Estudio de calidad y eficiencia de un enfoque de desarrollo software secuencial con programadores solos y en pareja

A study on quality and efficiency of a waterfall-like software development process applied by pair and solo programmers

Antonio A. Aguileta[1] Omar S. Gómez[2]*

Recibido 8 de mayo de 2017, aceptado 24 de mayo de 2018
Received: May 8, 2017 Accepted: May 24, 2018

RESUMEN

Los modelos de procesos de desarrollo software (MPDS) guían el desarrollo o el mantenimiento de productos software. Actualmente existen diversos MPDS aplicados en diferentes contextos. Uno de los MPDS ampliamente conocido es el MPDS en cascada, cuyas etapas que lo conforman se realizan de manera secuencial. Con el fin de obtener mayor conocimiento sobre la aplicación de un MPDS secuencial (tipo cascada), en el presente documento se reportan los resultados de un experimento controlado que estudia la calidad y eficiencia en el uso de un MPDS secuencial aplicado en programadores solos y en pareja. Dicho modelo es contrastado con un MPDS ad-hoc que es una aproximación de un MPDS iterativo usado como grupo de control. En cuanto a la calidad, los resultados de este experimento sugieren que el usar un MPDS secuencial arroja tasas de defectos similares que el MPDS usado como grupo de control. En cuanto a la eficiencia, los participantes quienes aplicaron el MPDS secuencial requirieron un número significativamente menor de compilaciones para realizar los ejercicios de programación planteados. En cuanto al tipo de programación, los programadores solos y en pareja se desempeñaron de manera similar. A pesar del criticismo existente en torno a un MPDS secuencial, nuestros resultados sugieren que en ciertos contextos su uso puede ser benéfico, como lo observado en nuestros hallazgos con respecto a la eficiencia.

Palabras clave: Modelo de cascada, programación en pareja, experimento controlado.

ABSTRACT

Software Development Life Cycle (SDLC) models guide trough the development or maintenance of software products. Currently we can find several SDLC models applied in different contexts. A well-known SDLC is the waterfall model, which its stages are performed in a sequential order. With the aim of gaining more knowledge about a waterfall-like SDLC model, in this paper we report the findings of a controlled experiment that studies the quality and efficiency of a waterfall-like SDLC model applied by solo and pair programmers. The studied SDLC model is contrasted with an iterative-like SDLC model defined as 'ad-hoc' that is used as a control group. Concerning the quality, our results suggest that any of these two SDLC models yield similar defect detection rates. Regarding the efficiency, results suggest that a waterfall-like SDLC model significantly requires less number of builds to complete the programming assignments. We did not observe significant differences between pair and solos programmers. Even though the criticism arose around the waterfall SDLC model, our results suggest that under certain contexts a waterfall-like SDLC model can yield beneficial effects, as the one observed concerning efficiency.

Keywords: Waterfall model, pair programming, controlled experiment.

[1] Facultad de Matemáticas. Universidad Autónoma de Yucatán. Mérida, México. E-mail: aaguilet@correo.uady.mx
[2] Facultad de Informática y Electrónica. Escuela Superior Politécnica de Chimborazo. Riobamba, Ecuador. E-mail: ogomez@espoch.edu.ec
* Autor de correspondencia

INTRODUCCIÓN

La forma en cómo se desarrollan nuevos productos software se ha diversificado desde los orígenes de la ingeniería de software. El desarrollo o mantenimiento de productos software suele regirse de acuerdo a un determinado modelo de proceso de desarrollo. Un modelo de proceso de desarrollo software (MPDS) es una representación simplificada de un proceso software particular [1]. Un MPDS se conforma por pasos o etapas que guían el proceso de desarrollo o mantenimiento de productos software.

En la actualidad existen diversos MPDS utilizados en mayor o menor medida por la industria del software. Ejemplos de MPDS conocidos son: modelo en cascada [2], modelo incremental [1], modelo en espiral [3], modelo V [4], y modelo RUP [5].

Un determinado MPDS no necesariamente se ajusta a todos los contextos del desarrollo software (donde se involucran aspectos técnicos y organizacionales), en este sentido, cada MPDS ofrece ventajas y desventajas [6-7] que pueden resultar en mayor o menor beneficio en un contexto determinado.

El MPDS que es ampliamente conocido tanto en el ámbito académico como en el industrial es el modelo en cascada [1]. Este modelo es también el que ha generado diversas críticas sobre su adopción [8 - 10], entre estas críticas principalmente destacan: la falta de involucramiento del cliente en las distintas etapas del desarrollo [11] y la dificultad de gestionar requisitos cambiantes o volátiles [12].

Con el fin de obtener mayor conocimiento sobre el uso de un MPDS secuencial (tipo cascada), en la presente investigación estudiamos en un ambiente controlado (experimento) la calidad y eficiencia en el uso de un enfoque de desarrollo software derivado de un MPDS secuencial. De manera complementaria también estudiamos la adopción de un MPDS secuencial en programadores solos y en pareja.

La programación en pareja (en Inglés, Pair Programming o PP) es una práctica común que se usa ya sea de forma independiente o como parte de la metodología conocida como programación extrema (en Inglés, *eXtreme Programming* o XP) [13], en esta práctica, dos programadores trabajan en conjunto en la misma tarea utilizando una computadora. Uno de los programadores (el controlador) escribe el código, mientras que el otro (el observador) revisa activamente el trabajo realizado por el controlador. En esencia, el observador revisa el trabajo por posibles defectos, toma notas, o define estrategias para resolver cualquier problema que pueda surgir en la tarea sobre la que están trabajando.

En esta investigación, las distintas fases que conforman el MPDS secuencial estudiado son abstraídas y operacionalizadas en un experi-mento controlado. El uso de experimentos controlados en la ingeniería de software ayuda a identificar y comprender distintas variables y conexiones que entran en juego en el desarrollo software. En un experimento controlado se modelan las principales características de una realidad (en este caso, la aplicación de un MPDS secuencial) bajo condiciones que pueden controlarse (como pueden ser: el sitio, el diseño del experimento, las características de los participantes, así como materiales, equipos y métodos a emplear), permitiendo así estudiar y comprender mucho mejor esa realidad [14].

El resto de este trabajo está organizado de la siguiente manera: En la segunda sección se presentan algunos modelos de procesos de desarrollo software. En la tercera sección se presenta el contexto del experimento. En la cuarta sección se describe el análisis y resultados. En la quinta sección se discuten los resultados. Finalmente en la sexta sección se presentan las conclusiones.

MODELOS DE PROCESOS DE DESARROLLO SOFTWARE

En esta sección presentamos una síntesis de algunos modelos de proceso de desarrollo software (MPDS) comúnmente conocidos en el ámbito académico e industrial.

Modelo en cascada

El modelo en cascada es un MPDS ampliamente conocido, sus orígenes se remontan a la década de los años setenta [2]. Éste consta de pasos o etapas que se realizan uno a uno de manera secuencial, como se muestran en la Figura 1.

Análisis y definición de requisitos. En esta etapa se llevan a cabo entrevistas a los usuarios, con el fin

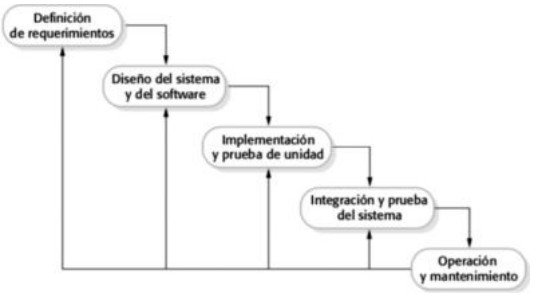

Figura 1. Adaptación del modelo en cascada descrito en [2].

de establecer los servicios, restricciones y metas del producto software a desarrollar. Posteriormente, tales servicios, restricciones y metas son detallados y sirven como especificaciones del producto software a desarrollar.

Diseño. En esta etapa, los requisitos sirven de base para el establecimiento de una arquitectura del producto software. El diseño de software implica identificar y describir las abstracciones necesarias para el producto software y así como sus relaciones.

Implementación y pruebas unitarias. En esta etapa, el diseño se implementa como un conjunto de programas o unidades de programa. Las pruebas unitarias consisten en verificar que cada unidad satisfaga su especificación.

Integración y pruebas de sistema. Durante esta etapa, las unidades de programa o programas son integradas y probadas como un sistema completo para asegurar el cumplimiento de los requisitos software. Una vez probado, el producto software se entrega al cliente.

Operación y mantenimiento. Es usual que esta etapa sea la más larga del ciclo de vida. El producto es instalado y puesto en operación. El mantenimiento consiste en: corregir defectos que no fueron detectados en etapas previas del ciclo de vida; mejorar la implementación del producto, o extender nuevas funcionalidades a partir del descubrimiento de nuevos requisitos.

Este MPDS puede ser usado cuando los requisitos se comprenden bien y estos son poco probables que cambien radicalmente durante el desarrollo del sistema. En el modelo cascada la documentación es producida en cada etapa, por lo que se puede hacer un seguimiento del progreso del desarrollo con respecto a la planificación del proyecto [1].

Uno de los inconvenientes de este modelo es el particionado inflexible del proyecto en distintas etapas. Los compromisos deben ser realizados en una etapa temprana del proceso, lo que dificulta la respuesta a los cambios en los requisitos. Sin embargo, a pesar de este inconveniente, procesos software basados en este MPDS son aún usados de manera habitual, esto debido a que es fácil utilizar un modelo de gestión común para todo el proyecto [1].

Modelo iterativo e incremental

El modelo iterativo e incremental [1] es aquel donde se desarrolla una versión inicial del producto software a través de ciclos repetidos (iterativos) donde a través de incrementos se añaden nuevas funcionalidades. En la Figura 2 se muestra una representación de este modelo.

Figura 2. Adaptación del modelo iterativo e incremental descrito en [1].

Generalmente, en las primeras iteraciones se incluyen las funcionalidades requeridas de mayor urgencia o las más importantes. En el modelo iterativo e incremental, las actividades de especificación, desarrollo y validación ocurren de manera intercalada, logrando una retroalimentación rápida entre estas actividades. Se dice que este modelo presenta algunos beneficios como son [1]:

Reducción del costo de soporte a requisitos cambiantes. El grado de análisis y documentación generada es mucho menor de la que suele requerir el modelo en cascada.

Mayor facilidad de retroalimentación. Los clientes pueden hacer comentarios sobre las funcionalidades implementadas.

Entregas y despliegues más rápidos. Aun cuando el software no implemente toda su funcionalidad especificada es posible comenzar a utilizarlo.

Desde una perspectiva de gestión, este modelo presenta los siguientes inconvenientes:

El proceso de desarrollo no es visible. Dado que la especificación, diseño, implementación y verificación se realiza de forma intercalada, es difícil gestionar del proceso de desarrollo.

Poca documentación. Dados los ciclos cortos que hay entre las distintas actividades asociadas al desarrollo del producto, la documentación producida suele ser escasa.

La estructura del sistema tiende a degradarse con nuevos incrementos. A menos que se inviertan recursos en la refactorización para mejorar el producto software, los cambios regulares tienden a corromper la estructura del producto, por lo que incorporar nuevos cambios o funcionalidades se hace cada vez más laborioso y costoso.

El modelo iterativo e incremental de alguna forma es ahora un método común para el desarrollo de productos software. Este modelo suele aplicarse en las metodologías ágiles de desarrollo de software.

Modelo en espiral

El modelo en espiral [3] representa al proceso software como una espiral en lugar de una secuencia de actividades. Cada iteración en la espiral representa una etapa del proceso de desarrollo. La primera iteración se relaciona con la factibilidad del producto a desarrollar, la siguiente iteración con la definición de los requisitos, la siguiente con el diseño del producto, la otra con la codificación y verificación, y así sucesivamente. En la Figura 3 se muestra la representación de este modelo.

En este modelo, los cambios son considerados como parte de los riesgos del proyecto, por lo que en éste se definen actividades de gestión del riesgo para minimizarlos. Cada iteración de la espiral se divide en los siguientes cuadrantes:

Definición de objetivos. En este cuadrante se definen los objetivos específicos, se identifican las restricciones del proceso y del producto, y se

Figura 3. Adaptación del modelo en espiral descrito en [3].

elabora un plan de gestión detallado; así mismo se identifican los riegos del proyecto así como estrategias de solución alternativas.

Evaluación y reducción de riegos. En este cuadrante se analiza a detalle cada uno de los riesgos identificados en el proyecto, esto con el fin de tomar medidas para reducirlos.

Desarrollo y validación. En este cuadrante se elige una estrategia de desarrollo considerando los riesgos identificados. Por ejemplo, si predominan los riegos asociados a la interfaz del usuario, el desarrollo por prototipos puede ser la mejor opción. Si los riesgos en la seguridad es la principal preocupación, el desarrollo puede realizarse basado en trasformaciones formales. Por el contrario, si la integración de subsistemas es la principal causa de riesgo, el modelo en casaca podría ser el mejor modelo en este caso.

Planificación. En este último cuadrante se revisa el proyecto y se toma la decisión de continuar o no con una nueva iteración en la espiral. Si se decide continuar, se elaboran planes para la siguiente fase del proyecto.

Modelo V

El modelo V [4] es una adaptación del modelo en cascada en el que se hace énfasis en la verificación del producto software en cada una de las etapas de desarrollo. En la Figura 4 se muestra la representación de este modelo.

Este modelo es representado con la letra mayúscula V, iniciando con las necesidades del usuario en el extremo izquierdo y finalizando con un sistema

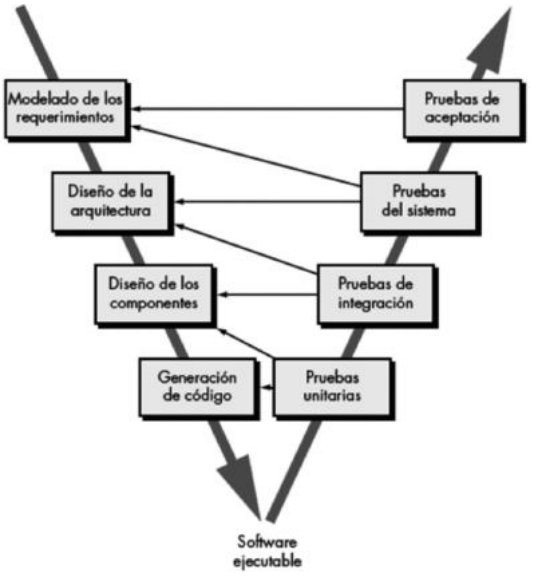

Figura 4. Adaptación del modelo en espiral descrito en [4].

validado por el usuario en el extremo derecho. En el costado izquierdo, la descomposición y la definición descienden como en el modelo en casada, mientras que en el costado derecho, la integración y verificación ascienden sucesivamente pasando por los diferentes niveles, es decir, en el ascenso se verifican los conjuntos, unidades, componentes, subsistemas y finalmente el sistema (producto) completo.

Modelo RUP

El modelo de proceso unificado (en Inglés, Rational Unified Process o RUP) [5] es un modelo derivado del lenguaje de modelado unificado (en Inglés, *Unified Modeling Language* o UML) y del proceso de desarrollo software unificado [15-16]. En la Figura 5 se presenta la estructura de este modelo.

Este modelo identifica cuatro etapas:
Inicio. En esta etapa se evalúa la factibilidad del producto a desarrollar desde la perspectiva de su funcionalidad, impacto en el negocio y valor financiero. Si el aporte del producto al negocio es marginal, éste puede no ser desarrollado.

Elaboración. Esta etapa tiene como finalidad comprender el dominio del problema, construir una arquitectura del producto, desarrollar un plan del proyecto, e identificar los principales riesgos del proyecto. Los artefactos software producidos en esta etapa son: el modelo de requisitos para el producto, los cuales pueden ser casos de uso, la arquitectura y el plan del desarrollo.

Construcción. Esta etapa incluye el diseño, la programación y la verificación. Durante esta etapa, algunas partes pueden desarrollarse en paralelo e integrarse. Al finalizar esta etapa, se debe tener un producto funcionando con su documentación correspondiente.

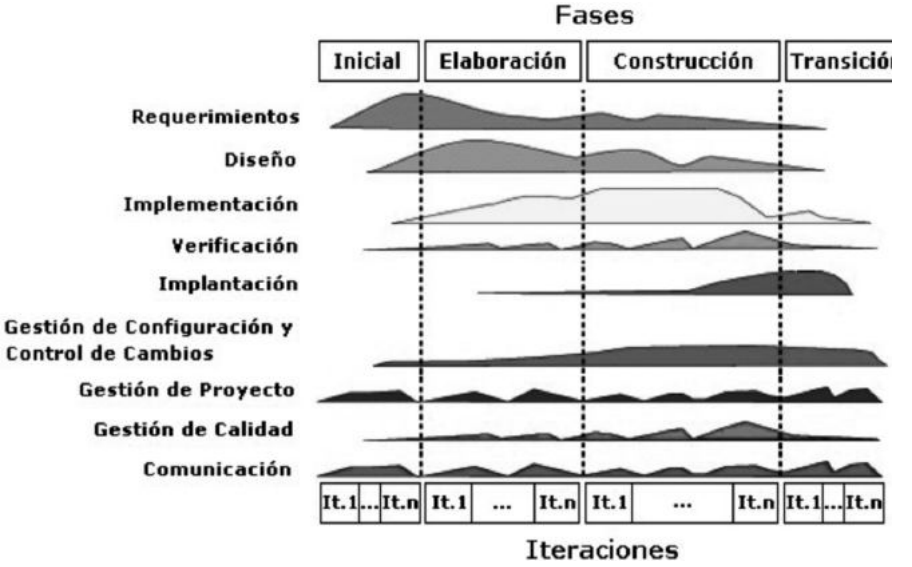

Figura 5. Adaptación del modelo RUP descrito en [5].

Transición. Esta etapa final se ocupa de trasladar el producto del ambiente de desarrollo, al ambiente de producción. Este movimiento es ignorado en la mayoría de los modelos de proceso, pero es, de hecho, una actividad costosa y muchas veces problemática. Al término de esta fase, se debe tener un producto software documentado, que funciona de manera correcta en el ambiente de producción.

En el modelo RUP se recomiendan algunas prácticas fundamentales como son:

Desarrollo de forma iterativa. En las etapas tempranas del desarrollo se planifican los incrementos del sistema considerando las funcionalidades prioritarias para el usuario.

Gestión de requisitos. Se documentan los requisitos del cliente y se realiza un seguimiento de los cambios en estos. Se analiza el impacto de los cambios antes de efectuarlos.

Arquitectura basada en componentes. Se organiza la arquitectura del producto en módulos.

Modelado visual. Se emplea el lenguaje de modela unificado para presentar las vistas estáticas y dinámicas del producto a desarrollar.

Verificación. Se vigila que el producto satisfaga los estándares de calidad previamente definidos.

Control de cambios. Se gestionan los cambios en el producto utilizando un sistema de gestión del cambio, así como procedimientos y herramientas de gestión de la configuración.

En la Tabla 1 se presentan las principales ventajas y desventajas de los modelos antes descritos.

CONTEXTO DEL EXPERIMENTO

Una vez descritos algunos de los MPDS, a continuación presentamos el contexto del experimento

Tabla 1. Estadística descriptiva con relación a la calidad.

Modelo	Ventajas	Desventajas
Cascada	– Mayor control en la gestión del proyecto. – Comúnmente utilizado	– Su rigidez (las etapas deben realizarse de manera secuencial). – Costo de implementación.
Iterativo e incremental	– La documentación generada es menor. – El cliente tiene mayor acceso a las funcionalidades implementadas. – Es posible priorizar las funcionalidades a entregar.	– Dificultad en la gestión del proyecto al solaparse las etapas del desarrollo o mantenimiento. – Poca documentación generada. – El exceso de cambios puede corromper la arquitectura diseñada del producto.
Espiral	– Mayor control en cambios a requerimientos. – Dado su particionamiento, es más fácil estimar los costos. – Mayor satisfacción del cliente debido a una mayor retroalimentación.	– Recomendado sólo para proyectos grandes con requerimientos volátiles. – Se requiere de experiencia para llevar a cabo la identificación de riesgos. – Costoso y requiere de mayor tiempo para su implementación.
Modelo V	– Sencillo y fácil de seguir. – Detección temprana de defectos. – Mayor control con respecto al proceso de verificación y validación del producto.	– Se asume que los requerimientos están claramente definidos. – Rigidez, no se puede regresar a etapas previas. – Costoso, dado que se deben efectuar procesos de verificación y validación en cada etapa.
RUP	– Es configurable de acuerdo a las necesidades de cada proyecto. – Las etapas de desarrollo no se asocian a flujos de trabajo específicos, por lo que distintos flujos de trabajo pueden coincidir en una misma etapa.	– Se requiere experiencia para su implementación. – En proyectos pequeños resulta costoso de implementar.

realizado. La definición del experimento aquí reportado se describe de acuerdo al enfoque Goal-Question-Metric [17]. Este enfoque ayuda a identificar el objeto de estudio, el propósito, enfoque de calidad, perspectiva y contexto de un experimento determinado. La definición del experimento aquí reportado se describe de la siguiente manera:

"Estudiar si el uso de un enfoque de desarrollo software secuencial (del tipo cascada) incide en la calidad y eficiencia de programas codificados por programadores solos y en pareja. El estudio se realiza a través de un experimento controlado desde el punto de vista del investigador en un contexto académico. Este contexto está compuesto por alumnos universitarios en su tercer año inscritos en un curso de Ingeniería de Software. Estos alumnos codificarán en pareja o individualmente varios programas usando uno de dos enfoques de desarrollo software: secuencial o ad-hoc, donde el enfoque ad-hoc es empleado como grupo de control".

La pregunta de investigación (RQ) planteada en este experimento se define de la siguiente manera: ¿Es la calidad y eficiencia de los programas codificados afectada por el uso de un enfoque de desarrollo software secuencial aplicado en programadores solos y en pareja?

Esta pregunta de investigación deriva en las siguientes hipótesis nulas:

H_{0a}: La calidad, medida como el número de defectos inyectados por cada 100 LOCs, no es afectada por el enfoque de desarrollo software (secuencial o ad-hoc).

H_{0b}: La calidad no es afectada por el tipo de programación empleado (programación en pareja o individual).

H_{0c}: La calidad no es afectada por los programas codificados durante el experimento.

H_{0d}: La calidad no es afectada por las siguientes relaciones: enfoque de desarrollo software y tipo de programación; enfoque de desarrollo software y programas codificados; tipo de programación y programas codificados; enfoque de desarrollo software, tipo de programación y programas codificados.

H_{0e}: La eficiencia, medida como el número de compilaciones y ejecuciones por hora empleadas en un entorno de desarrollo integrado (en Inglés, Integrated Development Environment, IDE), no es afectada por el enfoque de desarrollo (secuencial o ad-hoc).

H_{0f}: La eficiencia no es afectada por el tipo de programación empleado (programación en pareja o individual).

H_{0g}: La eficiencia no es afectada por los programas codificados en el experimento.

H_{0h}: La eficiencia no es afectada por las siguientes relaciones: enfoque de desarrollo software y tipo de programación; enfoque de desarrollo software y programas codificados; tipo de programación y programas codificados; enfoque de desarrollo software, tipo de programación y programas codificados.

Diseño experimental

Las hipótesis antes planteadas se contrastarán a través de las mediciones recolectadas pertenecientes a los participantes de este experimento. Las mediciones a recolectar pertenecen a cuatro grupos que son: (1) Participantes trabajando en parejas usando un enfoque de desarrollo secuencial, (2) Participantes trabajando en parejas siguiendo un enfoque de desarrollo ad-hoc, (3) participantes trabajando de manera individual usando un enfoque secuencial, y (4) participantes trabajando de manera individual siguiendo un enfoque ad-hoc.

Con el objetivo de recoger el número máximo de mediciones se empleó un diseño experimental factorial con medidas repetidas. Los diseños factoriales permiten el estudio de varios factores y la interacción entre estos [18]. En este experimento, se definió un diseño factorial 2^2 donde los factores principales se asocian con el enfoque de desarrollo (secuencial y ad-hoc) así como el tipo de programación (programación en pareja e individual). Este diseño es repetido en la mismas unidades experimentales (en este caso, los programadores en pareja y programadores individuales) pero variando los programas a codificar. Obtener mediciones repetidas en la misma unidad experimental es más eficiente en términos del uso de recursos que obtener mediciones recolectadas de diferentes unidades experimentales. Otra cualidad de este diseño experimental es la disminución de la varianza de los estimadores, lo cual permite que las inferencias estadísticas sean realizadas con un número menor de participantes [18].

Participantes, tareas y objetos

Estudiantes de tercer año inscritos en el curso de Ingeniería de Software I de la carrera de Ingeniería en

Sistemas Informáticos de la Facultad de Informática y Electrónica (FIE) de la Escuela Superior Politécnica de Chimborazo (ESPOCH) participaron en este experimento. Dicho experimento se llevó a cabo en uno de los laboratorios de cómputo de la FIE en noviembre de 2015. En este experimento, 23 estudiantes (8 parejas y 7 programadores solos) participaron y finalizaron de manera satisfactoria la codificación de los programas asignados. De acuerdo a la clasificación sobre experiencia en programación creada en [19], los participantes de este experimento pueden clasificarse como novicios avanzados, es decir, cuentan con experiencia práctica y conocimiento sobre los aspectos fundamentales sobre algún lenguaje de programación. Los participantes firmaron un formulario de consentimiento en el que estuvieron de acuerdo en participar de forma voluntaria en este estudio, por lo que información privada o sensible sobre los participantes no fue recolectada a fin de garantizar el anonimato.

Los participantes fueron asignados al azar en uno de los cuatro grupos de acuerdo al diseño factorial 2^2 empleado: (1) grupo de parejas trabajando con el enfoque secuencial, (2) grupo de parejas trabajando con el enfoque ad-hoc, (3) grupo de solos trabajando con el enfoque secuencial, y (4) grupo de solos trabajando con el enfoque ad-hoc. Siguiendo el diseño de medidas repetidas, todos los participantes codificaron dos programas en dos sesiones diferentes.

Antes de realizar el experimento, a los estudiantes quienes de manera aleatoria trabajaron con un enfoque de desarrollo secuencial se les explicó cómo debieran codificar sus programas. Esto es, siguiendo un MPDS secuencial, en donde los estudiantes primero elaboran un diseño acorde a la especificación, después codifican el diseño, una vez codificado el programa los estudiantes compilan y ejecutan el programa, para posteriormente pasar a la etapa de verificación, donde en este etapa los estudiantes corrigen su programa hasta que el comportamiento observado de éste es acorde a su especificación. Al resto de estudiantes se les dijo que usaran su propio enfoque de programación, identificado como ad-hoc. Este enfoque se aproxima a un enfoque de desarrollo iterativo en donde los estudiantes de forma iterativa escriben líneas de código, compilan, ejecutan y verifican el programa, repitiendo esta secuencia hasta que el programa realiza la funcionalidad requerida acorde a su especificación.

Todos los participantes atendieron una sesión en donde se les habló sobre la programación en pareja. En esta sesión, los estudiantes recibieron los principales conceptos de esta práctica y cómo aplicarla. En otra sesión, los estudiantes reforzaron los conceptos de programación en pareja y codificación de un programa en modo consola con el lenguaje de programación de su preferencia (en este caso, C#).

Finalmente, se les enseñó a los estudiantes cómo recolectar las mediciones durante las sesiones del experimento. Este procedimiento de recolección consistió en que los estudiantes registraran el número de compilaciones (y ejecuciones) en el IDE y que registraran los defectos lógicos que inyectaron durante la codificación. Se les explicó a los estudiantes dos tipos básicos de defectos que pudieran cometer: defectos de sintaxis y defectos lógicos. En un defecto de sintaxis, los programas no pueden ser compilados de manera satisfactoria. Por otra parte, en un defecto lógico, los programas pueden ser compilados y ejecutados, pero éstos no son capaces de efectuar la funcionalidad esperada de acuerdo a su especificación. Como herramienta de soporte, los estudiantes (programadores en pareja y solos) utilizaron el entorno de desarrollo integrado (IDE) Visual Studio con el lenguaje de programación C#. Se emplearon formularios impresos para el registro de mediciones como son: el número de compilaciones y los defectos inyectados.

Previo a las sesiones experimentales, se llevó a cabo una fase de entrenamiento en la que asistieron todos los participantes del experimento. Las sesiones de entrenamiento tienen como finalidad incrementar la precisión en las mediciones recabadas. Esta fase de entrenamiento consistió en una sesión en la cual los participantes codificaron un programa distinto a los empleados en las sesiones experimentales (programa que calcula la desviación estándar dado un conjunto de números enteros). Esta fase de entrenamiento permitió a los programadores en pareja aplicar los conceptos previamente suministrados y adquirir experiencia y cohesión en la práctica de la programación en pareja.

En cada sesión experimental los participantes codificaron, compilaron, ejecutaron y verificaron

dos programas en modo consola de acuerdo a su especificación. El primer programa (identificado como conversor, o programa A, convierte números romanos a números decimales y viceversa. El segundo programa (identificado como histograma, o programa B, representa con caracteres determinados los conteos de números positivos, negativos y ceros en un determinado conjunto de números enteros. Los programas utilizados en este experimento forman parte de los materiales utilizados en otros experimentos (familia de experimentos) afines a la programación en pareja y que están reportados en [20, 34-36].

Ejecución

Durante la ejecución, las unidades experimentales (programadores en pareja y solos) trabajaron con la misma combinación de tratamientos (enfoque de programación y tipo de programación) tanto en las sesiones de entrenamiento como en las sesiones del experimento, variando únicamente el programa a codificar en cada sesión.

En cada sesión se asignó un tiempo de 120 minutos donde en cada sesión los participantes recibieron una serie de instrucciones así como la especificación del programa a codificar. En la primera sesión experimental, los participantes codificaron el programa A (conversor), mientras que en la segunda sesión los participantes codificaron el programa B (histograma).

Los programas codificados en cada sesión experimental se consideraron como completos o finalizados una vez que el instructor evaluó su cumplimiento de acuerdo a su especificación.

Métricas

Las métricas con respecto a los constructos calidad y eficiencia se definieron como: el número de defectos inyectados por cada 100 LOCs (calidad del producto) y número de ejecuciones por hora efectuadas en el entorno de desarrollo integrado (eficiencia). Nótese que la evaluación de la calidad del producto está basada en una métrica externa, como se sugiere en [21], dado que la evaluación de la calidad del producto basada en métricas internas puede llevar a resultados poco confiables como se menciona en [22].

Al finalizar las sesiones experimentales, se tomaron en cuenta las mediciones de 23 participantes. Las mediciones incompletas ocurrieron en dos unidades experimentales (dos programadores solos) debido a que los participantes conformados en estas unidades no atendieron todas las sesiones del experimento, estas dos unidades experimentales se descartaron con el objetivo de ajustarse a los lineamientos del diseño experimental usado. Éste diseño (factorial con medidas repetidas) requiere recoger todas las mediciones repetidas de las mismas unidades experimentales.

ANÁLISIS Y RESULTADOS

Una vez recolectadas las mediciones del experimento, en esta sección se presenta el análisis descriptivo realizado así como el análisis estadístico inferencial.

Análisis descriptivo con respecto a la calidad

En la Tabla 2 se presenta la estadística descriptiva (tamaño de muestra, valor promedio, desviación estándar, valor mínimo y máximo) con respecto a la calidad medida como el número de defectos inyectados por cada 100 líneas de código. El análisis descriptivo se agrupa de acuerdo a los distintos factores: enfoque de desarrollo, tipo de programación y programas codificados. De acuerdo a los resultados de la Tabla 2, se observa que en el enfoque de

Tabla 2. Estadística descriptiva con relación a la calidad.

Factor	Enfoque de desarrollo		Tipo de Programa		Programa	
Nivel	Ad-hoc	Secuencial	Pareja	Solo	Conversor	Histograma
n	14	16	16	14	15	15
Media	2.71	2.10	2.50	2.25	2.35	2.42
Dev. Est.	1.40	1.47	1.36	1.59	1.56	1.38
Mín.	0.42	0	0	0	0.42	0
Máx.	5.97	4.65	4.65	5.97	5.97	4.65

desarrollo ad-hoc los participantes ligeramente tienden a inyectar más defectos que en el enfoque secuencial. En cuanto al tipo de programación, se observa que los programadores en pareja ligeramente inyectaron más defectos que los programadores solos. Referente al programa codificado, se observa que los participantes ligeramente inyectaron más defectos en el programa Histograma.

Análisis descriptivo con respecto a la eficiencia
En la Tabla 3 se presenta la estadística descriptiva (tamaño de muestra, valor promedio, desviación estándar, valor mínimo y máximo) con respecto a la eficiencia medida como el número de compilaciones y ejecuciones realizadas por hora. El análisis descriptivo se agrupa de acuerdo a los distintos factores: enfoque de desarrollo, tipo de programación y programas codificados.

De acuerdo a la información de la Tabla 3 se observa que, en promedio, los participantes que usaron un enfoque de desarrollo ad-hoc efectuaron un mayor número de compilaciones y ejecuciones en el IDE a diferencia de quienes utilizaron un enfoque de desarrollo secuencial. Se observa también que los participantes que trabajaron en parejas ligeramente realizaron más compilaciones y ejecuciones en el IDE. En cuanto el programa, se observa que en la codificación del programa histograma los participantes emplearon un número mayor de compilaciones.

Análisis inferencial
Una vez descrito el análisis descriptivo, en este apartado se presenta el análisis inferencial usado para contrastar las hipótesis previamente definidas.

El modelo estadístico empleado de acuerdo al diseño factorial seleccionado se define en la ecuación (1) descrita.

$$y_{ijkl} = \mu + \alpha_i + \beta_j + \delta_{ijl} + \gamma_k + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\beta\gamma)_{jk} + (\alpha\beta\gamma)_{ijk} + \varepsilon_{ijkl} \quad (1)$$

En esta ecuación, μ es la media general, α_i es el efecto del *i*-ésimo tratamiento (enfoque de desarrollo), β_j es el efecto de *j*-ésimo tratamiento (tipo de programación), δ_{ijl} es el error experimental aleatorio para las unidades experimentales (parejas y solos) dentro de los tratamientos (enfoque de desarrollo y tipo de programación) con varianza σ^2_d, γ_k es el efecto del *k*-ésimo programa, $(\alpha\beta)_{ij}$ es la interacción entre el enfoque de desarrollo y el tipo de programación, $(\alpha\gamma)_{ik}$ es la interacción entre el enfoque de desarrollo y el programa, $(\beta\gamma)_{jk}$ es la interacción entre el tipo de programación y el programa, y $(\alpha\beta\gamma)_{ijk}$ es la interacción entre el enfoque de desarrollo, tipo de programación y programa codificado. Finalmente, ε_{ijkl} es el error experimental aleatorio distribuido normalmente en las medidas repetidas con varianza σ^2_e.

Los componentes del modelo estadístico antes descrito son evaluados mediante el análisis de la varianza (ANOVA). ANOVA prueba estadísticamente si los promedios de varios grupos de mediciones son iguales. La hipótesis nula asume que todos los grupos son simplemente muestras aleatorias de la misma población. Esto implica que todos los tratamientos tienen el mismo efecto (quizás ninguno). Rechazar la hipótesis nula implica que los diferentes tratamientos arrojen como resultado efectos diferentes. Para el análisis ANOVA se usó el paquete ez [23] del entorno estadístico R [24]. Este paquete implementa una función para el análisis de diseños factoriales (en este caso el enfoque de programación y el tipo de programación) con medidas repetidas (los dos programas).

ANOVA con respecto a la calidad
En la Tabla 4 se muestra el resultado del ANOVA con respecto a la calidad del producto (número

Tabla 3. Estadística descriptiva con respecto a la eficiencia.

Factor	Enfoque de desarrollo		Tipo de Programa		Programa	
Nivel	Ad-hoc	Secuencial	Pareja	Solo	Conversor	Histograma
n	14	16	16	14	15	15
Media	8.86	3.87	6.63	5.48	3.99	8.42
Dev. Est.	4.9	2.48	5.17	3.68	2.6	4.99
Mín.	2.73	0.92	1.54	0.92	0.92	2.09
Máx.	18.86	9.23	18.86	11.25	9.55	18.86

de defectos inyectados por cada 100 líneas de código). Como se muestra en los resultados de la Tabla 4, ninguno de los componentes arrojó alguna diferencia significativa de acuerdo a un nivel alfa de 0.05. Estos resultados sugieren que los factores: enfoque de desarrollo, tipo de programa, programa así como sus interacciones no influyen en la tasa de defectos inyectados.

ANOVA con respecto a la eficiencia

En la Tabla 5 se presentan los resultados del análisis de la varianza (ANOVA) con respecto a la eficiencia (número de compilaciones y ejecuciones por hora).

Como se observa en la Tabla 5, los factores enfoque de desarrollo y programa codificado muestran diferencias significativas en un nivel alfa de 0,001 (***) y 0,01 (**), respectivamente. Los participantes quienes usaron un enfoque de desarrollo secuencial en promedio realizaron un número significativamente menor de compilaciones y ejecuciones en el IDE (3.87) a diferencia de los participantes que usaron un enfoque ad-hoc quienes en promedio realizaron un número significativamente mayor de compilaciones en el IDE (8,86) (ver análisis descriptivo en Tabla 3).

En cuanto al programa codificado, el programa identificado como Conversor implicó un número significativamente menor de compilaciones en el IDE (3,99) que el programa Histograma (8,42).

Validación del modelo estadístico

El modelo estadístico utilizado en este diseño experimental fue validado contra los supuestos de normalidad, aleatoriedad y esfericidad.

El supuesto de normalidad se examinó a través del uso de dos gráficos Q-Q (uno por cada aspecto estudiado). El gráfico de probabilidad normal (o gráfico Q-Q) es una herramienta que ayuda a contrastar gráficamente dos distribuciones, la empírica proveniente de una muestra de datos (en este caso los residuos normalizados del modelo estadístico empleado) con respecto a la distribución normal [25-26]. Cuanto más se asemejen los residuos a una recta (bisectriz de los ejes de coordenadas), más se aproximarán a una distribución normal. En las Figuras 6 y 7 se muestran los gráficos Q-Q resultantes.

Como se observa en los gráficos de ambas figuras, los residuos se distribuyen a lo largo de la bisectriz

Tabla 4. ANOVA para la calidad del producto.

Componente	DFn	DFd	F	P	$p< 0.05$
Enfoque de desarrollo	1	11	0,9422	0,3526	
Tipo de programación	1	11	0,1129	0,7432	
Programa	1	11	0,0164	0,9005	
Enfoque de des.: Tipo de prog.	1	11	0,5730	0,4650	
Tipo de prog.: Programa	1	11	1,2685	0,2840	
Enfoque de des.: Programa	1	11	0,0110	0,9182	
Enfoque de des.: Tipo de prog.: Programa	1	11	0,0162	0,9011	

Tabla 5. ANOVA para la eficiencia.

Componente	DFn	DFd	F	P	$p< 0.05$
Enfoque de desarrollo	1	11	25,84763	0,0004	***
Tipo de programación	1	11	1,0621	0,3249	
Programa	1	11	14,7270	0,0028	**
Enfoque de des.: Tipo de prog.	1	11	0,6586	0,4343	
Tipo de programación: Programa	1	11	1,9275	0,1925	
Enfoque de des: Programa	1	11	1,2924	0,2798	
Enfoque de des.:Tipo de prog.: Programa	1	11	2,4409	0,1465	

de los ejes por lo que se asume el supuesto de normalidad.

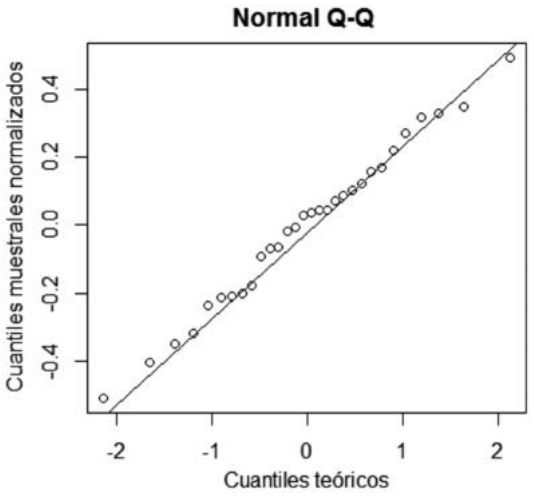

Figura 6. Gráfico Q-Q de probabilidad normal con respecto a la calidad.

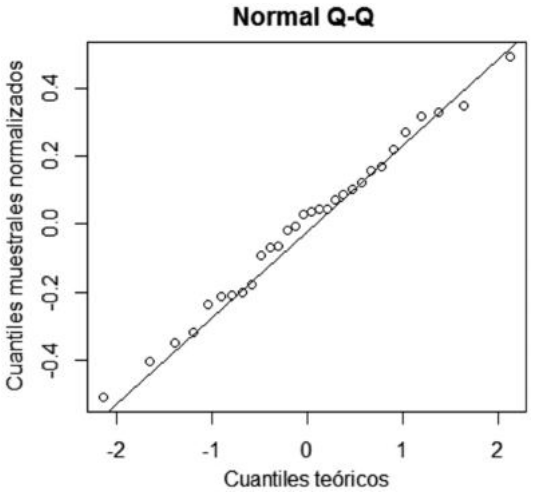

Figura 7. Gráfico Q-Q de probabilidad normal con respecto a la eficiencia.

Con respecto al supuesto de aleatoriedad, las mediciones recogidas corresponden a muestras aleatorias e independientes. Por otra parte, el supuesto de esfericidad que examina la semejanza de varianzas en los dos niveles de los factores intra-sujetos (factores repetidos dentro de la misma unida experimental). Este supuesto siempre se cumple cuando se emplean sólo dos niveles en factores intra-sujetos [27], es decir, dos mediciones repetidas. En el experimento aquí presentado únicamente se emplearon dos mediciones repetidas (programa conversor y programa histograma), por lo tanto se asume la esfericidad en el modelo estadístico empleado.

DISCUSIÓN

Una vez presentados los resultados del experimento, estos son discutidos con respecto a las hipótesis definidas en la Sección 2. Con respecto a la calidad (número de defectos inyectados por cada 100 LOCs), ésta no impacta en el enfoque de programación empleado (se ha fallado rechazar H_{0a}). El tipo de programación (en pareja o individual) no afecta la calidad de los programas codificados (se ha fallado rechazar H_{0b}). Los programas codificados no influyen en la calidad (se ha fallado rechazar H_{0c}). Las relaciones entre los factores: enfoque de programación, tipo de programación y programa codificado no influyen en la calidad (se ha fallado rechazar H_{0d}).

En relación con la eficiencia (número de compilaciones por hora), observamos que ésta sí influye de acuerdo al enfoque de programación usado (secuencial o ad-hoc) (H_{0e} es rechazada). El tipo de programación no influye en la eficiencia (se ha fallado rechazar H_{0f}). El programa codificado sí influye en la eficiencia (H_{0g} es rechazada). Las diferentes relaciones entre el enfoque de programación, tipo de programación y programa codificado no influyen en la eficiencia (se ha fallado en rechazar H_{0h}).

De acuerdo a la evidencia observada en este experimento observamos que la eficiencia es afectada por el uso de un enfoque de programación secuencial aplicado en programadores solos y en pareja. No obstante, en cuanto a la calidad, el enfoque de programación no parece influir, ambos enfoques arrojaron niveles de calidad similares (RQ).

Con respecto a investigaciones relacionadas con la programación en pareja, nuestros resultados sobre la calidad (con respecto a la tasa de defectos inyectados) son similares a los encontrados en [28], donde los autores tampoco encontraron diferencias en cuanto al tipo de programación (en pareja o individual). Sin embargo, nuestros resultados van en sentido opuesto a los reportados en [29-30]. Esta situación de aparente discordancia podría

explicarse por la complejidad o sencillez de los programas (Conversor e Histograma) asignados a los participantes (pares y solos). De acuerdo con [31], los programadores novicios que trabajan en pareja pueden aumentar la calidad del producto cuando desarrollan software con cierto grado de complejidad. Probablemente, dado que nuestros ejercicios de programación fueron concebidos con un grado de complejidad bajo observamos un desempeño similar entre los programadores en pareja y programadores solos.

LIMITACIONES

En cuanto a las limitaciones de la presente investigación es importante notar que los estudios empíricos (como el aquí reportado) suelen estar sujetos a diferentes tipos de amenazas a la validez [32]. A continuación detallamos cómo abordamos diferentes tipos de amenazas a la validez.

Validez de conclusión. Las mediciones recolectadas en nuestro experimento satisfacen los supuestos de normalidad, aleatoriedad y esfericidad.

Validez interna. Los participantes fueron asignados de manera aleatoria a los tratamientos, con lo cual se mitigó el efecto de aprendizaje. El uso de diferentes sesiones de entrenamiento y sesiones experimentales redujeron el aburrimiento o la fatiga. Los participantes trabajaron en el mismo laboratorio de cómputo bajo las mismas condiciones sin interacción entre otras unidades experimentales (parejas o solos).

Validez de constructo. Los participantes recibieron entrenamiento tanto en la programación en pareja así como en un enfoque de desarrollo software basado en un MPDS secuencial. Los constructos de causa y efecto estudiados fueron operacionalizados de la misma manera que en otros estudios relacionados.

Validez externa. El uso de estudiantes como participantes en experimentos controlados en lugar de profesionales pudo haber afectado esta validez; sin embargo, como se menciona en [33], el uso de estudiantes como participantes permite obtener evidencia empírica preliminar para confirmar o refutar hipótesis que pueden ser contrastadas posteriormente en ambientes industriales. Con esta estrategia se atiende la situación de la generalización que considera esta validez.

CONCLUSIONES

En esta investigación hemos realizado un experimento controlado para estudiar la calidad y eficiencia de un enfoque de desarrollo basado en un modelo de proceso de desarrollo software (MPDS) secuencial aplicado por programadores solos y en pareja.

En cuando a la calidad (número de defectos inyectados por cada 100 LOCs), nuestros resultados sugieren que el uso de un enfoque de desarrollo secuencial es equivalente al uso de un enfoque de desarrollo ad-hoc. En cuanto al tipo de programación, no observamos alguna diferencia significativa entre programadores en pareja y programadores solos. Respecto al tipo de programa codificado, tampoco observamos diferencias entre los dos programas codificados.

Con relación a la eficiencia (número de compilaciones por hora), nuestros resultados sugieren una diferencia significativa a favor del uso de un enfoque de desarrollo software secuencial. Los participantes quienes emplearon un enfoque de desarrollo ad-hoc requirieron poco más del doble de compilaciones y ejecuciones para completar los programas asignados (2.3 veces más). En cuanto al tipo de programación, no observamos diferencias significativas entre programadores solos y en pareja. No obstante observamos una diferencia significativa con el tipo de programa codificado. El programa identificado como histograma requirió poco más del doble de compilaciones y ejecuciones para ser completado (2.1 veces más).

Como conclusión de acuerdo a la evidencia observada, el uso de un MPDS secuencial puede resultar benéfico dependiendo del contexto en el que se implemente. En nuestro contexto, observamos una mayor concentración en los participantes quienes aplicaron el enfoque de desarrollo basado en un MPDS secuencial. Haber seguido un enfoque de desarrollo de forma secuencial y ordenada incidió de manera significativa en la eficiencia.

Desde una perspectiva académica, el uso de un enfoque de desarrollo software secuencial puede resultar benéfico en estudiantes inscritos en carreras afines con el desarrollo software, dado que este enfoque incentiva mayor disciplina y una mayor concentración en las distintas etapas involucradas en el desarrollo software.

REFERENCIAS

[1] I. Sommerville. "Software Engineering". Pearson. 9th ed. Boston, USA. 2010.

[2] W.W. Royce. "Managing the development of large software systems". Proc.Westcon. IEEE CS Press. pp. 328-339. 1970

[3] B.W. Boehm. "A Spiral Model of Software Development and Enhancement". IEEE Computer. Vol. 21 Nº 5, pp. 61-72. 1988.

[4] K. Forsberg and H. Mooz. "The relationship of system engineering to the project cycle". National Council On Systems Engineering (NCOSE) and American Society for Engineering Management (ASEM). Vol. 1 Nº 1, pp. 57-65. 1991.

[5] P. Kruchten. The Rational Unified Process-An Introduction. Addison-Wesley. 3rd Edition. Reading, USA. 2003.

[6] N.M.A. Munassar and A. Govardhan. "A comparison between five models of software engineering". International Journal of Computer Science Issues. Vol. 7 Nº 5, pp. 94-101. 2010.

[7] S. Balaji and M.S. Murugaiyan. "Wateerfall vs V-Model vs Agile: a Comparative Study on SDLC". International Journal of Information Technology and Business Management. Vol. 2 Nº 1, pp. 26-29. 2012.

[8] G. Cugola and C. Ghezzi. "Software Processes: A Retrospective and a Path to the Future". Software Process: Improvement and Practice. Vol. 4 Nº 3, pp. 101-123. 1998.

[9] J. Nandhakumar and D.E. Avison. "The fiction of methodological development: a field study of information systems development". Information Technology & People. Vol. 12 Nº 2, pp. 176-191. 1999.

[10] P.A. Laplante and J. Neill. "The Demise of the Waterfall Model Is Imminent". ACM Queue. Vol. 1 Nº 10, pp. 10-15. 2004.

[11] D.L. Parnas and P.C. Clements. "A Rational Design Process: How and Why to Fake It". IEEE Transactions on Software Engineering. Vol. 12 Nº 2, pp. 251-257. 1986.

[12] C.L. Larman and V.R. Basili. "Iterative and Incremental Development: A Brief History". Computer. Vol. 36 Nº 6, pp. 47-56. 2003.

[13] K. Beck. "Extreme programming explained: embrace change". Addison-Wesley Longman Publishing Co., Inc. Boston, USA. 2000.

[14] O.S. Gómez, J.P. Ucán y G. Gómez. "Aplicación del proceso de experimentación a la ingeniería de software". Abstraction & Application. Vol. 8, pp. 26-37. 2013.

[15] J. Rumbaugh, I. Jacobson and G. Booch. "The Unified Software Development Process". Addison-Wesley. Reading, USA. 1999.

[16] J. Arlow and I. Neustadt. "UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design". Addison-Wesley. 2nd Edition. Boston, USA. 2005.

[17] V.R. Basili, G. Caldiera and H.D. Rombach. "Goal question metric paradigm". Encyclopedia of Software Engineering. John Wiley & Sons. 1994.

[18] R. Kuehl. "Design of Experiments: Statistical Principles of Research Design and Analysis". Duxbury Thomson Learning. 2nd Ed. Pacific Grove, USA. 2000.

[19] H.L. Dreyfus and S.E. Dreyfus. "Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer". Free Press. New York, USA. 1986.

[20] O.S. Gómez, J.L. Batún and R.A. Aguilar. "Pair versus solo programming - an experience report from a course on design of experiments in software engineering". International Journal of Computer Science Issues. Vol. 10 Nº 1, pp. 734-742. 2013.

[21] N. Salleh, E. Mendes and J. Grundy. "Empirical studies of pair programming for cs/se teaching in higher education: A systematic literature review". IEEE Transactions on Software Engineering. Vol. 37 Nº 4, pp. 509-525. 2011.

[22] J. Vanhanen and C. Lassenius. "Effects of pair programming at the development team level: an experiment". International Symposium on Empirical Software Engineering. IEEE CS Press. Pp. 336-345. 2005.

[23] M.A. Lawrence. "ez: Easy analysis and visualization of factorial experiments". R Package Version 4.4-0. 20 November 2016. URL: http://CRAN.RProject. org/package=ez

[24] R. Core Team. "The R Project for Statistical Computing". 20 November 2016. URL: https://www.r-project.org/.

[25] M.B. Wilk and R. Gnanadesikan. "Probability Plotting Methods for the Analysis of Data". Biometrika. Vol. 55 Nº 1, pp. 1-17. 1968.

[26] J.I. Marden. "Positions and QQ Plots". Statistical Science. Vol. 19 Nº 4, pp. 606-614. 2004.

[27] S.E. Maxwell and H.D. Delaney. "Designing Experiments and Analyzing Data: A Model Comparison Perspective". Psychology Press. 2nd Ed. New York, USA. 2004.

[28] H. Hulkko and P. Abrahamsson. "A multiple case study on the impact of pair programming on product quality". 27th international conference on Software engineering. ACM. pp. 495-504. 2005.

[29] L. Williams. "Integrating Pair Programming into a Software Development Process". 14th Conference on Software Engineering Education and Training. USA. 2001.

[30] J.E. Tomayko. "A Comparison of Pair Programming to Inspections for Software Defect Reduction". Journal of Computer Science Education. Vol. 12, pp. 213-222. 2002.

[31] R. Sison. "Investigating the effect of pair programming and software size on software quality and programmer productivity". 16th Asia-Pacific Software Engineering Conference. IEEE. pp. 187-193. 2009.

[32] D.T. Campbell and J.C. Stanley. "Experimental and Quasi-Experimental Designs for Research". Houghton Mifflin Company. 1963.

[33] M. Genero, J. Cruz-Lemus and M. Piattini. "Métodos de Investigación en Ingeniería de Software". RA-MA. 2014.

[34] O.S. Gómez, A.A. Aguileta, R.A. Aguilar, J.P. Ucán, R.H. Rosero and K. Cortes-Verdin. "An Empirical Study on the Impact of an IDE Tool Support in the Pair and Solo Programming". IEEE Access. Vol. 5, pp. 9175-9187. 2017.

[35] O.S. Gómez and A.A. Aguileta. "Influence on the use of an IDE as tool support in the pair programming: A controlled experiment". IEEE Latin America Transactions. Vol. 16 Nº 3, pp. 948-956. 2018.

[36] O.S. Gómez, M Solari, C.J. Pardo and A.C. Ledezma. "A Controlled Experiment on Productivity of Pair Programming Gender Combinations: Preliminary Results". XX Ibero-American Conference on Software Engineering (CIbSE'2017). Argentina, pp. 679-692. 2017.