ESTÁNDAR DE HERRAMIENTAS TECNOLÓGICAS PARA DESARROLLO DE SISTEMAS

1. INTRODUCCIÓN

Este documento tiene como objetivo describir la propuesta y justificación del estándar de herramientas tecnológicas para el desarrollo de software con que cuenta la Unidad de Servicios de Cómputo Académico de la Facultad de Ingeniería (UNICA).

2. PROPUESTA

Con esto se intenta instaurar un estándar de herramientas tecnológicas para la Facultad de Ingeniería dada la amplia experiencia y recursos humanos con que cuenta la Facultad en las siguientes herramientas.

2.2. Lenguaje de modelado unificado (UML)

2.3. Manejador de base de datos POSTGRESQL

2.4. Lenguaje JAVA

2.5. Sistema operativo LINUX

3. JUSTIFICACION

3.2. *LENGUAJE DE MODELADO UNIFICADO (UML)*

UML es un lenguaje para hacer modelos y es independiente de los métodos de análisis y diseño. Existen diferencias importantes entre un método y un lenguaje de modelado. Un *método* es una manera explícita de estructurar el pensamiento y las acciones de cada individuo. Además, el método le dice al usuario qué hacer, cómo hacerlo, cuándo hacerlo y por qué hacerlo; mientras que el lenguaje de modelado carece de estas instrucciones. Los métodos contienen modelos y esos modelos son utilizados para describir algo y comunicar los resultados del uso del método.

Un modelo es expresado en un *lenguaje de modelado*. Un lenguaje de modelado consiste de vistas, diagramas, elementos de modelo y un conjunto de mecanismos generales o reglas que indican cómo utilizar los elementos. Las reglas son sintácticas, semánticas y pragmáticas

Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

3.2.1. UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

ESTÁNDAR DE HERRAMIENTAS TECNOLÓGICAS PARA DESARROLLO DE SISTEMAS

3.2.1.1. *Diagramas de comportamiento*: Especifican las partes dinámicas de un sistema tales como estados del sistema, flujo de control de actividades, secuencia de mensajes y mas.

3.2.1.2. *Diagramas de clases:* Conjunto de clases, interfaces y colaboraciones, y las relaciones entre ellas.

3.2.1.3. *Diagramas de objetos*: Instantáneas de las instancias de los elementos encontrados en los diagramas de clases.

3.2.1.4. *Diagramas de componentes*: Conjunto de componentes y sus relaciones.

3.2.1.5. *Diagramas de despliegue*: Conjunto de nodos y sus relaciones.

3.2.1.6. *Diagramas de casos de uso*: Conjunto de casos de uso y actores y sus relaciones. Son importantes para organizar y modelar el sistema.

3.2.1.7. Diagramas de interacción:

3.2.1.8. *Diagramas de secuencia*: Conjunto de objetos y los mensajes enviados y recibidos por ellos. Resalta ordenación temporal de los mensajes.

3.2.1.9. *Diagramas de colaboración*: Resalta organización estructural de objetos que envían y reciben mensajes.

3.2.1.10. *Diagramas de estados*: Representan máquinas de estados, construida por estados, transiciones, eventos y actividades. Útiles para modelar sistemas reactivos.

3.2.1.11. *Diagramas de actividades*: Muestran el flujo de actividades de un sistema. Importantes para modelar la función de un sistema, así como para resaltar el flujo de control entre objetos.

3.2.2. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas. Las vistas existentes en UML son:

3.2.2.1. *Vista casos de uso*: Muestra el comportamiento del sistema tal y como es percibido por usuarios, analistas y encargados de pruebas. Se forma con los diagramas de casos de uso, colaboración, estados y actividades.

3.2.2.2. *Vista de diseño*: Comprende el vocabulario del problema y su solución, y soporta los requisitos funcionales del sistema (servicios que el sistema debería proporcionar a los usuarios finales). Se forma con los diagramas de clases, objetos, colaboración, estados y actividades.

3.2.2.3. *Vista de procesos*: Hilos y procesos que forman mecanismos de sincronización y concurrencia del sistema. Se forma con los diagramas de la vista de diseño; recalcando las clases y objetos referentes a procesos.

3.2.2.4. *Vista de implementación*: Componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. Se forma con los diagramas de componentes, colaboración, estados y actividades.

3.2.2.5. *Vista de despliegue*: Nodos que forman la topología hardware sobre la que se ejecuta el sistema. Distribución, entrega e instalación de las partes. Se forma con los diagramas de despliegue, interacción, estados y actividades.

3.2.3. UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

3.2.4. UML permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica.

3.2.5. UML es el primer método en publicar un meta-modelo en su propia notación, incluyendo la notación para la mayoría de la información de requisitos, análisis y diseño. Se trata pues de un meta-modelo auto-referencial (cualquier lenguaje de modelado de propósito general debería ser capaz de modelarse a sí mismo).

3.2.6. Los principales beneficios de UML son:

3.2.6.1. Mejores tiempos totales de desarrollo (de 50 % o más).
3.2.6.2. Modelar sistemas (y no sólo de software) utilizando conceptos orientados a objetos.
3.2.6.3. Establecer conceptos y artefactos ejecutables.
3.2.6.4. Encaminar el desarrollo del escalamiento en sistemas complejos de misión crítica.
3.2.6.5. Crear un lenguaje de modelado utilizado tanto por humanos como por máquinas.
3.2.6.6. Mejor soporte a la planeación y al control de proyectos.
3.2.6.7. Alta reutilización y minimización de costos

3.2.7. Fases del desarrollo de un sistema

Las fases del desarrollo de sistemas que soporta UML son: *Análisis de requerimientos*, *Análisis*, *Diseño*, *Programación* y *Pruebas*.

3.2.7.1. ***Análisis de Requerimientos***

UML tiene casos de uso (use-cases) para capturar los requerimientos del cliente. A través del modelado de casos de uso, los actores externos que tienen interés en el sistema son modelados con la funcionalidad que ellos requieren del sistema (los casos de uso). Los actores y los casos de uso son modelados con relaciones y

tienen asociaciones entre ellos o éstas son divididas en jerarquías. Los actores y casos de uso son descritos en un diagrama use-case. Cada use-case es descrito en texto y especifica los requerimientos del cliente: lo que él (o ella) espera del sistema sin considerar la funcionalidad que se implementará. Un análisis de requerimientos puede ser realizado también para procesos de negocios, no solamente para sistemas de software.

3.2.7.2. *Análisis*

La fase de análisis abarca las abstracciones primarias (clases y objetos) y mecanismos que están presentes en el dominio del problema. Las clases que se modelan son identificadas, con sus relaciones y descritas en un diagrama de clases. Las colaboraciones entre las clases para ejecutar los casos de uso también se consideran en esta fase a través de los modelos dinámicos en UML. Es importante notar que sólo se consideran clases que están en el dominio del problema (conceptos del mundo real) y todavía no se consideran clases que definen detalles y soluciones en el sistema de software, tales como clases para interfaces de usuario, bases de datos, comunicaciones, concurrencia, etc.

3.2.7.3. *Diseño*

En la fase de diseño, el resultado del análisis es expandido a una solución técnica. Se agregan nuevas clases que proveen de la infraestructura técnica: interfaces de usuario, manejo de bases de datos para almacenar objetos en una base de datos, comunicaciones con otros sistemas, etc. Las clases de dominio del problema del análisis son agregadas en esta fase. El diseño resulta en especificaciones detalladas para la fase de programación.

3.2.7.4. *Programación*

En esta fase las clases del diseño son convertidas a código en un lenguaje de programación orientado a objetos. Cuando se crean los modelos de análisis y diseño en UML, lo más aconsejable es trasladar mentalmente esos modelos a código.

3.2.7.5. *Pruebas*

Normalmente, un sistema es tratado en pruebas de unidades, pruebas de integración, pruebas de sistema, pruebas de aceptación, etc. Las pruebas de unidades se realizan a clases individuales o a un grupo de clases y son típicamente ejecutadas por el programador. Las pruebas de integración integran componentes y clases en orden para verificar que se ejecutan como se especificó. Las pruebas de sistema ven al sistema como una "caja negra" y validan que el sistema tenga la funcionalidad final que le usuario final espera. Las pruebas de aceptación conducidas por el cliente verifican que el sistema satisface los requerimientos y son similares a las pruebas de sistema.

3.3. LENGUAJE JAVA[1]

Características del lenguaje java

Los siguientes puntos muestran las principales características del lenguaje de programación Java.

3.3.1. Simple. Java ofrece toda la funcionalidad de un lenguaje de programación simplificando algunas tareas que en otros lenguajes requieren un esfuerzo adicional por parte del programador. Un ejemplo de lo anterior es el manejo de la memoria. La maquina virtual de Java (JVM) se encarga de su administración haciendo uso de un hilo de ejecución de baja prioridad encargado de detectar y liberar bloques de memoria libres de referencias (Garbage Collector). Al eliminar características como la aritmética de punteros, los registros struct, operaciones de reserva de memoria (malloc() en C), liberación de memoria (como free()en C), existencia de referencias en lugar de punteros, etc., reduce considerablemente los errores de programación generados al delegar la administración de la memoria al programador.

3.3.2. Orientado a objetos. Java es un lenguaje totalmente orientado a objetos: encapsulación, herencia, polimorfismo, etc. Todos los programas en Java son clases, y su representación en memoria son las instancias de una clase. Java incorpora la sobrecarga y sobre escritura de métodos. No soporta herencia múltiple, sin embargo, incorpora la implementación de interfaces.

3.3.3. Distribuido. Java cuenta con capacidades de interconexión TCP/IP. Existen librerías para interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con facilidad. Java proporciona las herramientas y librerías necesarias para la ejecución de sistemas distribuidos mediante Java RMI, JNDI, RMI-IIOP, JNI.

3.3.4. Robusto. Java proporciona diversos mecanismos con la finalidad de minimizar los errores de una aplicación en tiempo de compilación y en tiempo de ejecución algunos de ellos:

3.3.4.1. Tiempo de compilación.
- Comprobación de tipos de datos
- Requiere declaración explicita de métodos

3.3.4.2. Tiempo de Ejecución
- Verificación de la integridad del archivo de bytecodes
- Manejo de memoria

[1] JAVA (nombre de un tipo de café, originario del este de Asia, de la isla del mismo nombre), aunque hay algunos que afirman que el nombre deriva de las siglas de James Gosling, Arthur Van Hoff, y Andy Bechtolsheim.

- Operaciones de comprobación de límites de índices en arreglos evitando la posibilidad de sobrescribir o corromper memoria como resultado de punteros que señalan a zonas erróneas.
- Manejo de Excepciones

3.3.5. Portable. Java es un lenguaje multiplataforma en el sentido de que una aplicación no requiere ser re-compilada ó modificada para que pueda ser ejecutada en cualquier plataforma para la cual exista una maquina virtual (JVM).

3.3.6. Seguro. Antes de ejecutar un programa en Java, el código generado pasa por diversas pruebas con la finalidad de detectar posibles anomalías ó alteraciones de la integridad del código en bytecodes a interpretar:

3.3.6.1. El código no debe producir desbordamiento de operandos en la pila.
3.3.6.2. El tipo de los parámetros de todos los códigos de operación deben ser conocidos y correctos.
3.3.6.3. No debe existir alguna conversión ilegal de datos, por ejemplo, convertir enteros en punteros.
3.3.6.4. El acceso a los atributos de un objeto debe ser legal : public, private, protected
3.3.6.5. El código debe cumplir con las reglas de acceso y seguridad establecidas.

De forma similar, en tiempo de compilación, existen mecanismos que evitan la generación de código inseguro ó vulnerable, En Java no existen instrucciones para la manipulación de la memoria eliminando la posibilidad de acceder a áreas ajenas ó la posibilidad de desbordamientos.

3.3.6.6. En cuanto a código proveniente de una fuente remota (red), Java cuenta con mecanismos de seguridad. Algunos de ellos:
3.3.6.7. Esquema de seguridad empleado por los Applets (Sandbox) ,
3.3.6.8. Verificación de una llave digital por parte del cargador de clases antes de instanciar cualquier clase, etc.
3.3.6.9. Separación de las clases provenientes de una fuente externa y las locales con la finalidad de prevenir el indebido reemplazo de una clase externa por una local. (El cargador primero carga las clases locales y después las remotas).

3.3.6.10. Un aspecto considerado inseguro por los programadores es la posibilidad de la generación del código fuente a partir del archivo en bytecodes (.class) mediante el comando *javap*.

3.3.7. Multihilos. En java es posible ejecutar varias aplicaciones a la vez mediante el empleo de hilos de ejecución, lo que permite incrementar y optimizar el desempeño de una aplicación, al permitir la distribución de las tareas a realizar asociadas a hilos de ejecución debidamente organizadas y sincronizadas. La programación con multihilos es fundamental para el desarrollo de interfaces gráficas (swing), aplicaciones distribuidas, etc.

Para el modulo de Operación y control del pago del SIAR, esta característica es importante, como se verá mas adelante, debido principalmente a la cantidad de tareas que se requieren ejecutar al calcular una nómina.

3.3.8. Dinámico. Java es dinámico debido a que la carga de las clases se realiza cuando estas son requeridas, ya sea de forma local ó de algún punto de la red empleando algún protocolo de comunicación.

3.4. *MANAEJADOR BASES DE DATOS POSTGRESQL*

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS) que ha sido desarrollado de varias formas desde 1977. Comenzó como un proyecto denominado *Ingres* en la Universidad Berkeley de California. *Ingres* fue más tarde desarrollado comercialmente por la *Relational Technologies/Ingres Corporation*.

En 1986 otro equipo dirigido por *Michael Stonebraker* de Berkeley continuó el desarrollo del código de *Ingres* para crear un sistema de bases de datos objeto-relacionales llamado *Postgres*. En 1996, debido a un nuevo esfuerzo de código abierto y a la incrementada funcionalidad del software, *Postgres* fue renombrado a *PostgreSQL*, tras un breve periplo como *Postgres95*. El proyecto *PostgreSQL* sigue actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto.

3.4.1. *Características PostgreSQL*

PostgreSQL está ampliamente considerado como el sistema de bases de datos de código abierto más avanzado del mundo.

PostgreSQL dispone de las siguientes características:

- **3.4.1.1. Transacciones**
- **3.4.1.2. Subselects**
- **3.4.1.3. Triggers**
- **3.4.1.4. Vistas**
- **3.4.1.5. Integridad referencial de claves externas**
- **3.4.1.6. Un sofisticado sistema de bloqueos**

Además, también presenta algunas funcionalidades que no encontramos en otras bases de datos comerciales, como tipos de datos definibles por el usuario, herencia, reglas, y control de concurrencia multi-versión que permite reducir el bloqueo de conexión.

3.4.2. A continuación se presenta una comparación de PostgreSQL con los demás motores:

Es posible instalarlo un número ilimitado de veces sin temor a sobrepasar la cantidad de licencias, la principal preocupación de muchos proveedores de bases de datos comerciales.

- **Velocidad y rendimiento**
- **Flexibilidad para extenderse según se requiera**
- **Diseño escalable**
- **Mínimos requerimientos de administración**
- **Sigue estándares ANSI**
- **Excelente Atomicidad, Consistencia, Aislamiento y Durabilidad (Prueba del ACID)**

El siguiente cuadro ilustra el comportamiento de PostgreSQL contra otros motores de datos.

	MySQL	Microsoft	Oracle	PostgreSQL
Conexiones simultáneas (instalación default)	101	1000	41	**32**
Columnas en tabla	2819	1024	1000	**1600**
Máximo tamaño de renglón en tabla (omitiendo blobs)	65534	8036	255000	**103275**
Tamaño del renglón en tabla sin nulos (omitiendo blobs)	65502	8036	255000	**103275**
Tamaño query	1048574	16777216	16777216	**16777216**
Valor FALSE	0			**0**
Valor TRUE	1			**1**

SISTEMA OPERATIVO DE LIBRE DISTRIBUCION[2]

3.5. Linux

El sistema operativo Linux es compatible con UNIX el cual es de gran uso por sus características como servidor, además cuenta con dos características que lo hacen diferente a los demás sistemas comerciales las cuales son:

3.5.1. Es de libre distribución **General Public License (GNU),** esto significa que no se tiene que pagar una licencia

3.5.2. Se distribuye junto con el **código fuente** por lo que si deseamos realizar correcciones o adecuaciones para alguna tarea específica podemos hacerlo directamente.

3.5.3. El sistema es formado por el núcleo del sistema **(KERNEL) y** una serie de programas y librerías, los cuales han sido diseñados e implantados por una gran cantidad de programadores a lo largo del mundo, lo que lo hace un sistema en constante desarrollo.

[2] Software libre vs software propietario Ventajas y desventajas Culebro Juárez, Montserrat. Gomez Herrera, Wendy Guadalupe. Torres Sanchez, Susana. México, Mayo 2006.

ESTÁNDAR DE HERRAMIENTAS TECNOLÓGICAS PARA DESARROLLO DE SISTEMAS

3.5.4. Características

Dentro de las características más significativas que distinguen a Linux están:

- **3.5.4.1.** Multitarea
- **3.5.4.2.** Multiusuario
- **3.5.4.3.** Multiplataforma
- **3.5.4.4.** Multiprocesador
- **3.5.4.5.** Memoria Virtual
- **3.5.4.6.** Consolas virtuales múltiples
- **3.5.4.7.** Shells programables

3.5.5. Ventajas del software libre.

El software libre presenta una serie de ventajas sobre el software propietario por los derechos que otorga a sus usuarios. Algunas de estas ventajas pueden ser mías apreciadas por los usuarios particulares, otras por las empresas, y otras por las administraciones públicas.

Principales ventajas.

3.5.5.1. Bajo costo de adquisición y libre uso. El software, como mercadería, por lo general no esta a la venta. Lo que el usuario adquiere, a través de una erogación monetaria o sin ella, es una licencia respecto de los usos que puede dar a los programas en cuestión. El software no sólo cuesta un precio de adquisición de licencia. También cuesta mantenerlo, operarlo, ajustarlo.

3.5.5.2. Innovación tecnológica. El software libre, tiene como objetivo principal compartir la información, trabajando de manera cooperativa. Este es principalmente el modelo sobre el que la humanidad ha innovado y avanzado. La ideología de los defensores del software libre, es que el conocimiento le pertenece a la humanidad, sin hacer distingos. Por lo tanto, los usuarios tienen un destacado papel al influir decisivamente en la dirección hacia donde evolucionan los programas: votando los errores que quieren que sean corregidos, proponiendo nueva funcionalidad al programa, o contribuyendo ellos mismos en el desarrollo del software

3.5.5.3. Requisitos de hardware menores y durabilidad de las soluciones. Aunque resulta imposible generalizar, si existen casos documentados que demuestran que las soluciones de software libre tienen unos requisitos de hardware menor, y por lo tanto son mías baratas de implementar. Por ejemplo, los sistemas Linux que actúan de servidores pueden ser utilizados sin la interfaz grafica, con la consecuente reducción de requisitos de hardware necesarios.

También es importante destacar que en el software propietario el autor puede decidir en un momento dado no continuar el proyecto para una cierta plataforma, para un hardware que considera antiguo, o descontinuar el soporte para una versión de su software. En las aplicaciones de software libre, estas decisiones no pueden ser tomadas por una empresa o individuo sino por toda una comunidad, con diferentes intereses. Lo que se traduce en un

mejor soporte -de manera general- para las versiones antiguas de software y de plataformas de hardware o software minoritarias.

3.5.5.4. Escrutinio publico. El modelo de desarrollo de software libre sigue un método a través de la cual trabajan de forma cooperativa los programadores que en gran parte son voluntarios y trabajan coordinadamente en Internet.

Lógicamente, el código fuente del programa esta a la vista de todo el mundo, y son frecuentes los casos en que se reportan errores que alguien ha descubierto leyendo o trabajando con ese código.

El proceso de revisión publica al que esta sometido el desarrollo del software libre imprime un gran dinamismo al proceso de corrección de errores. Los usuarios del programa de todo del mundo, gracias a que disponen del código fuente de dicho programa, pueden detectar sus posibles errores, corregirlos y contribuir a su desarrollo con sus mejoras. Son comunes los casos en que un error de seguridad en Linux se hace publico y con el la solución al mismo.

Con el software propietario la solución de los errores no llega hasta que el fabricante del programa puede asignar los recursos necesarios para solventar el problema y publicar la solución.

3.5.5.5. Independencia del proveedor. El software libre garantiza una independencia con respecto al proveedor gracias a la disponibilidad del código fuente. Cualquier empresa o profesional, con los conocimientos adecuados, puede seguir ofreciendo desarrollo o servicios para nuestra aplicación.

En el mundo del software propietario, sólo el desarrollador de la aplicación puede ofrecer todos los servicios, con el software libre, como su denominación lo indica, su uso es libre: todo aquel que lo tiene en su poder puede usarlo cuantas veces quiera, en cuantas maquinas quiera, a los fines que quiera.

De esta manera, utilizándolo, el usuario se libera de toda dependencia de un proveedor único, y puede administrar su crecimiento y operación con total autonomía, sin temor de costos ocultos ni extorsiones. Uno de los grandes problemas en la industria del software propietario es la dependencia que se crea entre el fabricante y el cliente.