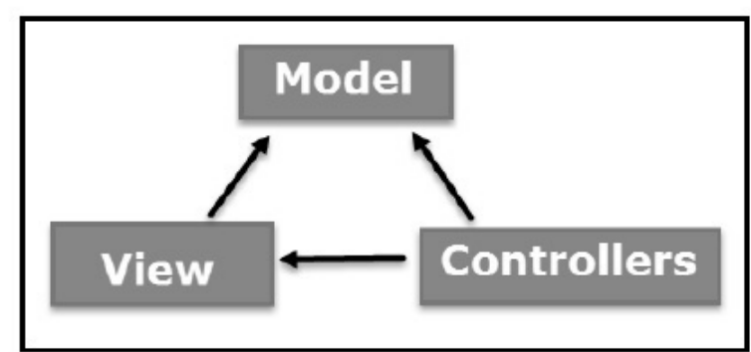
**MODELO MVC - Clase 9**

El Model-View-Controller (MVC) es un patrón arquitectónico que separa una aplicación en tres componentes lógicos principales: el modelo, la vista y el controlador. Cada uno de estos componentes está diseñado para manejar aspectos de desarrollo específicos de una aplicación. MVC es uno de los marcos de desarrollo web estándar de la industria más utilizados para crear proyectos escalables y extensibles.

**Componentes MVC**

Los siguientes son los componentes de MVC:



**Modelo**

El componente Modelo corresponde a toda la lógica relacionada con los datos con la que trabaja el usuario. Esto puede representar los datos que se transfieren entre los componentes View y Controller o cualquier otro dato relacionado con la lógica empresarial. Por ejemplo, un objeto Cliente recuperará la información del cliente de la base de datos, la manipulará y actualizará los datos en la base de datos o la utilizará para representar datos.

**Vista**

El componente Ver se utiliza para toda la lógica de la interfaz de usuario de la aplicación. Por ejemplo, la vista Cliente incluirá todos los componentes de la interfaz de usuario, como cuadros de texto, menús desplegables, etc., con los que interactúa el usuario final.

**Controlador**

Los controladores actúan como una interfaz entre los componentes Model y View para procesar toda la lógica comercial y las solicitudes entrantes, manipular datos utilizando el componente Model e interactuar con las Vistas para generar el resultado final. Por ejemplo, el controlador del Cliente manejará todas las interacciones y entradas de la Vista del Cliente y actualizará la base de datos usando el Modelo del Cliente. El mismo controlador se utilizará para ver los datos del Cliente.

**ASP.NET MVC**

ASP.NET admite tres modelos principales de desarrollo: páginas web, formularios web y MVC (controlador de vista de modelo). El marco ASP.NET MVC es un marco de presentación liviano y altamente comprobable que está integrado con las características existentes de ASP.NET, como páginas maestras, autenticación, etc. Dentro de .NET, este marco se define en el ensamblado System.Web.Mvc. La última versión de MVC Framework es 5.0. Usamos Visual Studio para crear aplicaciones ASP.NET MVC que se pueden agregar como plantilla en Visual Studio.

**Características de ASP.NET MVC**

ASP.NET MVC proporciona las siguientes características:

* Ideal para desarrollar aplicaciones complejas pero ligeras.
* Proporciona un marco extensible y conectable, que se puede reemplazar y personalizar fácilmente. Por ejemplo, si no desea utilizar el Razor integrado o el motor de visualización ASPX, puede utilizar cualquier otro motor de visualización de terceros o incluso personalizar los existentes.
* Utiliza el diseño basado en componentes de la aplicación al dividirlo lógicamente en componentes de modelo, vista y controlador. Esto permite a los desarrolladores gestionar la complejidad de los proyectos a gran escala y trabajar en componentes individuales.
* La estructura MVC mejora el desarrollo basado en pruebas y la capacidad de prueba de la aplicación, ya que todos los componentes pueden diseñarse en función de la interfaz y probarse utilizando objetos simulados. Por lo tanto, ASP.NET MVC Framework es ideal para proyectos con un gran equipo de desarrolladores web.
* Admite todas las amplias funcionalidades existentes de ASP.NET, como autorización y autenticación, páginas maestras, enlace de datos, controles de usuario, membresías, enrutamiento ASP.NET, etc.
* No utiliza el concepto de View State (que está presente en ASP.NET). Esto ayuda a crear aplicaciones, que son livianas y brindan control total a los desarrolladores.

Por lo tanto, puede considerar MVC Framework como un marco importante construido sobre ASP.NET que proporciona un gran conjunto de funcionalidades adicionales que se centran en el desarrollo y las pruebas basadas en componentes.

**AJAX**

La mejor tecnología para construir páginas web dinámicas es Ajax . El código JavaScript incrustado en la página HTML se utiliza para enviar solicitudes al servidor. En el lado del servidor, se requiere algún procesamiento para manejarlos, encontrar la información o almacenar los datos. Para hacer eso, necesitamos un marco especializado. El marco siempre tiene una parte de JavaScript y, a veces, una parte del lado del servidor en otro lenguaje de secuencias de comandos. Muchos de ellos existen en varios lenguajes de programación, en todos los entornos, pero retenemos aquí solo los más utilizados.

**¿Por qué un marco?**

En realidad, este marco es el motor Ajax descrito por JJ Garrett y destinado a suprimir la espera del usuario al acceder al servidor.

El marco proporciona funciones clásicas entre navegadores para usar el objeto XMLHttpRequest. Pero un marco puede ir más allá y permitir crear "aplicaciones web ricas", aplicaciones con una interfaz gráfica de usuario y otras funciones de software de escritorio que se ejecutan a través de un navegador, mientras intercambian datos con un servidor remoto.

**Características de un framework Ajax**

Cualquier framework Ajax es capaz de comunicarse con el servidor, y así, leer datos o enviarle datos o comandos. En el último caso, se requiere un script del lado del servidor.

Los marcos a menudo agregan componentes que hacen uso de la comunicación asíncrona con el servidor.

Los ejemplos clásicos son botones, paneles con pestañas, cuadrículas, cuadros de lista y otros widgets similares.

Un ejemplo más innovador, las "cajas", se implementan cada vez más, y Lightbox y Slimbox son dos de ellas. Hay galerías de imágenes que las colocan una al lado de la otra en la pantalla y que están haciendo uso de Ajax para mostrarlas instantáneamente.

El marco también puede ser controlado por el servidor y, en este caso, los componentes se crean en el servidor con un lenguaje de secuencias de comandos como PHP y se envían al navegador. Ajax se usa para transmitir las acciones del usuario a la parte del servidor y para manejar los resultados.

La capacidad de trabajar sin conexión que ofrece HTML 5 también es un complemento del marco Ajax.

**¿Qué framework Ajax elegir?**

Desea utilizar un marco Ajax para crear páginas dinámicas sin tener que reescribir todo el material usted mismo y probarlo en todos los navegadores...

Pero son tantos que se pregunta cuál usar. ¡No querrás hacer una mala elección y verte obligado a rehacer todas las páginas con un marco diferente!

Es menos probable que te arrepientas de la elección de un marco si adoptas uno de los más utilizados porque se complementan continuamente con nuevas extensiones y tienen un buen soporte por parte de los usuarios.

Entre estos, hay jQuery, bien documentado, Mootools que es modular, Ext JS que ofrece muchos widgets para RIA.

La interacción con un lenguaje del lado del servidor también debería guiar la elección.

**Ejercicios:**

* ¿En qué casos se aplica AJAX conocidamente?

AJAX se aplica para construir páginas web dinámicas que mejoran la experiencia del usuario al permitir la actualización de contenido sin recargar toda la página. Es útil en casos donde se requiere la interacción asíncrona con el servidor, como formularios de búsqueda en vivo, carga de contenido dinámico (como comentarios o secciones de noticias), validación de formularios, y la implementación de aplicaciones web ricas (RIA) con interfaces de usuario avanzadas, como galerías de imágenes interactivas (Lightbox, Slimbox) y widgets.

* Desarrolle brevemente MVC

El patrón arquitectónico **MVC (Model-View-Controller)** separa una aplicación en tres componentes principales:

* **Modelo**: Gestiona la lógica de los datos de la aplicación. Incluye la recuperación, manipulación y actualización de los datos, como la interacción con la base de datos.
* **Vista**: Representa la interfaz de usuario y muestra la información del modelo. Incluye elementos con los que el usuario interactúa, como cuadros de texto y botones.
* **Controlador**: Actúa como intermediario entre el modelo y la vista. Procesa las solicitudes del usuario desde la vista, realiza las operaciones necesarias sobre el modelo y actualiza la vista con el resultado.

**APIs – Clase 10**

En esta clase tendrá la oportunidad de aprender las características de las API REST, API First, Restul, XML y Json.

**API FIRST**

La irrupción de nuevos dispositivos e internet de las cosas, ha reemplazado la idea de hacer una plataforma web para convertirse en un plataforma que pueda estar presente, utilizada, en la mayor cantidad de dispositivos presentes: móviles, tabletas, TV y en cualquier cosa que esté conectado a interrnet.

La estrategia API First debe empezar con la idea de definir la API internamente antes que cualquier otro desarrollo. Ya sea ese proyecto una *web* o una *app* móvil, o ambas.

Deben fijarse los recursos que van a ser utilizados previamente. Antes, la API era creada después que las WEBs fueran creadas para interactuar con ellas.

Eso representaba situaciones complicadas al no haber sido optimizadas para tal uso, además de una importante carga de *legacy code*.

**API REST**

Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por **Roy Fielding**, quien también realizó la especificación HTTP.

En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales.

Las redes sociales y muchas empresas generan negocios gracias a REST y las API REST.

Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de API para servicios de Internet.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad.

A veces es preferible **una solución más sencilla de manipulación de datos como REST**.

Es un p**rotocolo cliente/servidor sin estado**: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.

Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como **protocolo cliente-caché-servidor sin estado**: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro **la misma respuesta para peticiones idénticas**. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.

Un servicio sin estado genera una respuesta que enlaza al número de la siguiente página del conjunto y deja que el cliente haga lo que tenga que hacer para conservar ese valor.

Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro:

**POST** (crear), **GET**(leer y consultar), **PUT**(editar) y **DELETE** (eliminar).

**Los objetos en REST siempre se manipulan a partir de la URI**.

Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST.

La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

**Interfaz uniforme**: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

**Sistema de capas**: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

**Uso de hipermedios**: es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

**VENTAJAS QUE OFRECE REST PARA EL DESARROLLO.**

**Separación entre el cliente y el servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos.

Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

**Visibilidad, fiabilidad y escalabilidad**. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas.

Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta.

Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

**La API REST siempre es independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo

Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

**JSON**

Es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por Douglas Crockford. Puede ser utilizado independientemente de JavaScript

Muchos ambientes de programación poseen la capacidad de leer (analizar; parse) y generar JSON. Convertir una cadena a un objeto nativo se denomina *parsing*, convertir un objeto nativo a una cadena para que pueda ser transferido a través de la red se denomina *stringification*

JSON es una cadena cuyo formato recuerda al de los objetos literales JavaScript.

Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript - cadenas, números, arreglos, booleanos, y otros literales de objeto. Esto permite construir una jerarquía de datos.



Se pueden insertar espacios en blanco entre cualquier par de tokens. Exceptuando algunos detalles de codificación, eso describe completamente el idioma.

Los JSON son cadenas - útiles cuando se quiere transmitir datos a través de una red. Debe ser convertido a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. JavaScript posee un objeto global [JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON) que tiene los métodos disponibles para convertir entre ellos.

Un objeto JSON puede ser almacenado en su propio archive: un archivo de texto con una extension .json, y una [MIME type](https://developer.mozilla.org/es/docs/Glossary/MIME_type) de application/json. JSON es sólo un formato de datos — contiene sólo propiedades, no métodos.

JSON requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas. Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione.

Es posible validar JSON utilizando una aplicación como [JSONLint](http://jsonlint.com/).

JSON Puede tomar la forma de cualquier tipo de datos que sea válido para ser incluido en un JSON, no sólo arreglos u objetos

Una cadena o un número único podrían ser objetos JSON válidos.

En JSON, sólo las cadenas entre comillas pueden ser utilizadas como propiedades.

**Ejercicios:**

1. **Desarrolle ventajas y desventajas de API REST**

**Ventajas:**

1. Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
2. Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
3. La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo

**Desventajas:**

1. Mayor complejidad en la infraestructura: Es necesario diseñar sistemas para trabajar sin estado, lo que requiere gestionar tokens y otros mecanismos de autenticación.
2. Distribución del sistema: En REST, diferentes servidores pueden no estar sincronizados, complicando la gestión de sesiones y datos.
3. Tiempos iniciales de desarrollo más largos: Configurar y desarrollar la API requiere más tiempo al inicio, especialmente si la API debe ser consumida por múltiples sistemas.
4. Rigidez en el desarrollo: La falta de sincronización entre el backend REST y los frontales puede generar retrasos si el cliente necesita funcionalidades que aún no están listas en el servidor.
5. Mayor curva de aprendizaje: Además de lenguajes y librerías habituales, se necesita conocimiento profundo del protocolo HTTP y sus particularidades.
6. Cambio de mentalidad: Requiere abandonar la noción tradicional de tener todos los recursos centralizados en un solo servidor. Esto puede resultar desafiante para los equipos de desarrollo acostumbrados a enfoques tradicionales.
7. **¿En qué casos se utiliza el REST API?**

**Su respuesta**

* **Interfaz uniforme**: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
* **Sistema de capas**: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
* **Uso de hipermedios**: es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos. Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

1. En qué casos en propicio utilizar JSON

* **Transferencia de datos entre frontend y backend:** Por su estructura ligera y fácil de interpretar.
* **APIs RESTful:** JSON se utiliza comúnmente para enviar y recibir datos, mejorando la compatibilidad entre sistemas.
* **Almacenamiento de configuraciones y datos estructurados:** Gracias a su formato sencillo y autodescriptivo.
* **Intercambio entre sistemas heterogéneos:** Muchos lenguajes soportan JSON, lo que facilita su uso en entornos variados.

**PROGRAMACION PARA DISPOSITIVOS MOVILES – Clase 11**

Con un aumento significativo en la cantidad de usuarios de aplicaciones móviles, se están desarrollando más aplicaciones en múltiples plataformas, entre las cuales, Android e iOS son las principales plataformas que dominan la industria de desarrollo de aplicaciones móviles.

Ahora, las empresas quieren ofrecer aplicaciones móviles de este tipo a sus usuarios que puedan ejecutarse en Android, iOS y otros dispositivos Windows. Se ha notado un tremendo aumento en los móviles multiplataforma.

Según las estadísticas publicadas por Dot Com Infoway, se ha observado que alrededor del 83 % de los desarrolladores de aplicaciones móviles utilizan herramientas multiplataforma principalmente para desarrollo, análisis, informes de fallas y pruebas. En este artículo, puede echar un vistazo a las ventajas de Xamarin que lo hacen tan popular.

**Xamarin: ¿Por qué elegirlo?**

Si bien la creación de una aplicación que se ejecuta en varias plataformas es una tarea abrumadora, Xamarin puede facilitarla. Xamarin de Microsoft es una plataforma cruzada de código abierto que permite que los dispositivos móviles funcionen en más dispositivos y proporciona la mejor interfaz de usuario.

La llegada al mercado de diversas aplicaciones basadas en el rendimiento ha endurecido la competencia más que nunca. Por lo tanto, todos los desarrolladores de dispositivos móviles buscan formas económicas de desarrollar aplicaciones ricas en funciones. Puede contratar desarrolladores en la India para crear una aplicación con Xamarin. En este artículo, proporcionamos información detallada sobre las ventajas de Xamarin que la convierten en la mejor plataforma para el desarrollo de aplicaciones híbridas.

**¿Qué es Xamarin?**

Basado en la pila de tecnología de Microsoft, Xamarin es una plataforma utilizada para el desarrollo de aplicaciones híbridas. Esta nueva herramienta desarrollada para el desarrollo de aplicaciones móviles multiplataforma es utilizada por más de 1,4 millones de desarrolladores. Xamarin es un marco apto para crear e implementar aplicaciones móviles de alto rendimiento. Puede leer más para obtener una breve mirada a las estadísticas y tendencias actuales de este marco popular. Después de leer este artículo, conocerá los principales beneficios de Xamarin que ofrece a los desarrolladores.

**¿Cuándo se descubrió Xamarin?**

Xamarin fue creado por los desarrolladores que crearon Mono, que es una plataforma de desarrollo de código abierto basada en .NET Framework. Dirigido por Miguel de Icaza, el equipo presentó por primera vez Xamarin en 2001. Más tarde, la empresa Xamarin se fundó el 16 de mayo de 2011. Xamarin pronto se convirtió en un popular producto multiplataforma de desarrollo de aplicaciones móviles híbridas para desarrollar aplicaciones móviles bien adaptadas que pueden ejecutar en múltiples plataformas. Puede leer más este artículo, que proporciona información detallada sobre los principales beneficios de Xamarin.

**¿Cuáles son las características destacadas de Xamarin?**

En los últimos años, Xamarin se ha convertido en un líder multiplataforma y parece ser la mejor opción para desarrollar aplicaciones móviles híbridas. Tanto los desarrolladores como las empresas creen que Xamarin ofrece la mejor plataforma para crear aplicaciones móviles híbridas. Si desea crear aplicaciones móviles multiplataforma que puedan funcionar bien en múltiples plataformas, puede contratar desarrolladores de aplicaciones disponibles en la industria.

Xamarin proporciona una experiencia de usuario nativa y alrededor del 95 % de reutilización del código. Xamarin es ampliamente utilizado por un gran número de desarrolladores. Hay alrededor de 70,00,000 desarrolladores de aplicaciones móviles que crean aplicaciones móviles con Xamarin. Al leer este artículo, puede explorar más para conocer las ventajas de Xamarin que lo convierten en una opción popular para el desarrollo de aplicaciones híbridas.

**1. Lenguaje C#**

Xamarin utiliza un único lenguaje, C#, para crear aplicaciones personalizadas y de alto rendimiento para todas las plataformas móviles. Xamarin se compila de forma nativa a diferencia de otras soluciones interpretadas, como Appcelerator Titanium. Esto convierte a Xamarin en una opción lista para usar para crear aplicaciones móviles interactivas que pueden ejecutarse en varias plataformas.

**2. Basado en el marco .NET**

Xamarin usa C# y bibliotecas nativas que están encerradas en la capa .Net para crear aplicaciones móviles híbridas muy atractivas. C# ofrece muchas características útiles de .NET como LINQ, Lambdas y programación asíncrona (async/wait) que se pueden usar para crear aplicaciones. Además, Xamarin puede usar el acceso API nativo y más reciente para utilizar las capacidades de la plataforma integrada, como Android Multi-Window o ARKit en iOS.

**3. Compilación**

Xamarin tiene dos plataformas principales que incluyen Xamarin. Android y Xamarin.iOS. En la plataforma Xamarin.Android, la aplicación Android Xamarin se compila primero en el lenguaje intermedio y luego en el código ensamblador original en tiempo de ejecución (compilación Just-in-Time), mientras que en el caso de las plataformas Xamarin.iOS, el código fuente es compilado directamente en el código ensamblador ARM original (compilación anticipada). Ambos procesos están automatizados y pueden solucionar problemas complicados de manera eficiente.

**4. Xamarin.Forms**

Xamarin.Forms es otra característica importante de Xamarin que proporciona las herramientas y bibliotecas integradas en la plataforma de desarrollo .NET para crear aplicaciones móviles de alta calidad. Permite a los desarrolladores desarrollar aplicaciones para iOS, Android y Windows a partir de un único código base compartido.

**5. Xamarin. Mac**

Con Xamarin.Mac, los desarrolladores pueden crear aplicaciones para Mac OS. Xamarin.Mac ayuda en el desarrollo de aplicaciones Mac 100 % nativas en C# y .NET utilizando las API de macOS. Mac es uno de los principales beneficios de Xamarin que lo convierte en una opción popular entre los desarrolladores de aplicaciones móviles para desarrollar aplicaciones móviles híbridas.

**6. SDK de plataforma**

Otra característica destacada de Xamarin que la convierte en la mejor plataforma para el desarrollo de aplicaciones híbridas es que los desarrolladores pueden aprovechar los beneficios de usar los SDK de la plataforma eligiendo Xamarin para crear aplicaciones móviles multiplataforma. Microsoft Xamarin ofrece a sus usuarios acceso a SDK específicos de la plataforma (Cocoa Touch para iOS y Android SDK) a través de una sintaxis simple de C#.

**7. Enlaces de código específicos de la plataforma**

Al usar Xamarin, los desarrolladores podrán crear aplicaciones móviles personalizadas mientras conservan la apariencia nativa de sus respectivas plataformas. Aún así, los ingenieros pueden llamar al código de plataforma inculcado para ejecutar algunas funciones específicas de la plataforma, como PayPal, la integración de Google Analytics y el escaneo de códigos de barras.

**8. Soporte de Wearables**

Xamarin ayuda a crear aplicaciones para dispositivos de reloj Apple y Android. Mediante el uso de Visual Studio, los codificadores pueden crear aplicaciones para iOS Apple Watch tanto para Mac como para Windows. Otra característica que convierte a Xamarin en la mejor plataforma para desarrollar aplicaciones móviles multiplataforma es su capacidad para admitir completamente Android Wear.

**9. Estructura del proyecto Xamarin**

Otra característica única de Xamarin que la convierte en la mejor plataforma para el desarrollo de aplicaciones híbridas es su estructura de proyecto. Todo el trabajo relacionado con Xamarin se realiza con Visual Studio y Xamarin. Las aplicaciones móviles multiplataforma creadas con Xamarin se pueden depurar directamente desde los dispositivos.

**¿Qué hace que Xamarin sea la mejor opción para el desarrollo de aplicaciones híbridas?**

Xamarin permite a los codificadores de aplicaciones móviles crear aplicaciones móviles multiplataforma y aplicaciones nativas para las principales plataformas como iOS y Android. Esta plataforma es simple y fácil de usar. Microsoft Xamarin ofrece muchas opciones para brindar la ayuda adecuada para resolver los problemas que enfrentan los desarrolladores al desarrollar sus aplicaciones móviles. Siga leyendo para conocer las ventajas de Xamarin y saber por qué Xamarin es la mejor plataforma de desarrollo de aplicaciones móviles híbridas.

Puede ofrecer la mejor experiencia a sus usuarios proporcionando un control de interfaz de usuario estandarizado con Xamarin.

En este blog, enumeraremos y especificaremos los beneficios de Xamarin que le permitirán saber por qué Xamarin es una de las mejores opciones para desarrollar aplicaciones móviles híbridas. Puede contratar desarrolladores de Android para crear aplicaciones móviles muy atractivas que satisfagan las necesidades de su empresa.

**1. Rendimiento nativo**

Xamarin ofrece un gran rendimiento a las aplicaciones que igualan el funcionamiento de las aplicaciones en sus plataformas nativas. Las aplicaciones desarrolladas con Xamarin funcionan en el hardware, a diferencia de otras aplicaciones de Ionic que funcionan en las abstracciones. Las aplicaciones aprovechan los beneficios de trabajar en hardware específico de la plataforma y están compiladas para un rendimiento nativo.

**2. Código reutilizable**

La reutilización del código es otra característica clave que ofrece Xamarin que ayuda en el desarrollo de aplicaciones móviles híbridas. Xamarin ofrece diferentes opciones para compartir código, como bibliotecas de clases portátiles (PCL), proyectos compartidos y bibliotecas .Net. El mismo código base se puede usar en otras plataformas mediante las funciones de Xamarin.Forms.

**3. Idioma único y múltiples plataformas**

El uso de Xamarin ayuda a desarrollar aplicaciones híbridas para múltiples plataformas mediante el uso de un lenguaje común. Los ingenieros que tienen conocimientos básicos de C# y .NET Framework pueden crear varias aplicaciones para Android, iOS y Windows. Esto es ventajoso para los líderes y gerentes de equipo, ya que reduce la complejidad y facilita la gestión del desarrollo general del producto, lo que ahorra tiempo y costos.

**4. Experiencia de usuario nativo**

La experiencia web que se ofrece a los usuarios finales juega un papel importante en el aumento de la participación de los usuarios. Xamarin brinda una experiencia de usuario nativa a los usuarios de la aplicación. El uso de Xamarin conserva la apariencia nativa de toda la aplicación, lo que permite a los usuarios acceder a las mismas funciones que proporciona la plataforma nativa.

**5. 100 % de cobertura de API en iOS y Android**

Esta es otra característica que convierte a Xamarin en la mejor opción para el desarrollo de aplicaciones híbridas. Xamarin permite a los desarrolladores de Xamarin crear el 100 % de las API proporcionadas por las plataformas iOS y Android tan pronto como estén disponibles en sus bases de código nativas. Esto significa que los desarrolladores de aplicaciones móviles pueden usar estas API una vez que se publiquen en sus plataformas nativas.

**6. Garantía de calidad perfecta**

Xamarin fomenta el desarrollo de aplicaciones móviles basadas en el rendimiento que pueden funcionar en muchas plataformas. Las aplicaciones móviles creadas con Xamarin deben probarse para asegurarse de que funcionan según los requisitos deseados. Xamarin permite al probador realizar varias pruebas para verificar la funcionalidad de estas aplicaciones móviles en varios dispositivos. Con la ayuda de Xamarin Test Cloud, los evaluadores pueden ejecutar fácilmente sus pruebas para garantizar que sus aplicaciones funcionen perfectamente bien.

**7. Entorno de desarrollo integrado de primera clase**

Xamarin ofrece una experiencia de desarrollador inigualable a los usuarios al proporcionar un espléndido entorno de desarrollo integrado. El excelente entorno de desarrollo Visual Studio que ofrece Xamarin permite a los desarrolladores desarrollar fácilmente los códigos con facilidad. Xamarin proporciona 'Visual Studio Team Services' (VSTS), que se compone de las herramientas que se combinan con Visual Studio, que incluyen TFS para el control del código fuente, la integración continua y las pruebas en la nube con Xamarin Test Cloud.

**8. Lanzamientos mejor coordinados en Android e iOS**

El mismo equipo de desarrollo participa en el desarrollo de aplicaciones móviles multiplataforma utilizando Xamarin. Las características y correcciones para estas dos plataformas están disponibles para ambas al mismo tiempo. Esto reduce los esfuerzos del gerente para controlar la funcionalidad de las dos aplicaciones en diferentes plataformas.

**9. Bien administrado y enfocado**

Xamarin, propiedad de Microsoft, está bien administrado y ofrece una gran asistencia en el desarrollo de aplicaciones móviles híbridas impulsadas por el rendimiento. Además, el kit de desarrollo de software (SDK) de Xamarin es de código abierto, lo que permite a los desarrolladores de aplicaciones móviles explorar más para obtener información detallada sobre cómo funciona Xamarin.

**10. Desarrollo más rápido**

Xamarin permite a los usuarios desarrollar aplicaciones móviles a un ritmo más rápido. Dado que el 75 por ciento del código debe escribirse solo una vez, se ahorra mucho tiempo y dinero. Esto fomenta un desarrollo rápido, lo que permite a los ingenieros de aplicaciones móviles codificar toda la lógica de la aplicación solo una vez y luego compartirla en varias plataformas.

**11. Menos errores**

C# es un lenguaje orientado a objetos simple y con seguridad de tipos. Por lo tanto, es fácil desarrollar aplicaciones móviles con Xamarin, ya que hay menos fallas y errores. Xamarin admite la creación de aplicaciones móviles de alto rendimiento con un comportamiento impecable del programa, lo que permite ahorrar tiempo y dinero. Dado que la mayor parte del código se comparte en Xamarin, los códigos que se utilizan no son largos para todas las plataformas. Los códigos pequeños se usan en Xamarin, por lo que hay menos posibilidades de errores.

**Ejercicios:**

1. Desarrolle brevemente Xamarin

**Xamarin** es un entorno de desarrollo para aplicaciones móviles multiplataforma creado por **Microsoft**. Permite crear apps nativas para **Android**, **iOS**, **Windows** y **macOS** utilizando **C#** como lenguaje principal junto con el marco de trabajo **.NET**. Esta solución destaca por simplificar la creación de aplicaciones con **experiencia de usuario nativa** y alto desempeño, maximizando la reutilización del código hasta en un **95 %** entre diferentes plataformas.

1. **¿Cómo se utiliza Xamarin para compilar para todas las plataformas mobile?**

**Xamarin** se divide en dos plataformas principales: **Xamarin.Android** y **Xamarin.iOS**. En **Xamarin.Android**, las aplicaciones pasan por una **compilación JIT (Just-in-Time)**, donde el código se convierte primero a un lenguaje intermedio y luego a ensamblador en tiempo de ejecución. Por otro lado, en **Xamarin.iOS**, el código se convierte directamente en **código nativo ARM** mediante **compilación anticipada (AOT)**. Ambos procesos están automatizados y optimizan la solución de problemas complejos durante el desarrollo.

1. **Desarrolle las características añadidas**

**Características Principales de Xamarin:**

* **Xamarin.Forms:** Facilita la creación de interfaces unificadas en **XAML** o **C#** que funcionan en Android, iOS y Windows a partir de un único proyecto.
* **Xamarin.Android y Xamarin.iOS:** Proveen acceso directo a las APIs específicas de cada sistema, utilizando JIT en Android y AOT en iOS para mejorar el desempeño.
* **Xamarin.Mac:** Permite desarrollar aplicaciones nativas para **macOS** con C# y .NET.
* **Acceso a SDKs Nativos:** Habilita el uso de las APIs más recientes, como **ARKit** en iOS o **multitarea** en Android.
* **Integración con Visual Studio:** Ofrece un entorno de desarrollo integral que incluye control de versiones, pruebas y despliegue continuo.
* **Xamarin Test Cloud:** Plataforma de pruebas automatizadas en dispositivos reales, garantizando un comportamiento óptimo de las apps.
* **Soporte para Wearables:** Permite desarrollar para **Apple Watch** y **Android Wear**, ampliando las posibilidades de los proyectos.
* **Reutilización de Código:** Hasta un 95 % del código puede compartirse entre plataformas, lo que facilita la gestión y reduce costos.
* **Experiencia de Usuario Nativa:** Las aplicaciones conservan la estética y funcionalidades propias de cada sistema operativo.
* **Lanzamientos Simultáneos:** Posibilita publicar las aplicaciones en **Android** e **iOS** al mismo tiempo.
* **Plataforma Segura y de Código Abierto:** Respaldada por Microsoft, ofrece una infraestructura sólida y escalable.

1. **¿Para qué sirve Xamarin?**

**Xamarin** es un entorno ideal para crear aplicaciones móviles multiplataforma, compatible con **Android**, **iOS**, **Windows** y **macOS** utilizando **C#** y **.NET**. Su enfoque en la reutilización de hasta el **95 % del código** reduce tiempos y costos, asegurando un **rendimiento nativo** en cada plataforma. Xamarin brinda acceso directo a las APIs específicas del sistema operativo, habilitando el desarrollo de aplicaciones para wearables como **Apple Watch** y **Android Wear**. Gracias a su integración con **Visual Studio**, se simplifica el proceso de pruebas, control de versiones y despliegue continuo, posicionándolo como una opción eficiente para empresas y desarrolladores que buscan soluciones escalables y de alto rendimiento.

**Pruebas – Clase 12**

Los dispositivos reales, por un lado, tienen un gran valor y beneficios, pero también tienen su propio conjunto de costos. Los simuladores y emuladores móviles también son beneficiosos y pueden brindar un valor único tanto a los desarrolladores como a los probadores, pero también tienen inconvenientes.

**¿Qué es un dispositivo real en las pruebas móviles?**

Los dispositivos reales en las pruebas móviles le permiten probar su aplicación en un dispositivo real. Esto es útil para determinar si la aplicación funcionará en las manos de un cliente.

**¿Qué es un emulador en pruebas móviles?**

Un emulador en pruebas móviles es un dispositivo virtual. Le permite probar su aplicación emulando un dispositivo real. Un emulador de dispositivo imita el hardware o el sistema operativo del dispositivo.

**¿Qué es un simulador en pruebas móviles?**

Un simulador en pruebas móviles también es un dispositivo virtual. Le permite probar su aplicación simulando el comportamiento de un dispositivo real. Un simulador imita el comportamiento básico de un dispositivo.

**Dispositivos reales frente a emuladores frente a simuladores**

El debate de las pruebas móviles en dispositivos reales frente a emuladores frente a simuladores es uno de los debates más antiguos y emotivos en el espacio móvil.

**Diferencia entre simuladores y emuladores**

Un emulador es una aplicación de escritorio que virtualiza por completo todos los aspectos de un dispositivo real, incluido el hardware y el sistema operativo. Los simuladores no imitan el hardware o el sistema operativo. Permiten al usuario validar flujos de aplicaciones, pero no en un verdadero entorno de producción.

**Cuándo probar en un simulador frente a un emulador**

Los simuladores y emuladores son ideales para realizar pruebas en las primeras etapas del ciclo de desarrollo. Los emuladores son mejores para probar el comportamiento externo y los simuladores para el comportamiento interno. Por ejemplo, use emuladores para probar cálculos y transacciones, y simuladores para pruebas internas de hardware y firmware.

**Cuándo probar en un dispositivo real**

Las pruebas de dispositivos reales son las más confiables y las mejores para las pruebas de usabilidad y rendimiento. También puede aplicar condiciones de usuario reales para realizar pruebas más realistas.

**Ventajas de los Simuladores y Emuladores**

Tanto los simuladores como los emuladores son dispositivos virtuales. Ambas soluciones virtualizan dispositivos móviles y ofrecen una serie de ventajas para probar aplicaciones móviles .

**1. Variedad**

Los simuladores y emuladores pueden virtualizar una gran variedad de dispositivos y permutaciones de sistemas operativos. Esta capacidad permite a los usuarios validar muy fácilmente en múltiples plataformas, así como en casos que requieren combinaciones específicas de dispositivo/SO. Además, hay plataformas que solo son compatibles (para fines de desarrollo y prueba), a partir de ahora, a través de dispositivos virtuales.

**2. Precio**

Las soluciones de dispositivos virtuales son más baratas que los dispositivos reales por un amplio margen. Esto se aplica tanto a las soluciones locales como a las basadas en la nube . La necesidad de impulsar las pruebas al principio del proceso de desarrollo requiere que los equipos escalen y prueben más.

En organizaciones grandes, si cada desarrollador quiere ejecutar un conjunto de validaciones previas al compromiso, esto requiere una gran cantidad de dispositivos para ejecutar. La economía de correr contra dispositivos reales puede no tener sentido a tal escala cuando los dispositivos virtuales permitirán que la organización escale y ejecute antes en el proceso.

**3. A partir de una línea de base**

Con los dispositivos virtuales, siempre puede comenzar desde el mismo estado del dispositivo. Lograr esto con dispositivos reales puede requerir un restablecimiento de fábrica, lo que puede llevar mucho tiempo y esfuerzo. El hecho de que el dispositivo virtual siempre comience desde el mismo punto ayuda. En muchos casos, aumenta la confiabilidad de la ejecución de pruebas automatizadas ya que no hay sorpresas. El dispositivo siempre está listo para la ejecución en la forma en que se diseñaron las pruebas. Por ejemplo, el dispositivo no será bloqueado por otro usuario.

**Principales casos de uso de simuladores y emuladores móviles**

**Desarrollo Local y Validación**

Los desarrolladores y probadores usan simuladores y emuladores en sus máquinas locales para el desarrollo, la depuración de aplicaciones y las validaciones locales. Los IDE más comunes para aplicaciones nativas vienen con herramientas de dispositivos virtuales como parte de la instalación básica.

Xcode viene con la funcionalidad del simulador, mientras que Android Studio viene con un emulador. Ambos se volvieron estables y maduros en los últimos años y cada uno tiene una gran variedad de capacidades para validaciones avanzadas.

**Pruebas de integración continua**

El principal caso de uso de los laboratorios de dispositivos virtuales es para las pruebas de integración continua (CI). La mayor adopción de metodologías DevOps y Agile impulsa a los equipos a realizar más pruebas en las primeras etapas del proceso de desarrollo, también conocidas como pruebas de cambio a la izquierda . Los nuevos marcos de automatización de pruebas que están más alineados con las habilidades y herramientas de los desarrolladores, como Espresso y XCUITest , ayudan a los equipos de desarrollo a aumentar su cobertura de automatización de pruebas.

Sin embargo, para ejecutar estas pruebas, necesita un entorno de ejecución adecuado: una nube de prueba contra la que ejecutar. Estas pruebas automatizadas pueden ejecutarse en la fase previa a la confirmación, ejecutando un conjunto de pruebas para una validación rápida antes de confirmar o fusionar el código, o activarse a través del CI varias veces al día, proporcionando un valor rápido a los equipos de desarrollo en los cambios de código recientes.

**4 ventajas de usar un dispositivo móvil real**

**1. No hay nada como lo real**

Si bien los dispositivos virtuales son buenos para las validaciones básicas, para validar realmente que una aplicación funciona, debe ejecutar pruebas en dispositivos reales. Si bien los dispositivos virtuales pueden ser muy útiles para validar flujos funcionales, puede haber casos de falsos positivos. Esto significa que, si bien las pruebas pueden pasar, en realidad, puede haber problemas con la aplicación.

Esto se debe a que los simuladores y emuladores realizan pruebas en el "camino feliz", el flujo principal de la aplicación, en lugar de tratar de ver cómo la aplicación maneja los casos extremos y negativos.

**2. Mejores validaciones de la interfaz de usuario (UI)**

La validación de la interfaz de usuario debe realizarse en dispositivos reales para validar la precisión de la interfaz de usuario. Además, los problemas de usabilidad son muy fáciles de encontrar mientras se trabaja en un dispositivo real, a diferencia de los dispositivos virtuales.

En la mayoría de los casos, cuando es necesario ingresar datos desde el teclado, los dispositivos reales se superponen a la aplicación, a diferencia de los dispositivos virtuales, donde el teclado se presenta junto a la interfaz del dispositivo.

**3. Pruebas de rendimiento más precisas**

Los dispositivos reales brindan mediciones de rendimiento más precisas y confiables en los tiempos de transacción. Además de las implicaciones que tiene el hardware específico en el rendimiento, los dispositivos virtuales también representan la interfaz de usuario de manera diferente.

**4. Validaciones mejoradas relacionadas con el sensor y el hardware**

Los casos de uso comunes que se pueden virtualizar son aquellos que requieren interacción con el hardware y los sensores del dispositivo, como la cámara, el acelerómetro y la biometría. En algunos casos, el comportamiento de los dispositivos reales y virtuales puede diferir.

**Equilibrio entre simuladores, emuladores y dispositivos reales**

Dado que las actividades de prueba ocurren desde las primeras etapas de desarrollo hasta la implementación y el monitoreo de la aplicación, existe la necesidad de ambos tipos de plataformas, reales y virtuales.

**Fase de código**

**Validación local**

Los desarrolladores deben validar el código durante la fase de desarrollo, ya sea con un dispositivo local o un dispositivo virtual, que en la mayoría de los casos forman parte del IDE del desarrollador. En el caso de la validación de la IU, se recomienda validar la nueva IU en dispositivos reales en lugar de virtuales para asegurarse de que el resultado sea el esperado.

**Validación previa a la confirmación**

La ejecución en pruebas unitarias y unitarias de la interfaz de usuario debe ejecutarse, en la mayoría de los casos, en dispositivos virtuales (90 %). Además, hay casos en los que los desarrolladores realizan cambios en componentes que, según su conocimiento y experiencia, se comportarán de manera diferente en dispositivos reales. En esos casos, se recomienda que brinde a los desarrolladores herramientas bajo demanda para seleccionar qué pruebas quieren ejecutar y en qué plataforma ejecutar, ya sea virtual o real.

**Fase de construcción**

Dependiendo de la madurez del equipo, los equipos ejecutarán el proceso de CI una o más veces al día. El objetivo es crear una combinación que permita la escalabilidad al mismo tiempo que proporciona una buena cobertura de prueba e información sobre la experiencia del usuario final. Para permitir la escala de ejecución y la validación paralela para múltiples desarrolladores y equipos al mismo tiempo, una combinación típica de CI podría tener este aspecto.

**Confirmar trabajo**

Esto se ejecutará para cada compromiso que realice un desarrollador, ejecutando una verificación de cordura básica para asegurarse de que el cambio no rompa la compilación o introduzca una regresión significativa. Debido a la necesidad de ejecutar este trabajo a escala en cada confirmación, estas pruebas deben ejecutarse en un dispositivo virtual.

**Commit job**

Estos trabajos validan los últimos X compromisos y prueban si uno de ellos rompió la compilación o introdujo una regresión. Por lo general, este tipo de trabajo se ejecutará cada pocas horas/X cantidad de confirmaciones de código. Este trabajo ejecuta más pruebas para proporcionar una mayor cobertura. Aquí, comenzamos a introducir dispositivos reales en la CI: la combinación debe ser 70 % de dispositivos virtuales y 30 % de dispositivos reales.

**Night job**

Este trabajo ejecuta pruebas de regresión y amplía la cobertura. En este caso, las pruebas se ejecutarán en dispositivos reales.

**Fase de prueba**

Las actividades de prueba adicionales que se ejecutan fuera del proceso de CI se ejecutarán en dispositivos reales. Aquí, el objetivo es validar la experiencia completa del usuario final, la cobertura de flujo funcional, la validación de la interfaz de usuario y las pruebas de rendimiento. Todas las pruebas (100%) deben ejecutarse en dispositivos reales.

**Fase de Monitoreo**

Si bien hay organizaciones que tienen soluciones de monitoreo de usuarios reales (RUM) que se ejecutan en dispositivos reales, también hay valor en el monitoreo sintético: ejecución de pruebas automatizadas de un solo uso que miden el rendimiento de la aplicación que se ejecuta cada 10-15 minutos. En este caso, para obtener los resultados más precisos, la mejor práctica es usar dispositivos reales.

**Ejemplo: dispositivos reales frente a emuladores**

**Un estudio de caso de LinkedIn**

LinkedIn  (adquirida por Microsoft) ahora brinda servicios a más de 500 millones de usuarios en múltiples plataformas, como la web y los dispositivos móviles.

LinkedIn solía sufrir de mala calidad, problemas de estabilidad y malas críticas. Mientras intentaba resolver los problemas de calidad, además de seguir creciendo, LinkedIn anunció el proyecto  Voyager . Este es un nuevo modelo de lanzamiento y estrategia SDLC que tiene como objetivo mejorar tanto la calidad como la velocidad de lanzamiento para la organización.

LinkedIn cambió a una admirable  estrategia de lanzamiento de 3x3 . Ahora pueden enviar una nueva versión a producción 3 veces al día, cada 3 horas a partir de las confirmaciones de código del desarrollador.

**La cadencia de lanzamiento mejoró, pero la experiencia del usuario disminuyó**

La cadencia de lanzamiento mejoró. Esto permitió que el equipo de productos de LinkedIn se adaptara rápidamente a los cambios, corrigiera errores más rápido e innovara. Pero la experiencia del usuario final de alguna manera se redujo. Y muchos de los usuarios de dispositivos móviles lo expresan en las tiendas de aplicaciones, afirmando que preferirían usar el navegador de escritorio en lugar de la aplicación móvil.

LinkedIn ejecutó su plan de prueba para todo el alcance de la prueba en 16 emuladores en paralelo. No hubo cobertura en dispositivos reales.

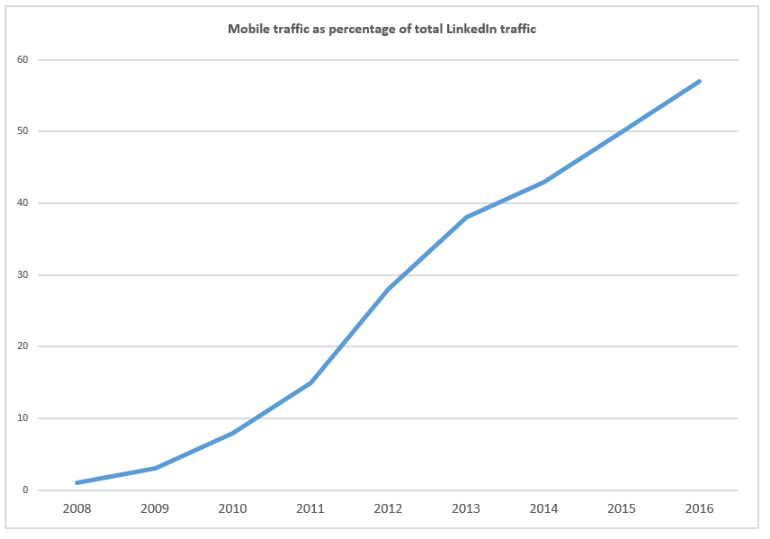
Esto condujo a muchos defectos escapados a la producción que afectan a los usuarios de dispositivos reales.

Esto es lo que sucedió:

* La aplicación falla en los dispositivos Android cuando se cambia de Wi-Fi a redes de operadores reales.
* Las invitaciones desde el móvil a varias conexiones no funcionaban correctamente.
* Hubo problemas de sincronización entre el feed de LinkedIn y lo que se muestra en la versión del navegador.
* Hubo problemas de instalación en dispositivos reales.
* Y más.

**Su estrategia de prueba debe cambiar**

* La mayoría del tráfico a la aplicación de LinkedIn proviene de dispositivos móviles. Por lo tanto, la estrategia de prueba de LinkedIn debe ajustarse en consecuencia.



LinkedIn necesita basar su estrategia de prueba móvil en personas de la vida real que operan desde diferentes ubicaciones geográficas, condiciones variables, aplicaciones en segundo plano y más.

Las pruebas en emuladores son esenciales y deben mantenerse como parte de la estrategia. Pero no puede ser la única plataforma para probar esta aplicación. No garantiza calidad continua y UX.

**Conclusión**

La clave para implementar pruebas continuas y maximizar la velocidad en el espacio móvil es tener el equilibrio perfecto entre simuladores, emuladores y dispositivos reales. Cada una de estas plataformas extremadamente diferentes aporta valores y beneficios únicos a los desarrolladores y probadores. Sin embargo, estos valores se maximizan cuando se utilizan en la fase correcta del ciclo de vida del desarrollo.

Asegúrese de planificar adecuadamente la cobertura de prueba, la plataforma bajo prueba y las herramientas de prueba a lo largo de sus actividades de prueba continuas y monitoree continuamente el espacio móvil ya que constantemente se lanzan al mercado nuevos dispositivos, así como nuevas versiones de simuladores y emuladores.

**Ejercicios:**

1. ¿Qué ejemplos de emuladores conoces?

**Ejemplos de emuladores**:

* **Android Studio Emulator**: Emula dispositivos Android para probar aplicaciones móviles.
* **Xcode Simulator**: Permite emular dispositivos iOS en Mac.
* **Bluestacks**: Emulador de Android orientado a aplicaciones de juegos.
* **Genymotion**: Emulador Android con enfoque en desarrolladores y testers.
* **NoxPlayer**: Otro emulador para correr aplicaciones Android, especialmente juegos.

1. Ventajas y desventajas de utilizar un emulador a la hora de probar.

**Ventajas**:

* **Variedad de dispositivos y configuraciones**: Permite emular múltiples combinaciones de hardware y software sin necesidad de comprar dispositivos reales.
* **Económico**: Mucho más barato que adquirir dispositivos físicos​.
* **Estado controlado**: Los emuladores siempre empiezan en el mismo estado, garantizando que las pruebas sean consistentes.
* **Escalabilidad**: Facilitan la ejecución de pruebas automatizadas a gran escala, especialmente en procesos de integración continua​.

**Desventajas**:

* **Falta de precisión en pruebas de hardware**: No pueden replicar de forma exacta comportamientos de sensores como acelerómetros, cámaras o biometría.
* **Falsos positivos**: Las pruebas realizadas en emuladores pueden mostrar resultados positivos que luego fallan en dispositivos reales.
* **Rendimiento no representativo**: Los tiempos de respuesta o consumo de recursos pueden variar respecto a un dispositivo real​.

1. ¿Cómo es el proceso de deploy en cada plataforma?

**Proceso de deploy en cada plataforma**:

* **Android (Android Studio)**:
  1. Compilar el código en un APK.
  2. Subir el APK a Google Play Store.
  3. Google Play realiza la validación y finalmente lo pone a disposición de los usuarios.
* **iOS (Xcode)**:
  1. Compilar el código en un archivo IPA.
  2. Subir la aplicación al App Store mediante el portal de Apple.
  3. Apple valida y aprueba la aplicación antes de publicarla​.

1. ¿Cuál es la diferencia entre emuladores de, por ejemplo, Android Studio y Xcode contra otros emuladores?

**Emuladores integrados (Android Studio, Xcode)**:

* Ofrecen herramientas específicas para desarrollo y depuración avanzada, facilitando la integración con los entornos de desarrollo.
* Están diseñados para trabajar de manera nativa con sus respectivas plataformas (Android e iOS), garantizando compatibilidad y soporte constante​.

**Otros emuladores (Bluestacks, Genymotion)**:

* Enfocados en pruebas más generales o en ejecución de aplicaciones sin propósitos de desarrollo específico.
* Pueden no ofrecer la misma precisión en la simulación de comportamientos internos o integración con frameworks de desarrollo, pero son útiles para probar la aplicación desde la perspectiva del usuario final.

**AGENTES DE SOFTWARE – Clase 13**

Antes de intentar definir qué es un Agente de software, primero intentemos comprender el significado y las características de un agente. Todos estamos, en un sentido u otro, familiarizados con el concepto de agente. Probablemente la mayoría de nosotros hemos tratado con agentes de viajes y conocemos el papel que desempeñan. El principal papel representativo de una agencia de viajes, por ejemplo, es que *actúa en nombre de otros* . Esta característica puede considerarse como la primera propiedad fundamental de la agencia. Un agente de viajes actúa en nombre de un viajero en un grado variable de autonomía. Es decir, cuando un agente de viajes llama a una aerolínea para reservar un asiento, lo hace de forma autónoma; es decir, no le dicen a la aerolínea para quién necesitan un asiento, solo dicen que necesitamos un asiento. Esta es la segunda característica de la agencia: la *autonomía* . Una tercera característica de un agente es el grado de *proactividad* y *reactividad* en su comportamiento. Por ejemplo, una vez que un agente recibe los detalles de sus tareas, trata proactivamente de alcanzar las metas definidas por las tareas asignadas. Y reacciona a los cambios en los datos disponibles modificando su plan. Los agentes también pueden exhibir otros atributos, que incluyen: *aprendizaje* , *cooperación* y *movilidad* .

**¿Qué es un Agente?**

Con base en la discusión anterior, un agente puede definirse, junto con sus características, de la siguiente manera:

*Un agente es una entidad que:*

* actúa en nombre de otros de manera autónoma
* realiza sus acciones en cierto nivel de proactividad y reactividad
* exhibe algunos niveles de los atributos clave de aprendizaje, cooperación y movilidad.

Esta definición, que se basa en la definición dada en es igualmente cierta para los agentes de software. Pero, ¿qué es exactamente un agente de software y en qué se diferencia de un objeto de software? Puede pensar en un agente de software como uno (o más) objetos de software que se ajustan a las características anteriores de los agentes y pueden describirse como inhibidores de computadoras y redes, ayudando a los usuarios con tareas basadas en computadoras. Sin embargo, es responsabilidad del programador determinar qué puede hacer un agente, así como la información requerida del usuario o del software para que un agente realice sus acciones de manera reactiva. El comportamiento del agente puede ser configurado por otro software, que puede considerar como una especie de súper agente , que bifurca (o clona) nuevos agentes cuando una tarea requiere ayuda adicional.

**Clasificación de Agentes de Software**

La clasificación se refiere al estudio de tipos y entidades. Hay varias dimensiones para clasificar los agentes de software existentes. Se pueden clasificar según : las tareas que realizan; su arquitectura de control; el alcance y la eficacia de sus acciones; el rango de sensibilidad de sus sentidos; o cuánto estado interno poseen [6]. En este artículo clasificamos a los agentes, junto con varias características ideales y primarias que deben exhibir los agentes. Identificamos tres características: *autonomía* , *aprendizaje* y *cooperación* , que hemos discutido en la Sección 1. Nuestra clasificación aquí se basa en [4]. Comencemos elaborando un poco más sobre estas características. La autonomía se refiere a la característica de que un agente puede operar por sí mismo sin necesidad de guía humana. En otras palabras, un agente tiene un conjunto de estados y objetivos internos, actúa de tal manera que cumple sus objetivos en nombre del usuario. Para hacerlo, un agente debe ser proactivo en el sentido de que tiene la capacidad de tomar la iniciativa en lugar de actuar simplemente en respuesta a su entorno [15]. La cooperación con otros agentes es necesaria para llevar a cabo una tarea complicada. Sin embargo, para cooperar, un agente debe poseer una habilidad social que le permita interactuar con otros agentes. Finalmente, un agente es inteligente si es capaz de aprender y sentir como actúa y reacciona a su entorno externo. Estas tres características de los agentes se utilizan para derivar algunos tipos de agentes para incluirlos en nuestra clasificación, como se muestra en la Figura 1.

**Agentes de interfaz**

Los agentes de interfaz realizan tareas para sus propietarios enfatizando la autonomía y el aprendizaje [16]. Apoyan y brindan asistencia a un usuario que está aprendiendo a usar una aplicación en particular, como una hoja de cálculo. El agente aquí observa las acciones que lleva a cabo el usuario y trata de aprender nuevos atajos, luego intentará sugerir mejores formas de hacer la misma tarea. Como señala P. Maes , la metáfora clave que subyace a los agentes de interfaz es la de un asistente personal que colabora con el usuario en el mismo entorno de trabajo. Los agentes de interfaz aprenden a ayudar mejor a sus usuarios de cuatro maneras [8]:

* Observando e imitando al usuario
* Mediante la recepción de comentarios positivos y negativos del usuario.
* Al recibir instrucciones explícitas del usuario
* Pidiendo consejo a otros agentes

Si bien los agentes de interfaz piden consejo a otros agentes (aprender de sus compañeros), su cooperación con otros agentes , sin embargo, es limitada.

El objetivo del trabajo de P. Maes sobre los agentes de interfaz fue inicialmente migrar de la metáfora de la manipulación directa a una que delega algunas de las tareas a los agentes de interfaz de software para adaptarse a los usuarios novatos. Con el crecimiento explosivo de la Web, ya medida que nos acercamos al siglo XX, tenemos tantos usuarios de computadoras sin capacitación como microprocesadores. Por lo tanto, hay una brecha que se vuelve más evidente a medida que avanzamos con las computadoras. Las interfaces informáticas actuales , por ejemplo, no hacen nada a menos que des órdenes desde el teclado, el ratón o la pantalla táctil. Por lo tanto, la computadora es simplemente una entidad pasiva que espera sus comandos para ejecutarlos. No nos proporciona ninguna ayuda cuando deseamos realizar algunas tareas complejas. En el futuro, los agentes de interfaz, así como los agentes de información (como se analiza más adelante), pueden conocer los intereses de los usuarios y actuar de forma autónoma en su nombre. Eso es lo que los investigadores en el área de la tecnología de agentes tienen grandes esperanzas.

**Agentes Colaborativos**

Como la proliferación de las redes informáticas de comunicación supuso un gran paso hacia el desarrollo de las “sociedades virtuales”. La colaboración entre individuos (en la sociedad virtual) requiere que los vínculos de comunicación se establezcan y utilicen de manera efectiva. La Inteligencia Artificial Distribuida, que es un subcampo de la Inteligencia Artificial, se ocupa de una sociedad virtual de solucionadores de problemas (agentes) que interactúan para resolver un problema común.

El objetivo de los agentes colaborativos es interconectar agentes colaborativos desarrollados por separado, permitiendo así que el conjunto funcione más allá de las capacidades de cualquiera de sus miembros. Implementar formas eficientes de cooperación entre agentes es en realidad uno de los temas centrales para el desarrollo de Sistemas Multi-Agente.

Una de las motivaciones para tener agentes colaborativos es proporcionar soluciones a problemas inherentemente distribuidos, como la red de sensores distribuidos o el control del tráfico aéreo .

**Agentes de Información**

El crecimiento explosivo de la información en Word-Wide Web ha dado lugar a agentes de información (también conocidos como agentes de Internet) con la esperanza de que estos agentes puedan ayudarnos a administrar, manipular o recopilar información de muchos recursos distribuidos [7 ]. Sin embargo, uno puede notar que los agentes de información parecen un poco similares a los agentes de interfaz [16]. Sin embargo, es importante tener en cuenta que no todos los tipos de agentes discutidos aquí comenzaron al mismo tiempo. Entonces, con el crecimiento explosivo de la información y la necesidad de herramientas para administrar dicha información, uno esperaría cierto grado de superposición entre los objetivos de algunos agentes. Sin embargo, una distinción entre los agentes de interfaz y los de información es que los agentes de información se definen por *lo que hacen* , en contraste con los agentes de interfaz que se definen por *lo que son* .

Hay algunos agentes de interfaz, desarrollados en MIT Media Lab, que han sido desplegados para roles basados en web y, por lo tanto, pueden considerarse agentes de información. Una característica importante de los Agentes de Información es que pueden ser estáticos o móviles.

Los agentes de información son más útiles en la Web, donde pueden ayudarnos con tareas mundanas. Por ejemplo, llevamos a cabo acciones que pueden consumir mucho tiempo ( por ejemplo , buscar información en la Web). ¿Por qué la computadora ( por ejemplo , un agente de información) no lleva a cabo tales tareas por nosotros y luego nos presenta los resultados?

**Agentes reactivos**

Los Agentes Reactivos actúan y responden de manera estímulo-respuesta al estado actual del entorno en el que están incrustados. P. Maes destaca las siguientes tres ideas clave que sustentan los agentes reactivos.

* Funcionalidad emergente: la dinámica de la interacción conduce a la complejidad emergente.
* Descomposición de tareas: un agente reactivo se considera un conjunto de módulos que funcionan de forma autónoma y son responsables de tareas específicas ( p. ej ., detección, computación, etc.).
* Tienden a operar en representaciones que están cerca de los datos de sensores sin procesar.

Hasta ahora, hay un número relativamente pequeño de aplicaciones basadas en agentes de software reactivo. Una buena área de aplicación para ellos parece ser la industria del entretenimiento. Los investigadores de Philips ya están trabajando en video digital y animación de agentes reactivos basada en gráficos 3D.

**Agentes Híbridos**

Los Agentes Híbridos se refieren a aquellos agentes cuya constitución es una combinación de dos o más filosofías de agente dentro de un agente singular. Estas filosofías pueden ser móviles, de interfaz, de información, colaborativas, etc. El objetivo de tener agentes híbridos es la idea de que los beneficios que se obtienen al tener la combinación de filosofías dentro de un solo agente son mayores que las ganancias obtenidas del mismo agente basado en una filosofía singular. Un ejemplo de esto son *los agentes de interfaz colaborativa*.

**NOTA** : El autor cree que los agentes móviles (que se analizan a continuación) en realidad pueden considerarse agentes híbridos, porque para que un agente móvil sea útil, es posible que tenga que ser reactivo, proactivo y colaborativo, por ejemplo.

**Agentes Móviles**

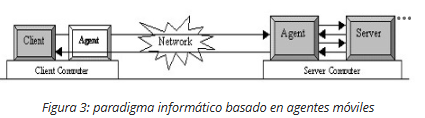
Un agente de software es un agente de software *móvil* si puede migrar de un host a otro para trabajar en un entorno de red heterogéneo. Esto significa que también debemos considerar el entorno de software en el que existen los agentes móviles. Esto se denomina entorno de agente móvil, que es un sistema de software distribuido en una red de computadoras heterogéneas y su tarea principal es proporcionar un entorno en el que puedan ejecutarse los agentes móviles. Tenga en cuenta que no solo un agente se transporta a sí mismo, sino también su estado. Cuando llega al nuevo host, el agente debería poder funcionar correctamente en el nuevo entorno.

**De nuevo Paradigma por Computación Distribuida**

El principio central de la programación distribuida actual es la llamada a procedimiento remoto (RPC). El enfoque RPC, que se concibió en la década de 1970, considera que la comunicación de computadora a computadora permite que una computadora llame a un procedimiento en otra. En RPC, todos los mensajes pasan por la red, cada uno solicita o reconoce las acciones de un procedimiento. Este enfoque, sin embargo, tiene sus propias limitaciones. En particular, todas las interacciones entre el cliente y el servidor deben pasar por la red, como se muestra en la Figura 2.



 Otro enfoque que está formando un nuevo paradigma para la computación distribuida es el que emplea agentes móviles. Inicialmente, este enfoque se conocía como Programación Remota. El enfoque de programación remota considera la comunicación de computadora a computadora como una computadora no solo para llamar a los procedimientos en otra, sino también para proporcionar los procedimientos que se realizarán. Cada mensaje que pasa por la red comprende un procedimiento que debe realizar la computadora receptora y datos que son sus argumentos. El procedimiento y su estado se denominan agente móvil, ya que representan la computadora emisora incluso mientras están en la computadora receptora, como se muestra en la Figura 3.



Este enfoque es atractivo ya que la confiabilidad de la red no es crucial por las siguientes razones:

* Los agentes móviles no consumen mucho ancho de banda de red. Solo consumen ancho de banda cuando se mueven.
* Continúan ejecutándose después de moverse, incluso si pierden la conectividad de red con sus creadores.

Por lo tanto, si un cliente requiere comunicaciones extensas con un servidor en particular en algún lugar de la red, entonces es atractivo implementar un sistema de este tipo utilizando agentes móviles. Esto se debe al hecho de que un agente puede acercarse al servidor remoto, reduciendo el tráfico de la red, realiza todas las tareas y regresa. Durante ese período, la máquina cliente no no tiene que estar encendido. Deberá encenderse solo cuando sea el momento de darle la bienvenida al agente. La figura 3, que se muestra arriba, ilustra la idea de este paradigma.

En este punto, alguien puede decir que esto es exactamente de lo que se trata *la migración de procesos* y esto se ha hecho en los años 60. Ese alguien tendría toda la razón . Sin embargo, los agentes móviles son diferentes en el sentido de que exhiben las características de un agente como se discutió en la Sección 1.

**Aplicaciones de Agentes Móviles**

Un área atractiva para usar agentes móviles es el procesamiento de datos en redes poco confiables. En tales redes, la red de baja confiabilidad se puede usar para transferir agentes, en lugar de una porción de datos, de un lugar a otro. En este paradigma, el agente viaja a los nodos de la red, procesa la información en esos nodos (sin riesgo de desconexión de la red) y luego regresa a casa. Otra área interesante que está atrayendo mucha atención es el comercio electrónico. Por ejemplo, en lugar de gastar una gran cantidad de tiempo en las librerías en línea para encontrar la mejor oferta en un libro, enviar a un agente para que haga esta tarea nos ahorraría una cantidad considerable de tiempo. El agente estaría programado para visitar varias librerías y encontrar las mejores ofertas en los libros que necesitamos.

Otra área interesante de aplicaciones para agentes móviles es la gestión de redes. En los entornos de red heterogéneos de hoy en día, los operadores de red deben tener un amplio conocimiento de las diversas redes para administrarlas. Dicha gestión requiere la recopilación de una gran cantidad de datos de los nodos de la red. Los agentes móviles representan una herramienta ideal para recopilar los datos y analizarlos.

**Ventajas de los agentes móviles**

Los agentes móviles parecen ser útiles para muchas aplicaciones diferentes. Sin embargo, se puede afirmar que prácticamente cualquier tarea que se pueda realizar con agentes móviles se puede realizar con otras tecnologías (por ejemplo, la invocación de métodos remotos). A pesar de que no hay muchos problemas de computación distribuida que no puedan resolverse sin agentes móviles, los agentes móviles facilitan el desarrollo de ciertas aplicaciones y pueden mejorar la confiabilidad y la eficiencia. La Tabla 1, que se basa en contiene algunas de las ventajas reclamadas de los agentes móviles sobre los enfoques convencionales.

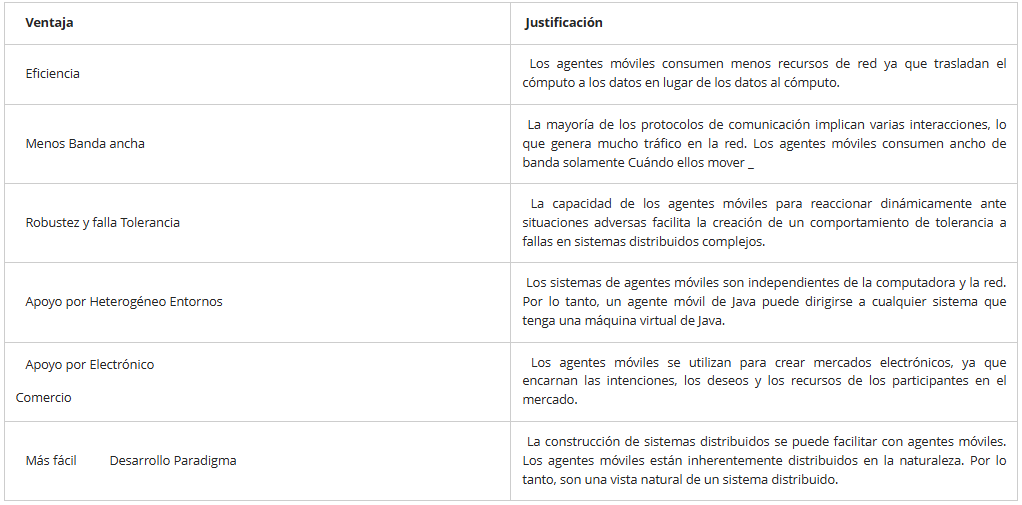


Tabla 1: posibles ventajas de los agentes de software móvil

**Seguridad en Agentes Móviles**

A pesar de que los agentes móviles representan un nuevo paradigma útil para la informática distribuida, rara vez se han utilizado en la práctica. La razón principal de esto se debe al hecho de que existen una serie de desafíos técnicos y sociales relacionados con la implementación de aplicaciones basadas en agentes móviles. Los problemas técnicos incluyen, entre otros, dificultades con la portabilidad y la seguridad. El problema de la seguridad también puede considerarse un desafío social: convencer a las personas de que utilicen agentes de software móvil para comprar artículos a través de Internet. Como señaló B. Laurel, “pocos de nosotros contrataríamos a un agente para presionar los botones de nuestra calculadora; la mayoría de nosotros contrataría a un agente para escanear 5.000 piezas de correo basura”.

El desafío de la seguridad, sin embargo, es complicado. Los agentes móviles plantean problemas similares a los Applets de Java. Hay varias cuestiones de seguridad que deben tenerse en cuenta en la informática basada en agentes móviles. Algunas personas en la comunidad informática piensan que los agentes móviles son virus , ya que pueden exhibir un comportamiento similar.

 La seguridad de los agentes móviles se puede dividir en dos áreas:

* Protección de los nodos host de agentes destructivos.
* Protección de agentes móviles de nodos maliciosos.

 Un enfoque para proteger los nodos del host de los agentes destructivos es a través de la autenticación y las firmas de agentes donde se rechazan todos los agentes desconocidos. Sin embargo, esto realmente no parece una buena solución, porque ¿cómo sabríamos que un agente móvil es quien dice ser?

La otra área de seguridad se ocupa del tema de la protección de los agentes móviles de los hosts que pueden querer escanear al agente en busca de información; alterar el estado del agente; o incluso matar al agente. El problema crucial aquí es que el agente tendrá que exponer sus datos e información al host para poder ejecutarse en él. Sin embargo, la investigación actual ha demostrado que es computacionalmente imposible proteger a un agente móvil de un host malicioso. Algunos investigadores están buscando medios sociológicos (en lugar del modelo computacional duro) para imponer un buen comportamiento del anfitrión.

**Ejercicios:**

1. Desarrolle brevemente qué son los agentes

Un agente es una entidad que actúa de forma autónoma en nombre de otros. Posee la capacidad de operar con cierto nivel de **proactividad y reactividad**, adaptándose al entorno para alcanzar los objetivos que se le asignen. Además, los agentes pueden exhibir atributos como **aprendizaje, cooperación y movilidad**. Estos agentes de software se diferencian de objetos tradicionales porque pueden ejecutar tareas de forma independiente y tomar decisiones según la situación sin intervención constante del usuario.

1. ¿Qué tipos de agentes existen?

* **Agentes de interfaz:** Se enfocan en aprender del comportamiento del usuario para asistirlo mejor, sugiriendo alternativas o atajos en tareas específicas, como un asistente personal.
* **Agentes colaborativos:** Trabajan en conjunto con otros agentes para resolver problemas complejos, siendo esenciales en sistemas multi-agente.
* **Agentes de información:** Se especializan en recopilar, manipular o gestionar información en entornos distribuidos, como búsquedas en la web.
* **Agentes reactivos:** Operan con un enfoque de estímulo-respuesta, reaccionando rápidamente a los cambios en su entorno.
* **Agentes híbridos:** Combinan características de varios tipos de agentes, como la movilidad y la reactividad, para maximizar su eficacia.
* **Agentes móviles:** Pueden migrar entre diferentes sistemas para ejecutar tareas en múltiples entornos de red, siendo útiles en entornos de red heterogéneos y poco confiables​.