre un escenario real. Y esta puede ser una buena alternativa antes de soportar procesos críticos de la organización sobre un producto nuevo.

6. Conclusiones

Es razonable analizar y comparar opciones de software propietario y *open source* **en conjunto**, usando esencialmente el mismo enfoque. Sin embargo, la forma de conseguir la información es distinta para cada caso.

Existen muchas variables que se deben considerar y se deberá ponderar cuáles son las variables de mayor importancia para la organización. Por supuesto, los costos serán un factor de mucho peso, pero también se debe validar al menos que el resto de las características cumplen el mínimo de las necesidades de la organización.

Luego de realizar el análisis, por lo general será necesario realizar un **informe técnico** en el que se justifiquen los resultados y se realice una recomendación sobre qué producto incorporar.

Una posible estructura para dicho documento podrá incluir la siguiente información:

- Introducción, comentando la necesidad que impulsa la incorporación del producto (el problema a resolver).
- La solución recomendada, con una breve descripción de por qué es la mejor solución.
- Información acerca del proceso seguido para realizar el análisis. Se puede citar este documento como fuente.
- Introducción a las distintas opciones analizadas. Información técnica y comercial, si aplica.
- Comparación de los distintos productos, citando las fuentes y los supuestos considerados.
- Si se realizaron pruebas de concepto, hay que dedicar una sección a la metodología y los resultados obtenidos de dichas pruebas.
- Conclusión, con recomendaciones y comentarios breves respecto de las razones que justifican dicha selección.

Si el análisis se realiza en base a las propuestas de una licitación, es recomendable incluir en el TDR la solicitud de información en cada uno de los puntos mencionados, de forma tal de facilitar el proceso de comparación. En el sitio de Agesic se podrán encontrar algunos requerimientos recomendados para incluir en un pliego de este tipo.