ONTI
Oficina Nacional de Tecnologías de Información

Código de buenas prácticas en el desarrollo de software público[1]

Versión 1.0.1 - 21-Nov-2019

Guía técnica para el desarrollo sustentable de software en la Administración Pública

[1] *El software público es aquella solución tecnológica de software desarrollada, usada, modificada y distribuida para facilitar el cumplimiento de objetivos gubernamentales y sociales por el Estado y para el Estado.*

¿Qué encontrarás en este documento?

Este documento contiene lineamientos generales y recomendaciones específicas de la ONTI para el desarrollo de software en la Administración Pública. Vas a encontrar varias referencias al Decálogo Tecnológico ONTI porque ambos lineamientos se basan en los mismos principios rectores.
El mismo propone ser un marco de referencia para los organismos acerca de las mejores prácticas en el desarrollo de software público. Su confección es el resultado de un esfuerzo colaborativo entre distintos referentes y organismos en varias instancias incluyendo los eventos de la Comunidad Digital AR. Dado que las tecnologías se van mejorando y aparecen nuevas y más potentes herramientas la guía de buenas prácticas será revisada y actualizada al menos 2 veces por año. Invitamos a toda la comunidad de agentes técnicos a sumarse a esta actualización comunicándose con el equipo de software público: softwarepublico@modernizacion.gob.ar

¿Cómo leer esta guía?

La guía técnica para las buenas prácticas de desarrollo de software contiene 7 (siete) puntos principales con algunas subguías en forma de hipervínculo para aquellos temas que requieren mayor profundidad. A su vez los 7 puntos están divididos en tres niveles de acción: El *Ideal* es el rumbo aspiracional de mediano-largo plazo que la ONTI recomienda a las áreas de sistemas en la Administración Pública en donde se encuentran las mejores prácticas y tendencias en en la industria TIC a nivel regional y global. El nivel *Mínimo* consiste en un punto de transición hacia el *Ideal* a partir del cual se pretende que las direcciones de sistemas logren alcanzar en el mediano plazo a través de buenas prácticas. *Por dónde empezar* describe los primeros pasos a seguir para comenzar el camino a la innovación y sustentabilidad tecnológica en el desarrollo de software.

1. Entendé las necesidades de los usuarios

Entender los contextos y las necesidades reales de los usuarios es fundamental para que tus sistemas cumplan su función y para que puedas evaluar mejor las prioridades de desarrollo.

Ideal

- ➔ Creá mapas de historias de usuario (BL: 6935169 - Fecha: 04-11-19 11.54) para entender la experiencia de todos los distintos tipos de usuarios, incluyendo personas con discapacidad
- ➔ Desarrollá prototipos (BL: 6935166- Fecha: 04-11-19 11.54) para probar el servicio con usuarios reales
- ➔ Tomá decisiones basadas en la recolección y análisis de datos objetivos, incluyendo Test A/B, métricas de uso de los servicios y pruebas de usabilidad
- ➔ Diseñá teniendo en cuenta toda la experiencia del usuario incluyendo las acciones que realiza fuera de los sistemas digitales
- ➔ Evaluá tu servicio con las herramientas de la W3C

Mínimo

- ➔ Cumplí las normativas de accesibilidad para garantizar igualdad real de oportunidades y trato a todas las personas
- ➔ Conocé los estándares de la Dirección Nacional de Servicios Digitales
- ➔ Investigá desde qué tipo de dispositivos acceden tus usuarios, para poder priorizar el diseño y desarrollo de una interfaz y experiencia acorde

Por dónde empezar

- ➔ Consultá el Decálogo Tecnológico ONTI #1 - Conocé tu proyecto
- ➔ Consultá el Decálogo Tecnológico ONTI #7 - Asegurá que tus soluciones sean accesibles
- ➔ Creá un primer mapa de historia de usuario (BL: 6935169 - Fecha: 04-11-19 11.54) para entender la experiencia de tus usuarios más frecuentes

2. Aprovechá la nube y la virtualización

Alojar tus aplicaciones y tomar ventaja de las herramientas y soluciones de infraestructura que ofrece la nube aumenta la flexibilidad, escalabilidad y seguridad, a su vez facilita la reducción de tiempos y costos.

Ideal

- ➔ Considera utilizar una solución de Nube para implementar tu servicio antes que cualquier otra opción.
- ➔ Las características de tu proyecto podrán indicar como la mejor opción una solución de Nube pública, privada o incluso híbrida. En cualquier caso utilizando entornos virtualizados para tomar ventaja de la economía, flexibilidad y dinamicidad que estos aportan.
- ➔ Aprovechá los entornos de desarrollo en la nube pública[2]
- ➔ Optimizá tareas comunes con herramientas de automatización[3]
- ➔ Utilizá contenedores para el despliegue de aplicaciones y el mantenimiento[4]
- ➔ Gestioná y automatizá la configuración de despliegues con herramientas de DevOps [5]

Mínimo

- ➔ Opta por un entorno de Nube para tu desarrollo, aprovechando sus facilidades y herramientas.Si tu aplicación consiste en una combinación de funcionalidades comunes (formularios, ABM, gráficos) entonces considerá las herramientas para desarrollo de aplicaciones en la nube (ofrecidas como SaaS) antes de iniciar un nuevo desarrollo[6]
- ➔ Minimizá las diferencias entre los ambientes de desarrollo y producción para favorecer un despliegue continuo de los cambios a tus sistemas

Por dónde empezar

- ➔ Consultá el Decálogo Tecnológico ONTI #3 - Preferí soluciones que utilicen la Nube y buscá cuál es la mejor opción de Nube para desarrollar e implementar el servicio o solución.

[2] Por ejemplo Cloud9 o CodeAnywhere
[3] Por ejemplo Zapier o IFTTT
[4] Por ejemplo Docker, Kubernetes, CloundFoundry
[5] Por ejemplo, Puppet, Chef, Ansible
[6] Por ejemplo Zoho Creator o Google Apps

3. Reutilizá y compartí software y datos

Desarrollar nuevos productos digitales que reutilicen software y datos permiten aprovechar el trabajo realizado por otros, optimiza recursos, reduce costos y facilita la interoperabilidad entre servicios favoreciendo a la eficiencia y la transparencia

Ideal

- ➔ Compartí tu código en el repositorio oficial de la República Argentina y mantenelo actualizado siguiendo el proceso de apertura y reutilización del software público (en desarrollo)
- ➔ Utilizá una licencia abierta y estándar (BL: 7163323- Fecha: 17-11-19 16.59) , como MIT, GPL o Creative Commons
- ➔ Cumplí con las pautas técnicas de interoperabilidad para el intercambio de información entre sistemas.
- ➔ Exponé tus datos mediante una API estilo REST que cumpla con las especificaciones OpenAPI
- ➔ Documentá y probá tu API con herramientas diseñadas para el caso[7]
- ➔ Diseñá tus aplicaciones como un SaaS para que otros organismos las puedan utilizar con facilidad[8]

Mínimo

- ➔ Utilizá plataformas comunes ya desarrolladas por el gobierno, como GDE, Argentina.gob.ar, Mi Argentina, Sistema de Identidad Digital o Autenticar
- ➔ Elegí cómo encarar un nuevo desarrollo de acuerdo a la necesidad, disponibilidad y sustentabilidad (BL: 7163347 - Fecha: 17/11/2019 17:01:00)
- ➔ Exponé tus datos mediante una API estilo REST que cumpla con las especificaciones OpenAPI
- ➔ Utilizá un framework abierto y activo (BL: 7163306 - Fecha: 17-11-19 16.58)
- ➔ Aplicá y conocé los estándares de la Plataforma Web Abierta de la W3C
- ➔ Favorecé las tecnologías y lenguajes de código abierto por sobre las propietarios para reducir costos y evitar la dependencia de oferentes[9]
- ➔ Utilizá tecnologías con una comunidad activa para contar con soporte y recibir actualizaciones periódicas de funcionalidad y seguridad.

[7] Por ejemplo Swagger
[8] Recomendamos la metodología 12factor para cubrir todas las necesidades técnicas involucradas en el proceso
[9] Por ejemplo el MEAN stack (Express, AngularJS y Node.js), el LAMP stack (Linux, Apache, MySQL y PHP, Python, Java) o ASP.NET Core.

- ➔ Gestioná las bases de datos de tus aplicaciones (relacionales y no relacionales) a modo de *cluster* para que puedan crecer de manera horizontal y favorecé las tecnologías abiertas[10]
- ➔ Asegurá una estrategia de mantenimiento y soporte en tus contrataciones.
- ➔ Desarrollá codigo modular para que los componentes también se puedan reutilizar
- ➔ Agregá tu software al catálogo de software público
- ➔ Abrí tus datos públicos en el portal de datos públicos

Por dónde empezar

- ➔ Consultá el Decálogo Tecnológico ONTI #4 - Utilizá estándares abiertos y soluciones interoperables
- ➔ Consultá el Decálogo Tecnológico ONTI #5 - Elegí plataformas y soluciones comunes de Gobierno
- ➔ Consultá el Decálogo Tecnológico ONTI #6 - Desarrollá soluciones reutilizables y compartilas
- ➔ Analizá cómo abrieron su código algunas agencias de la administración pública, como por ejemplo la Dirección Nacional de Servicios Digitales en cuanto a la implementación de Poncho
- ➔ Unite a la Comunidad Digital para compartir y consultar experiencias sobre soluciones y herramientas
- ➔ Contactá al equipo de software público para asesorarte sobre cómo abordar tus nuevos desarrollos

[10] Relacionales: PostgreSQL 10x, MySQL 5.7x. No relacionales: MongoDB 3.6x, Elasticsearch 5.5x, Redis 5.0x

4. Probá el servicio de punta a punta

Realizar pruebas de tu sistema en todo momento del desarrollo te permite adaptarte fácilmente a los cambios y asegura que las herramientas cumplan las necesidades para las cuáles fueron diseñadas.

Ideal

- ➔ Integrá prácticas DevOps en el ciclo de producto para realizar despliegues continuos que te permitan realizar pruebas automatizadas en todos los ambientes de manera rápida y segura.
- ➔ Realizá pruebas en todo el proceso de desarrollo además de contemplarla como una fase del ciclo.
- ➔ Escribí pruebas unitarias como parte de tus tareas diarias de desarrollo.
- ➔ Asegurate que por cada unidad funcional exista una prueba automatizada como caso de fallo[11]
- ➔ Reducí la deuda técnica refactorizando tu código regularmente.
- ➔ Incluí pruebas de seguridad dentro de los requerimientos del producto.
- ➔ Automatizá las pruebas funcionales y mantenelas actualizadas.
- ➔ Separá adecuadamente los ambientes de desarrollo, prueba, aceptación y producción y llevá a cabo pruebas de integración continua con herramientas de automatización[12]
- ➔ Escribí y mantené actualizado tu plan de pruebas unitarias, de integración y funcionales, y aseguráte de integrarlos y ejecutarlos desde una herramienta de integración continua.

Mínimo

- ➔ Desarrollá tus sistemas orientados a pruebas o Test-Driven-Development
- ➔ Llevá a cabo pruebas unitarias para las funcionalidades de tu código
- ➔ Reducí la deuda técnica refactorizando tu código regularmente
- ➔ Incluí pruebas de seguridad dentro de los requerimientos del producto

Por dónde empezar

- ➔ Realizá un Plan de Pruebas y registra su ejecución y resultados para cada unidad funcional
- ➔ Escribí casos de prueba unitarios para tus nuevas funcionalidades
- ➔ Incorporá la refactorización como parte de tus tareas diarias de desarrollo
- ➔ Utilizá herramientas de inspección[13] de código para aumentar la calidad semántica.

[11] Recomendamos utilizar frameworks de pruebas automatizadas como Cucumber, Selenium y JUnit
[12] Por ejemplo, Jenkins, Circle CI, Travis CI, GitLab
[13] Codeclimate, Sonarqube

5. Protegé al software y a los usuarios

Mantener la seguridad del software es una responsabilidad que sólo puede cumplirse con el esfuerzo conjunto de todas las partes involucradas en el ciclo de vida del mismo..

Ideal

- ➔ Involucrá tempranamente al equipo de seguridad informática en tu equipo u organismo
- ➔ Entendé los requerimientos de disponiblidad y rendimiento de tu software
- ➔ Revisá el diseño de tu software con la ayuda de un especialista de seguridad.
- ➔ Aplicá los principios de diseño seguro de software Recomendaciones de Seguridad en el Desarrollo de Software (BL: 7163400 - Fecha: 17/11/2019 17:05:00)
- ➔ Considera las recomendaciones de la Guía de seguridad para aplicaciones
- ➔ Tené en cuenta la normativa vigente respecto a la Política de Seguridad de la Información Modelo (dispo ONTI 1/2015)
- ➔ Cumplí con la Ley de Protección de los Datos Personales vigente
- ➔ Publicá el código de tu aplicación, recibí mejoras de terceros.
- ➔ En tu desarrollo basado en pruebas, incluí pruebas de seguridad.
- ➔ Elegí herramientas y librerías con soporte activo y aptas para producción
- ➔ Planificá para mantener la seguridad durante toda la vida útil de tu software
- ➔ Ofrecé un canal para recibir reportes de vulnerabilidades de la comunidad experta

Mínimo

- ➔ Involucrá tempranamente al equipo de seguridad informática de tu equipo u organismo y designá un responsable de seguridad en tu equipo.
- ➔ Entendé los requerimientos de disponiblidad y rendimiento de tu solución.
- ➔ Aplicá los principios de diseño seguro de software Recomendaciones de Seguridad en el Desarrollo de Software (BL: 7163400 - Fecha: 17/11/2019 17:05:00)
- ➔ Cifrá las comunicaciones de tu software.
- ➔ Validá todas las entradas a tu aplicación.
- ➔ Realizá pruebas de seguridad sobre tu aplicación antes de entrar en producción..
- ➔ Minimiza la cantidad de información sensible de tus usuarios que almacenes

Por dónde empezar

- ➔ Consultá el Decálogo Tecnológico ONTI #8 - Protegé al sistema y a los usuarios.
- ➔ Consultá la Guía rápida "Consideraciones importantes para la Seguridad de Aplicaciones" (BL: 7163334 - Fecha: 17/11/2019 17:00:00)

6. Utilizá metodologías ágiles e iterativas en todo el ciclo de vida del software

Un paradigma flexible y adaptable a los cambios permite abordar integralmente el desarrollo de productos digitales adecuados a las necesidades de los usuarios, los recursos y las capacidades de tu equipos maximizando la entrega de valor y minimizando los costos.

Ideal

- ➔ Empezá construyendo una visión compartida de tu producto con tu equipo, los interesados y comprendé las necesidades de tus usuarios
- ➔ Construí una hoja de ruta a partir de usuarios ideales que respondan a necesidades reales
- ➔ Definí en conjunto con el equipo el objetivo de cada finalización de ciclo de trabajo que oriente el desarrollo de tus productos
- ➔ Priorizá las entregas de valor y las tareas necesarias para desplegar tus productos y servicios
- ➔ Desarrollá prototipos y versiones mínimas de tu producto para validar ideas (BL: 6935166- Fecha: 04-11-19 11.54)
- ➔ Planificá tus entregas y reflexioná periódicamente sobre los procesos y los productos (BL: 7163371- Fecha: 17/11/2019 17:03:00)
- ➔ Incorporá técnicas de DevSecOps *y* DevOps para entregar valor frecuentemente de forma segura
- ➔ Construí un equipo multidisciplinario y lo más autónomo posible para evitar dependencias
- ➔ Desarrollá las capacidades del equipo utilizando programación por pares
- ➔ Visibilizá las tareas y el flujo de trabajo para que puedan ser compartidos con tu equipo

Mínimo

- ➔ Entregá valor frecuentemente a tus usuarios desarrollando prototipos y versiones mínimas de tu producto para validar ideas (BL: 6935166- Fecha: 04-11-19 11.54)
- ➔ Reflexioná sobre los procesos y los productos que entrega tu equipo (BL: 7163371- Fecha: 17/11/2019 17:03:00)
- ➔ Colaborá con tu equipo y otros miembros de la organización para generar una cultura de confianza
- ➔ Mejorá continuamente tus productos y servicios realizando experimentos planteándote hipótesis a validar con datos cualitativos y cuantitativos
- ➔ Construí un equipo multidisciplinario y lo más autónomo posible
- ➔ Visibilizá las tareas y el flujo de trabajo para que puedan ser compartidos con tu equipo

- ➔ Priorizá los requerimientos de tu software utilizando roles como el de encargado de producto y que conozca la visión del mismo
- ➔ Consensuá con tu equipo la definición de trabajo terminado para gestionar las expectativas de tus productos.

Por dónde empezar

- ➔ Hacé los cursos de Agilidad y Scrum en el INAP
- ➔ Priorizá los requerimientos de tu software utilizando roles como el de encargado de producto y que conozca la visión del mismo
- ➔ Construí un equipo multidisciplinario y lo más autónomo posible
- ➔ Explorá herramientas de código abierto que trabajen en la nube que apoyen la construcción de software con dinámicas iterativas e incrementales
- ➔ Reflexioná sobre los procesos y los productos que entrega tu equipo (BL: 7163371- Fecha: 17/11/2019 17:03:00)

7. Documentá y compartí el conocimiento de tus sistemas

Documentar correctamente aumenta la sustentabilidad de aplicación, ya que facilita que otras personas la instalen, mantengan y utilicen.

Ideal

- ➔ Compartí toda la información sobre tus productos para que sea lo más clara posible y disponible en el Repositorio Público
- ➔ Creá documentación tanto para personal técnico como para usuarios nuevos
- ➔ Documentá el código mediante comentarios y entidades comprensibles
- ➔ Utilizá alguna herramienta de documentación automática[14]
- ➔ Automatizá el versionado de tu software
- ➔ Registrá los eventos de tus sistemas con herramientas abiertas[15]

Mínimo

- ➔ Escribí instrucciones detalladas de instalación y probá que los pasos te lleven a instalar correctamente el sistema
- ➔ Si estás reutilizando un software, documentá exhaustivamente todos los cambios que realices al código fuente, para que tu equipo o algún equipo futuro los pueda reproducir en caso de actualizar el software
- ➔ Utilizá alguna herramienta de seguimiento de defectos[16]
- ➔ Incorporá algún sistema de versionado para tu software[17]
- ➔ Mantené informados a tus usuarios respecto a actualizaciones y cambios en el sistema

Por dónde empezar

- ➔ Mirá y toma como referencia la documentación de otros proyectos oficiales, como Poncho y Autenticar
- ➔ Visibilizá tu proyecto a través de una página pública con enlaces a otros recursos relevantes[18]
- ➔ Defini al comienzo del proyecto todos los entregables y documentos que deberá disponerse con la finalización del desarrollo del software.

[14] Por ejemplo Doxygen o Javadoc
[15] ElasticSearch 5.5, Logstash, Kibana, Sentry
[16] Por ejemplo, utilizando Taiga, Redmine, GitLab o GitHub para software subido al repositorio oficial en GitHub, o Phabricator para proyectos más grandes y complejos
[17] Por ejemplo utilizando GitHub, BitBucket, GitLab.
[18] Puede ser una página en Argentina.gob.ar, un README en el repositorio oficial en GitHub, o una GitHub Page

Glosario

- ## API:

  Una API (Application programming interface o interfaz de programación de aplicaciones) es un conjunto de métodos y parámetros claramente definidos para permitir que diversos componentes se comuniquen entre sí de manera programática. Mediante una API es posible exponer la funcionalidad de un sistema para que terceras partes puedan acceder a ella independientemente del lenguaje de programación utilizado.

- ## Caso de fallo:

  Un caso de prueba en el que se fuerza a un escenario a los posibles fallos en lugar de los pasos usuales correctos.

- ## Casos de prueba unitarios:

  Una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo en diseño estructurado o en diseño funcional una función o un procedimiento, en diseño orientado a objetos una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido, entonces el estado final es válido también.
  https://es.wikipedia.org/wiki/Prueba_unitaria

- ## Deuda técnica:

  Se refiere exclusivamente a aquellos errores en la codificación que, pese a que no afectarán al funcionamiento de la misma, dificultan su mantenimiento, ampliación, despliegue o desarrollos posteriores, por ser código desarrollado con malas prácticas como la duplicidad o la ausencia de una correcta refactorización.
  https://es.wikipedia.org/wiki/Deuda_t%C3%A9cnica

- ## DevOps:

  Un conjunto de prácticas para automatizar procesos que existen entre las actividades de desarrollo y las de operaciones con el fin de construir, probar, y entregar software de manera frecuente y continua. Se basa también en una cultura o

paradigma de colaboración entre equipos que tradicionalmente funcionaban de manera aislada. Al igual que *Agile* o metodologías ágiles, sus prácticas y valores proponen gestionar la incertidumbre típica de la industria del conocimiento.

- ## DevSecOps:

  Prácticas DevOps a las cuáles se le suma la mirada y prácticas relacionadas a la seguridad por diseño.

- ## Entregar valor:

  En términos del paradigma *Agile,* un producto o una funcionalidad entregará valor cuando lo haga para sus usuarios.

- ## IaaS:

  Se entiende como Infraestructura como Servicio (Infrastructure as a service) la capacidad de proveer al consumidor procesamiento, almacenamiento, redes y otros recursos informáticos fundamentales donde el consumidor puede implementar y ejecutar software arbitrario, donde puede incluir sistemas operativos y aplicaciones. El consumidor no administra ni controla la infraestructura subyacente de la nube, pero tiene control sobre los sistemas operativos, el almacenamiento y otras aplicaciones implementadas; y posiblemente un control limitado de los componentes de red seleccionados (por ejemplo, firewalls host). Ejemplos: Amazon EC2, Windows Azure, Google Compute Engine u Openstack.

- ## Integración Continua:

  La Integración Continua (Continuous Integration -CI-) es un conjunto de prácticas de desarrollo que orientan a los equipos a integrar el código fuente en un repositorio compartido de manera continua durante diferentes períodos del día. En cada integración el código es verificado de forma automática permitiendo a los equipos detectar preventivamente cualquier error disminuyendo el esfuerzo dedicado a la prevención y corrección de errores.

- ## Priorización:

  Determinar una escala de valoración de determinadas acciones en relación a uno o más tipos de actores, ya sean usuarios, interesados (*stakeholders*) u otros. Existen distintas maneras de priorizar. La más conocida es la Matriz de Eisenhower.

- ## Programación por pares:

  Técnica desarrollada por el *XP programming* en la que dos programadores participan en un esfuerzo combinado de desarrollo en un sitio de trabajo. Cada miembro realiza una acción que el otro no está haciendo actualmente. Por ejemplo, mientras que uno codifica las pruebas de unidades el otro piensa en la clase que satisfará la prueba.

- ## Prueba de integración:

  Prueba en la que unidades individuales son combinadas y probadas como un grupo. El propósito es exponer y detectar de los defectos posibles en la combinación entre las unidades integradas.

- ## Pruebas de usabilidad:

  Las pruebas de usabilidad consisten en seleccionar a un grupo de usuarios de una aplicación y solicitarles que lleven a cabo las tareas para las cuales fue diseñada, en tanto el equipo de diseño, desarrollo y otros involucrados toman nota de la interacción, particularmente de los errores y dificultades con las que se encuentren los usuarios. No es necesario que se trate de una aplicación completamente terminada, pudiendo tratarse de un prototipo.
  https://es.wikipedia.org/wiki/Prueba_de_usabilidad

- ## Prueba Unitaria

  Un test o prueba unitaria puede definirse como un fragmento de programa escrito y mantenido por los desarrolladores de un determinado producto de software constituido por muestras del código fuente para las cuales se diseñan y ejecutan pruebas de esas pequeñas unidades de código. El resultado de las mismas pasará o fallará de acuerdo a la consistencia en la comparación entre el comportamiento esperado y el de la ejecución de la prueba.

- ## Pruebas funcionales o e2e (*end to end*)

  Estas pruebas están diseñadas para ser contrastadas con las especificaciones de requerimientos y que puedan ser validados. No evalúa el procesamiento sino sus resultados. Simula el funcionamiento usual sin tomar en cuenta ningún supuesto o resultado esperado. Cuando se conjugan varias pruebas funcionales como parte de un flujo o conjunto de requerimientos asociados las pruebas funcionales pasan a ser pruebas de punta a punta o *end to end.*

- ## Refactorizar:

  La refactorización es la parte del mantenimiento del código que no arregla errores ni añade funcionalidad. El objetivo, por el contrario, es mejorar la facilidad de comprensión del código o cambiar su estructura y diseño y eliminar código muerto, para facilitar el mantenimiento en el futuro. Añadir nuevo comportamiento a un programa puede ser difícil con la estructura dada del programa, así que un desarrollador puede refactorizarlo primero para facilitar esta tarea y luego añadir el nuevo comportamiento.
  https://es.wikipedia.org/wiki/Refactorizaci%C3%B3n:

- ## REST:

  En sentido amplio se trata de cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP. Es posible diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST de Fielding y también es posible diseñar interfaces XMLHTTP de acuerdo con el estilo de llamada a procedimiento remoto (RPC), pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas, aunque RPC no es un ejemplo de REST.
  https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional:

- ## SaaS:

  Acrónimo para “Software como servicio”. Se trata de un modelo de distribución de software en el que el proveedor produce y alberga la aplicación y la disponibiliza a los usuarios a través de la Internet. hosts applications and makes them available to customers over the Internet.

- ## Seguimiento de defectos (*bug-tracking system*):

  Aplicación diseñada para ayudar a asegurar la calidad de software y asistir a los programadores y otras personas involucradas en el desarrollo y uso de sistemas informáticos en el seguimiento de los defectos de software.
- https://es.wikipedia.org/wiki/Sistema_de_seguimiento_de_errores

- ## Test A/B:

  Experimentos aleatorios con dos variantes, A y B, siendo una la de control y la otra la variante. Otra forma de referirse generalmente a los test A/B es con el término split test, aunque este último método se aplica cuando se realizan experimentos con más de dos variantes.
  https://es.wikipedia.org/wiki/Test_A/B

- ## Test-Driven-Development

- Desarrollo guiado por pruebas de software, o Test-driven development (TDD) es una práctica de ingeniería de software que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring). Para escribir las pruebas generalmente se utilizan las pruebas unitarias (unit test en inglés). En primer lugar, se escribe una prueba y se verifica que las pruebas fallan. A continuación, se implementa el código que hace que la prueba pase satisfactoriamente y seguidamente se refactoriza el código escrito. El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione. La idea es que los requisitos sean traducidos a pruebas, de este modo, cuando las pruebas pasen se garantizará que el software cumple con los requisitos que se han establecido.
  https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas

- ## Tipos de usuario / personas:

  Conocida como “test persona”, se trata de un personaje ficticio diseñado para representar a un usuario/a potencial de un sistema. Al identificar y considerar las necesidades, prejuicios, impedimentos, etc. de una persona pueden descubrirse escenarios y áreas de un problema que no se habían contemplado si no fuera por la construcción narrativa que involucra su desarrollo.

- ## Versionado de código o control de versiones:

  Se trata de una categoría de herramientas que sirven para que los equipos de desarrollo administren los cambios al código fuente a lo largo del tiempo a través del registro de todas las modificaciones realizadas al código en una base de datos diseñada para tales propósitos. Esto permite comparar y modificar diferentes versiones del código a través del tiempo.