Enl@ce: Revista Venezolana de Información,
Tecnología y Conocimiento
ISSN: 1690-7515
Depósito legal pp 200402ZU1624
Año 5: No. 3, Septiembre-Diciembre 2008, pp. 11-23

Cómo citar el artículo (Normas APA):
Rincón, C., Acurero, A. y Bracho, D. (2008). Aspectos de diseño de un entorno de programación colaborativo. *Enl@ce Revista Venezolana de Información, Tecnología y Conocimiento*, 5 (3), 11-23

# Aspectos de diseño de un entorno de programación colaborativo

***Carlos Rincón***
***Alfredo Acurero***
***David Bracho***[1]

**Resumen**

La programación colaborativa, o programación en par, es una de las áreas más estudiadas actualmente debido a su impacto en el proceso de desarrollo de software. El propósito de la presente investigación consistió en determinar los aspectos básicos de diseño que un entorno de programación colaborativo debe ofrecer a los desarrolladores. La metodología utilizada consistió en realizar un análisis documental sobre la propuesta de Lussier acerca de la programación colaborativa con la finalidad de determinar los aspectos de diseño, estructura e implementación de este tipo de entornos. Como resultado se obtuvo que desde el punto de vista del diseño, la comunicación entre desarrolladores, la revisión de código y la construcción del proyecto de software son los principales aspectos a considerar, y desde el punto de vista de la implementación se propone utilizar el modelo cliente - servidor y una arquitectura multiplataforma.

**Palabras clave:** entorno de programación, programación colaborativa, aspectos de diseño

Recibido: 22-11-07 Aceptado: 28-04-08

[1] Todos profesores miembros de la Unidad de Redes e Ingeniería Telemática, Departamento de Computación, Facultad Experimental de Ciencias, Universidad del Zulia. Correo electrónico para correspondencia crincon@luz.edu.ve

# Design Aspects of a Collaborative Programming Environment

**Abstract**

Collaborative programming or pair programming is today one of the most studied areas, because of its impact on the software developing process. The purpose of the present investigation was to determine the basic design aspects that a collaborative programming environment must offer to developers. The methodology used was a documental review of Lussier's work about collaborative programming, to determine the design aspects, the structure and the implementation aspects of this type of environments. As result, from the design point of view, the communication between developers, code review and the construction of software project are the basic aspects to consider, and from the implementation point of view, the use of the client - server model and a multiplatform architecture are proposed.

**Key words:** Programming Environment, Collaborative Programming, Design Aspects

## Motivación

Desde la aparición de los sistemas informáticos, los seres humanos los hemos utilizado como una herramienta para realizar un procedimiento de forma rápida y confiable. A la traducción de un proceso en lenguaje natural a lenguaje binario se le conoce como programa. El software es un programa o conjunto de programas, que facilitan la interacción entre el usuario y el hardware con la finalidad de realizar una tarea o proceso.

El proceso de desarrollo de software ha evolucionado a la par de los sistemas informáticos. En las primeras generaciones de sistemas informáticos, el usuario debía interactuar directamente con el hardware, haciendo el proceso de desarrollo de software complicado y dependiente del sistema de cómputo utilizado. Sin embargo, en la actualidad y como consecuencia de la evolución de los sistemas de cómputo, el proceso de desarrollo de software es independiente en gran porcentaje del hardware utilizado (abstracción del hardware), lo permite que este proceso sea menos complicado.

Según Pérez (1998), los lenguajes de programación se han transformado de simples traductores a sofisticados compiladores optimizadores. La evolución de los lenguajes de programación (de bajo nivel a alto nivel), se tradujo en la masificación del proceso de programación, mejorando la calidad y tiempo de desarrollo del software.

Desde el punto de vista de la metodología de trabajo utilizada en el proceso de desarrollo de software (programación), es importante diferenciar la programación en equipo de la programación colaborativa. Para Nosek (1998), la programación

en grupo significa esfuerzos coordinados de programadores individuales, quienes se dividen una tarea de programación de un sistema amplio y complejo, mientras que la programación colaborativa significa dos o más programadores trabajando conjuntamente sobre el mismo algoritmo o código.

Varios investigadores como Nawrocki y Wojciechowski (2001); Nosek (1998); Williams, Kessler, Cunningham y Jefries (2000), entre otros, confirman que mediante el uso de la programación colaborativa, se obtienen mejores resultados considerando métricas como tiempo de desarrollo y eficiencia del código desarrollado. Estos resultados fundamentan la necesidad de generación de herramientas que se fundamenten en la programación colaborativa, para el desarrollo de software.

La comunicación entre los usuarios de un entorno de programación colaborativo debe ser confiable y simple, principalmente debido a la necesidad de garantizar el intercambio de información entre los programadores sin que esto signifique una sobrecarga que influya en el rendimiento del entorno.

La mayoría de los entornos de programación distribuidos se basan en métodos de comunicación propietarios, lo que se traduce en la incapacidad de los mismos para interactuar entre sí, aunado a la complejidad implícita en la utilización de protocolos binarios.

Según Cerami (2002), en la actualidad los servicios web se presentan como la herramienta de comunicación ideal para interconectar aplicaciones informáticas, esto basado en su simplicidad (basados en XML) y su confiabilidad. Lo antes expuesto, fundamenta la idea propuesta de utilizar los servicios web como plataforma de comunicación de los entornos de programación colaborativos, dado a que permitirá la interacción de los programadores cuyas estaciones de trabajo se encuentren conectadas a una red de datos.

## Fundamentos teóricos

La fundamentación teórica de toda investigación, es la base para la generación de conocimiento y la formulación de conclusiones y aportes relevantes. Es por ello, que se definirán algunos términos asociados con la investigación:

1. Programación par: según Nawrocki y Wojciechowski (2001), se refiere a la programación hecha por dos programadores. Mientras uno de ellos se enfoca en el código, el otro se encuentra continuamente revisando la calidad, haciendo preguntas, tratando de entender, y buscando alternativas que puedan mejorar lo realizado y evitar los defectos.

   Asimismo, destacan Nawrocki y Wojciechowski (2001), que ambos programadores se intercambian los roles cada cierto tiempo. La Programación Par es considerada como una de las doce prácticas de la llamada Programación eXtrema(XP).

2. Programación eXtrema (XP): resalta Williams (2000), que se trata de un tipo de metodología que no tiene evidencia estadística, como lo hace el Proceso de Programación Personal (PSP),

para lograr su efectividad. La efectividad en este tipo de programación, es altamente anecdótica (o basada en experiencias), pero que ha logrado llamar la atención de investigadores y consultores de ingeniería de software. Explica Williams que la XP se basa fuertemente en la programación par. Durante la programación, se realiza una revisión constante del código, orientándose a las filosofías de la remoción eficiente de la programación defectuosa, y la prevención de ella. Para la mayoría de las metodologías que se basan en XP, la acumulación de requerimientos, la localización de los recursos y las prácticas de diseño, son los aspectos más relevantes.

3. Grupos de trabajo virtuales: según Baheti, Gehringer y Stotts (2002), un grupo de trabajo virtual se refiere a un conjunto de personas que buscan un objetivo común que es el desarrollo del software, tomando en cuenta la distancia, la cultura y los límites organizacionales. En comparación con los grupos de trabajo en sitio, presenta ciertas desventajas, tales como: la comunicación entre los miembros, que suele estar limitada por factores de distancia, tiempo e infraestructura de comunicación. Asimismo, los miembros suelen no preocuparse por el resto del grupo, y el acceso a ciertos recursos u objetos es complicado.
4. Servicios WEB: Según Cerami (2002), un servicio web es un servicio que:
   - Está disponible en Internet o en una red privada (intranet),
   - usa el sistema de mensajería ya estandarizado XML,
   - no está atado a un sistema operativo o lenguaje de programación específico,
   - se autodescribe mediante gramática XML (Web Service Definition Languaje - WSDL) y,
   - se puede descubrir mediante un mecanismo simple.

El proceso de descubrimiento de servicios web, es uno de los problemas principales que plantea la arquitectura orientada a servicios, debido a la heterogeneidad entre los mismos. Existen diferentes soluciones al problema del descubrimiento de servicios que son clasificadas por Garofalakis, Panagis, Sakkopoulos y Tsakalidis (2004), según el rol que desempeñe la solución:

*Rol en el proceso de descubrimiento:*

*Catálogos:*

Los catálogos de servicios web son la base tecnológica predominante para los mecanismos de descubrimiento de servicios. Son repositorios especializados que contienen la información sobre un grupo de servicios web. El principal mecanismo de descubrimiento de servicios web mediante catálogos es UDDI (universal description, discovery and integration). UDDI normalmente es descrita como las páginas amarillas de los servicios web, que contiene la información sobre los fabricantes de servicios, sus direcciones de contacto, la lista de los servicios que estos ofrecen, las direcciones de los servicios web, las interfaces de los servicios web, entre otros.

El principal problema de UDDI (y de los catálogos), es que está basado en una arquitectura centralizada, lo que se traduce en problemas de congestionamiento (a medida que aumente el uso de los servicios web) y el problema del único punto de fallo, es decir, si el repositorio falla, se pierde la capacidad de descubrimiento de servicios web.

La arquitectura de UDDI en la actualidad tiene similitud con la arquitectura utilizada para prestar el servicio de DNS (*domain name server*), la cual ha rendido frutos en Internet, sin embargo, el apoyo de las corporaciones de software es fundamental para permitir el crecimiento de la plataforma UDDI.

Soluciones Punto a Punto (P2P):

La arquitectura P2P provee una plataforma para el enrutamiento y localización de información de forma descentralizada, donde cada punto provee el servicio de localización y además actúa como servidor de acceso a los servicios. La utilización del modelo distribuido, evita los problemas planteados en el modelo centralizado, pero presenta la complicación de la transmisión de la información a todos los usuarios del la red P2P.

5. Conocimiento distribuido: Según Williams (2000) el conocimiento distribuido es un campo de la ciencia cognitiva, que se basa en los siguientes puntos:

   Metas y planes compartidos: donde se busca que, mientras dure la interacción entre los participantes, se logre una comunicación eficiente, se aumente el rango de planes alternativos para llevar a cabo las actividades y se mejoren los planes anteriores constantemente (tener presente las experiencias de los planes aplicados anteriormente).

   Comunicación eficiente: consiste en mejorar las conversaciones hasta el punto de comunicar sólo lo que es necesario comunicar, y para ellos las metas y planes deben ser compartidos en igual medida.

   Aumento del rango de planes alternativos: se refiere a que un sistema con múltiples participantes, tiene más probabilidad de generar diversidad planes, principalmente, debido a que cada participante tiene una experiencia propia; los mismos participantes podrían tener diferentes tipos de acceso o privilegios a cierta información relevante para el proyecto; y, de acuerdo a su rol dentro del grupo, pueden establecer actividades que se relacionen con las de los demás.

   Tener presentes las experiencias de los planes implementados anteriormente: generalmente es útil en los casos en que se presente una situación que haya sido tratada en ocasiones anteriores, y pudiera tomarse esa experiencia (buena o mala) para enrumbar el plan actual.

6. Proceso de Software Personal (PSP): Williams (2000) recalca que el PSP define un marco de trabajo para el desarrollo de software, e incluye operaciones definidas o subprocesos, y técnicas de análisis y de medición, que ayuda a los programadores a desarrollar sus propias habilidades y su rendimiento personal. Se basa en la filosofía de la revisión del diseño y del código para la eliminación de errores. Sin embargo,

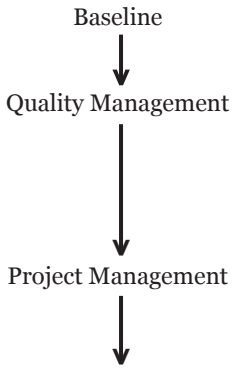
su mayor enfoque es hacia la prevención de los errores como factor de eficiencia: antes que corregir los errores es preferible evitarlos.

7. Proceso de Software Colaborativo (PSC): explica Williams (2000) que el PSC se basa en varios niveles (Ver **Figura 1**):

**Figura 1**

**Proceso de Software Colaborativo**

| Level | CSP |
|---|---|
| 0.0 | Baseline / Current process |
| 0.1 | Coding standard<br>Size measurement<br>Process improvement plan |
| 1.0 | Analysis (Use Case)<br>CRC Card Design Brainstorming<br>Design |
| 1.1 | Code review<br>Design reviews<br>Testing<br>Measurements |
| 2.0 | Size estimating<br>Resource estimating |
| 2.1 | Task planning<br>Schedule planning |




Fuente: Williams (2000)

- Nivel 0.0: los programadores utilizan su proceso natural y no recomiendan o imponen ningún otro proceso adicional. Este nivel sirve de línea base para el proceso, o se comienza desde el proceso actual.
- Nivel 0.1: se realizan varias mejoras pequeñas a los procesos descritos, y se comienza a utilizar una codificación estándar.
- Nivel 1.0.: se basa en el análisis y el diseño. Específicamente, el análisis tiene que ver con entender el problema, dar respuestas a las preguntas más relevantes y estar sintonía con las metas.
- Nivel 1.1.: en este nivel se realiza la revisión del código y del diseño, se incluye una etapa de pruebas (se aplican las técnicas de la caja negra, caja blanca y regresión automática) y, por último, la medición de la efectividad de la calidad obtenida, con los datos que han sido registrados desde fases o experiencias anteriores.

- Nivel 2.0.: este nivel se relaciona con la gestión del proyecto, enmarcada por la estimación del tamaño del producto y de los recursos. Para ello se utiliza un análisis de la regresión lineal estadística sobre una base de datos histórica sobre proyectos pasados.
- Nivel 2.1.: el último nivel se refiere a la planificación de tareas necesarias para culminar el proyecto, y el tiempo que se ha empleado en cada tarea. Para ello se toma como tiempo base, el comienzo en el nivel 0.0.

**Algunos antecedentes**

Varios autores han analizado algunos aspectos de la programación en entornos avanzados. A continuación se presentan algunas de las investigaciones más relevantes, que sirven como base al presente trabajo:

Según Nawrocki y Wojciechowski (2001), la programación en entornos avanzados ha sido relacionada por muchos con la llamada programación en pareja (o programación colaborativa donde dos personas trabajan al mismo tiempo, en el mismo programa.

McDowell, Werner, Bullock y Fernald (2003) realizaron un estudio que examinó la efectividad de la programación en pareja, y cómo afectaba el rendimiento y la toma de decisiones de los estudiantes. Entre los resultados obtenidos, se encontró que este tipo de programación en particular produjo mejores programas, con soluciones confiables y se generó una motivación particular por la programación en dichos estudiantes, hasta el punto de culminar los cursos de programación introductorios que se les impartía durante el estudio (un total de cuatro cursos). Todos estos resultados fueron obtenidos en comparación con estudiantes que programaron solos. Explican los autores que lo más relevante de los resultados obtenidos, es que este tipo de técnicas puede aplicarse como una herramienta pedagógica en los cursos a todo nivel. Además, recalca que la representación femenina en las carreras de programación (al menos en los cursos aplicados para el estudio) es muy baja, lo cual propone una futura investigación, para determinar si esta técnica también puede atraer y retener el factor femenino.

Nosek (1998), realizó un estudio basado en la programación de grupos, y tomó como referencia el uso de la programación en pareja definida por Nawrocki y Wojciechowski (2001). Para ello se basó en las dos variables principales: LEGIBILIDAD, la cual se refiere al nivel de la estrategia utilizada para resolver el problema, y FUNCIONALIDAD, la cual se refiere al grado de correspondencia de la solución presentada con respecto al problema planteado. Todos los experimentos fueron realizados bajo las mismas condiciones (sistema operativo, lenguajes, entre otros) tanto para los equipos (parejas) como para los programadores individuales. Basados en la técnica estadística t-test, se obtuvo que las parejas de programadores, produjeron soluciones más legibles y funcionales que los programadores individuales. Asimismo, se confirmó que los niveles de confianza mutua (confidence, en inglés) y motivación (enjoyment, en inglés) fueron más elevados, que los mostrados por los programadores individuales. Sin embargo,

aunque estadísticamente no pudo corroborarse que el tiempo promedio para presentar la solución fuera menor en el caso de la programación en pareja (debido a que no pudo medirse con exactitud el tiempo dedicado a cada tarea específica), los resultados se corresponden con ahorro de tiempo cuando se programa en pareja. En general, se confirmó que los programadores con más años de experiencia tuvieron un mejor rendimiento.

Sin embargo, destaca Nosek que muchas de las investigaciones son realizadas con estudiantes que no poseen las suficientes destrezas y habilidades, y por ello, los resultados suelen ser, a veces, alejados de la realidad. En este experimento en particular, la calidad de los resultados mostrados por programadores de experiencia, fue muy superior a la de aquellos con poca experiencia. Definitivamente, debe estudiarse más en profundidad, la relación de la programación en pareja con respecto a las ganancias o dividendos que pueda ofrecer a las organizaciones, partiendo del hecho de que los tiempos son significativamente relevantes para ser competitivos hoy. Los puntos a desarrollar: la rapidez de obtención de la solución y la calidad de lo obtenido.

Otro estudio importante, realizado por Cockburn y Williams (2000), se refiere a los costos y beneficios asociados a la programación en pareja. En dicho experimento, se determinó que la programación colaborativa mejora la calidad del diseño, reduce los riesgos, mejora las habilidades técnicas, mejora las comunicaciones del equipo de trabajo, y es significativamente mucho más motivadora que la programación individual, con respecto a un costo de tiempo de desarrollo de apenas el 15%.

Destacan Cockburn y Williams, que los grandes beneficios que aporta, son muchos mas relevantes que los costos asociados. Más específicamente, los beneficios se relacionan con la detección de errores al momento o la revisión continua del código, la tormenta de ideas produce mejor calidad de código, los programadores complementan sus conocimientos y por ende, sus niveles de aprendizaje son más elevados; cuando termina el proyecto más de una persona conoce los detalles del mismo, lo que evita la dependencia sobre una sola persona, mejora las relaciones entre los miembros del equipo y disfrutan mejor su trabajo. De este último punto, se deriva que los niveles de productividad tiendan a aumentar significativamente.

Williams y colaboradores (2000) explican en un artículo que recopila ciertas anécdotas y experiencias sobre el uso de la programación en pareja o colaborativa, los beneficios de su uso casi en todos los niveles. En primer lugar, se hace referencia a ciertas anécdotas y luego a ciertas experiencias y experimentos realizados en la Universidad de Utah que validan cuantitativamente lo obtenido a través de experiencias individuales. Un aspecto a recalcar es que generalmente en situaciones y ambientes de alto nivel de presión, los programadores individuales tienden a aplicar técnicas indisciplinadas, lo cual puede conllevar a consecuencias fatales para las organizaciones. En cambio, en la programación colaborativa la probabilidad de que ambos desvíen las técnicas es mucho menor, con lo que se garantizaría aún más la calidad del producto obtenido y la disciplina: los niveles de responsabilidad grupal son mayo-

res a los individuales. Sin embargo, dado que los experimentos fueron realizados a nivel individual y universitario, es necesario realizarlos a niveles industriales, para corroborar los resultados y los beneficios obtenidos, de manera que contemplen el aspecto de la integración de las tareas y mayor cantidad de grupos de trabajo.

Williams (2000), realizó un estudio que formula el Proceso de Software Colaborativo (PSC), en contraste con el Proceso de Software Personal o Individual (PSP). Debido a que ambos procesos se basan en procesos repetitivos y definidos, se trató de establecer el PSP como parámetro principal para la formulación del PSC. Para ello se realizaron experimentos estadísticos para comprobar la efectividad del nuevo método (proceso). La base de los experimentos tomó en cuenta dos grupos de trabajo, los cuales aprendieron tanto PSC como PSP. Tal investigación produjo un proceso repetitivo y bien definido para la programación en parejas validando, entre otras cosas, un nivel de calidad más elevado en la programación en parejas. Se determinó además que dada la mayor calidad del trabajo colaborativo, los costos organizacionales se reducen considerablemente; y que el trabajo se disfruta más cuando se hace de forma colaborativa.

En general, se obtuvo una mejora en las habilidades para la resolución de problemas, mejores diseños, mayor aprendizaje, mayor comunicación organizacional, menores riesgos y un equipo de trabajo compenetrado. Sin embargo, y aunque estadísticamente no es significativo según Williams (2000), los tiempos para la obtención del producto final fueron mayores cuando se trabajó en parejas (15% adicional). Tendría que analizarse más a fondo si el tiempo se convierte definitivamente o no en factor determinante a la hora de comparar PSC con PSP. También se propone la validación a nivel industrial, para contrastar los resultados obtenidos empíricamente con el mundo en producción. Otro experimento interesante podría ser los efectos de la programación en grupos grandes y la eficiencia de la comunicación.

También propone Williams que la teoría sobre el área del conocimiento distribuido y la ciencia cognitiva, debe estudiarse más a fondo en la programación colaborativa. Asimismo, la colaboración distribuida debe explotarse a profundidad, debido a que el problema de programación colaborativa remota se incrementa con las distancias geográficas, la plataforma de comunicación, e inclusive, el idioma. Por último, se propone orientar otras investigaciones hacia el reconocimiento y aislamiento de los factores que afectan positivamente la llamada programación extrema (XP), debido a que no es sencillo identificarlos.

**Aporte**

El diseño de los entornos de programación colaborativos debe fundamentarse en una metodología de trabajo en equipo que fortalezca la interacción entre los miembros del mismo, minimice el tiempo requerido para desarrollar una solución informática y maximice la eficacia en el proceso de desarrollo de la misma.

Lussier (2004), presenta la metodología de trabajo del grupo de desarrolladores del proyecto de software libre WINE (API de Windows), la cual

sirve de base para generar los aspectos de diseño propuestos para los futuros entornos de programación colaborativos:

Roles en el grupo de trabajo:

- Los desarrolladores: Son los encargados de programar la solución informática. Escogen las tareas de programación a desarrollar basados en una lista de tareas por realizar especificada para el grupo.
- Los revisores: Cuando un desarrollador envía su código a la lista de correo, cualquier desarrollador puede criticarlo, encontrando errores y haciendo sugerencias, sin embargo es el desarrollador del código el responsable de hacer los cambios.
- El comprometedor: Es el líder del grupo de trabajo. Se encarga de decidir si un código sometido a su consideración pasa a ser parte de la solución informática o es devuelto para correcciones.

Procedimiento del grupo de trabajo:

El proceso de entrega de código es sencillo. Los desarrolladores generan el código y lo someten a la consideración de los revisores y el comprometedor a través de la lista de correo. En este punto del proceso cualquier miembro de la lista puede revisar el código y someter comentarios. El comprometedor tiene la decisión final sobre la aceptación o modificación del código.

El resultado de la investigación de Lussier, determinó que la metodología utilizada por el grupo de desarrolladores del proyecto WINE, mejora el rendimiento del proceso de desarrollo de software en relación a la metodología basada en organigramas jerárquicos, utilizados comúnmente por las compañías desarrolladoras de software propietario.

Analizando el proceso explicado por Lussier y considerando los resultados obtenidos por éste, se proponen unos aspectos de diseño para los entornos de programación colaborativos basados en la optimización de este proceso, considerando la automatización como factor fundamental.

Aspectos de diseño:

Comunicación entre Desarrolladores: el entorno de programación debe ofrecer un mecanismo automatizado para la interacción entre los desarrolladores. Soluciones como listas de correo son consideradas como elementos externos al entorno, por lo que se propone un mecanismo de comunicación propio del entorno basado en servicios web. La utilización de servicios web facilita la interrelación entre los usuarios del entorno, permitiendo la comunicación entre éstos, desde cualquier parte del mundo.

Revisión de código: mediante la comunicación ofrecida por los servicios web, los desarrolladores deben, además de realizar la tarea de programación, evaluar el código sometido por otros programadores para su evaluación. Mediante este proceso, los usuarios que evalúen el código podrán emitir sugerencias al programador del código.

Construcción del proyecto de software: considerando el proyecto de software como la suma de las tareas de programación encomendadas a los usuarios del entorno, éste debe recopilar de forma automatizada todos los códigos sometidos por los programadores y revisados por sus compañeros,

con la finalidad de conformar el producto final, previa evaluación definitiva realizada por el líder del proyecto de software.

Es importante resaltar que mediante estos aspectos de diseño se pretende potenciar la idea de la programación en par. Según Nawrocki y Wojciechowski (2001), la idea de la programación en par consiste en dos programadores desarrollando una misma tarea de programación, en donde un programador escribe el código mientras el otro se encarga de revisarlo y garantizar su eficiencia. Los aspectos planteados maximizan este concepto debido a que se eleva el número de programadores que revisan el código (n-1, donde n es la cantidad de programadores del entorno), ofreciendo la automatización de todo el proceso de desarrollo de software, lo que se traduce en un mejor rendimiento del equipo de desarrollo.

Características del entorno de programación colaborativo

El entorno a desarrollar estará fundamentado en el modelo Cliente-Servidor, por lo que constará de 2 tipos de procesos, los clientes y el servidor. La selección de este modelo se fundamenta en la necesidad de ofrecer una solución informática a los desarrolladores que consuma la menor cantidad de recursos computacionales.

*Los procesos Clientes:*

Será el proceso que permitirá al desarrollador generar los programas (editor + compilador + enlazador), utilizando los recursos locales. Además ofrecerá las siguientes características:

1. Un mecanismo de comunicación orientado a la conexión como la mensajería instantánea, que permita a los programadores tener una relación directa buscando con esto minimizar el tiempo de desarrollo de las soluciones.
2. Actualización en línea de recomendaciones o correcciones realizadas por los revisores. Este proceso debe permitir al desarrollador que recibe las observaciones sobre su código, decidir si acoger dichas observaciones o debatir con el revisor las observaciones realzadas (utilizando el mecanismo de comunicación definido en el punto anterior). Considerando que el modelo seleccionado para lograr la comunicación entre procesos es el Cliente – Servidor, la tecnología a utilizar para implementar este modela serán los servicios web. La selección de los servicios web se fundamenta en su simplicidad de desarrollo y bajo costo computacional en el uso de la red.
3. Rutinas para la evaluación de código de otros desarrolladores, mediante las cuales el desarrollador pueda proponer correcciones u observaciones a los códigos presentados por otros desarrolladores. Estas rutinas deberán permitir que las correcciones u observaciones se realicen en tiempo real para garantizar así el efecto que se desea lograr con la implementación de la metodología de desarrollo propuesta por Lussier.

*El proceso Servidor:*

El proceso servidor se encargará de permitir la comunicación entre los procesos clientes (tanto a nivel de usuario como a nivel de programa). Por

lo antes expuesto deberá ofrecer las siguientes características:

1. Un mecanismo de validación de usuarios que permita al entorno conocer la ubicación (a nivel de red) y estado (activo o inactivo) de los desarrolladores. Este mecanismo facilitará las tareas que tienen los procesos clientes que estén directamente relacionadas con el uso de la red.
2. Una aplicación de control de versiones que garantice el proceso de desarrollo de software y minimice el trabajo que debe desarrollar el comprometedor al momento de hacer pública una nueva versión de una aplicación desarrollada en el entorno.
3. Una bitácora que almacene las diferentes operaciones que se realizan sobre un código en desarrollo, con la finalidad de ofrecer al comprometedor de una herramienta para el seguimiento del proceso de desarrollo, así como ofrecer métricas que permitan a éste tomar decisiones que mejoren dicho proceso.

El entorno de programación propuesto debe ser multiplataforma (ejecutarse en sistemas operativos basados en la filosofía de software libre o propietarios), razón por lo cual se propone ser desarrollado utilizando el lenguaje de programación JAVA, que permite mediante la utilización de máquinas virtuales, ejecutar aplicaciones en múltiples plataformas. De igual forma, el desarrollo e implementación de servicios web (necesarios para establecer la comunicación entre los distintos tipos de procesos que conforman el entorno) en JAVA, presenta poca o nula complejidad.

## Bibliografía

Baheti, P., Gehringer, E. F., y Stotts, P. D. (2002). *Exploring the efficacy of distributed pair programming*. Recuperado el 16 de septiembre de 2007 del sitio web del Departamento de Ciencias de la Computación. North Carolina University: http://rockfish.cs.unc.edu/pubs/XPU2002DXP.pdf.

Cerami, E. (2002). *Web Services Essentials, Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. USA: Editorial O'Reilly.

Cockburn, A. y Williams, L. (2000). *The costs and benefits of pair programming*. Recuperado el 16 de septiembre de 2007 del sitio web del Departamento de Ciencias de la Computación. North Carolina State University: http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

Garofalakis, J., Panagis, Y., Sakkopoulos, E., y Tsakalidis, A. (2004). *Web service discovery mechanisms: Looking for a needle in a haystack*? Hypermedia Development & Web Engineering Principles and Techniques: Put them in use. Recuperado el 15 de septiembre del 2007 del sitio web del Workshop on Web Engenniering. Santa Cruz, USA: http://www.ht04.org/workshops/WebEngineering/.

Lussier, S. (2004). New tricks: How open source changed the way my team works. [Versión Electrónica]. *IEEE* Software, Vol: 21:Pages: 68 - 72.

McDowell, C., Werner, L., Bullock, H. E., y Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. Recuperado el 18 de septiembre de 2007 del sitio web de la *IEEE Computer Society*. Washington, DC, USA: http://csdl.computer.org/dl/proceedings/icse/2003/1877/00/18770602.pdf.

Nawrocki, J. y Wojciechowski, A. (2001). *Experimental evaluation of pair programming*. Recuperado el 6 de septiembre del sitio web de la Universidad de Massachusetts. USA: http://www2.umassd.edu/swpi/xp/pairprogramming/Nawrocki.pdf

Nosek, J. T. (1998). *The case for collaborative programming*. Commun. [Versión Electrónica] ACM, 41(3):105-108.

Pérez-Quiñones, M. A. (1998). *Teaching history of programming languages to undergraduate students*. Recuperado el 18 de septiembre de 2007 del sitio web de la IEEE Computer Society. Washington, DC, USA: http://csdl.computer.org/dl/proceedings/fie/1998/4762/01/00736853.pdf.

Williams, L. (2000). *The Collaborative Software Process*. Recuperado el 5 de septiembre de 2007 del sitio web del Departamento de Ciencias de la Computación. The University of Utah: http://www.cs.utah.edu/~lwilliam/.

Williams, L., Kessler, R. R., Cunningham, W., y Jefries, R. (2000). *Strengthening the case for pair programming*. [Versión Electrónica]. IEEE Softw., 17(4):19-25.