Efectividad del Test-Driven Development: Un Experimento Replicado

Oscar Dieste
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid
Madrid, España
odieste@fi.upm.es

Efraín R. Fonseca C., Geovanny Raura, Priscila Rodríguez
Departamento de Ciencias de la Computación
Universidad de las Fuerzas Armadas ESPE
Sangolquí, Ecuador
{rfonseca, jgraura, pprodriguez2}@espe.edu.ec

***Resumen*—Los métodos ágiles y sus prácticas asociadas, e.g.: Test-Driven Developement (TDD), son ampliamente utilizadas en la industria y han sido repetidamente sometidas a estudios empíricos. *Antecedentes:* Se han realizado diversos experimentos en empresas y academia acerca de TDD. En general, los experimentos no muestran un efecto positivo de TDD en la calidad del código o la productividad de los programadores. *Objetivo:* Replicar el experimento UPM 2014 efectuado por N. Juristo y su equipo, para reproducir sus resultados y secundariamente, estudiar el efecto de la experiencia del desarrollador en la efectividad de TDD. *Método:* Replicación experimental manteniendo similares el training y los materiales del experimento original. La replicación fue llevada a cabo en la Universidad de las Fuerzas Armadas ESPE sede Latacunga (ESPEL). Los sujetos experimentales fueron 17 estudiantes del Master en Ingeniería de Software. *Resultados:* Los resultados de la replicación confirman los efectos observados en UPM 2014. La efectividad de TDD ha resultado menor que ITL, aunque las diferencias no son significativas. La productividad y calidad del código producido por los estudiantes ESPEL cuando utilizan ITL y TDD es comparable a la de los estudiantes UPM, aunque menor en valores absolutos. *Conclusiones:* TDD no produce beneficios en calidad o productividad, o al menos no de forma inmediata. Parece necesario que los sujetos experimentales reciban training intensivo para que los efectos de TDD sean evidentes.**

Palabras Clave—Test-Driven Development, TDD, Experimentación, Ingeniería de Software

I. Introducción

Desde su introducción en la década de los 90's, las metodologías ágiles han venido ganando adeptos y, hoy en día, se configuran como una de las aproximaciones más utilizadas para desarrollar software [1]. Las metodologías ágiles se basan en una serie de prácticas, tales como la programación por pares o el desarrollo dirigido por pruebas (*TDD*, por sus siglas en inglés) que prometen aumentar la calidad del producto software y la productividad de los programadores. Para comprobar dichas promesas, se han realizado múltiples estudios empíricos, los cuales ponen de manifiesto que estas prácticas pueden ser en ocasiones perjudiciales (por ejemplo: la programación por pares puede reducir la productividad) [2]. Asimismo, existe un buen número de aspectos que matizan o moderan la calidad y la productividad, tales como la complejidad del producto o la experiencia de los desarrolladores.

TDD también ha sido objeto de múltiples estudios empíricos. Una reciente revisión sistemática [3] sugiere que TDD no posee efecto alguno, ni positivo ni negativo, sobre la productividad. Esto contrasta fuertemente con las opiniones de los evangelistas [4]. Adicionalmente, y a diferencia de la práctica de programación por pares, pocos estudios empíricos han estudiado posibles variables moderadoras (por ejemplo: la experiencia de los programadores, antes citada).

Las limitaciones en el conocimiento científico acerca de TDD han propiciado que algunos investigadores lleven a cabo estudios experimentales en TDD. A este respecto, N. Juristo y su equipo de la Universidad Politécnica de Madrid (UPM) han iniciado una línea de investigación, en el marco de la cual vienen realizando experimentos en distintas universidades y empresas. Este artículo reporta la replicación de uno de los experimentos realizados en UPM en Marzo de 2014. Esta replicación se llevó a cabo en la Universidad de las Fuerzas Armadas ESPE Sede Latacunga (ESPEL) en Ecuador en Mayo 2014. De acuerdo a la tipología propuesta por Gómez et al. [5], esta replicación puede ser clasificada como literal, conjunta y externa. El propósito de llevar a cabo la replicación fue verificar los resultados del experimento original y secundariamente, estudiar el efecto de la experiencia del desarrollador en la efectividad de TDD.

Los resultados de la replicación están en línea con los efectos observados en el experimento original. El uso de TDD no ha logrado aumentar la productividad de los programadores ni la calidad del código. Por el contrario, tanto la productividad como la calidad han sufrido una acusada caída una vez que los sujetos experimentales han comenzado a aplicar TDD. Esto no se ha observado en el experimento original con estudiantes. Finalmente, la tarea experimental parece ser el elemento que más influye en la productividad y calidad resultantes, tanto en el experimento original como en la replicación.

El presente artículo está organizado de acuerdo a las guías de Carver [6], simplificadas para adaptarse al espacio disponible. En la sección II, se presentan los antecedentes y trabajos relacionados. La información acerca del experimento original se incluye en la Sección III. Las diferencias entre el experimento original y la replicación se indican en la Sección IV. La comparación de los resultados de la replicación versus los del experimento original se muestran en la Sección V. Finalmente, se presentan las conclusiones de la investigación.

II. ANTECEDENTES Y TRABAJOS RELACIONADOS

Test-Driven-Development (TDD) [4] es una técnica que cobra fuerza alrededor del manifiesto ágil publicado por un grupo de practicantes de software [7], como parte de la metodología de desarrollo Xtreme Programming (XP) [8] y propone que en lugar de realizar algún diseño o modelo de software, se debe enfrentar el desarrollo en base a la generación de pruebas unitarias antes de la generación efectiva del código.

El proceso de TDD consiste de los siguientes pasos: En primer lugar una característica o requerimiento de usuario es seleccionado. A partir de ello, se escribe una prueba que cumpla una tarea pequeña o un pedazo de la característica o requerimiento de usuario (por ejemplo, un método o parte de la funcionalidad incluida en un método); esto produce una prueba que falla. A continuación, se escribe el código de producción que implementa la funcionalidad a ser probada. Una vez implementada la funcionalidad se ejecutan de nuevo las pruebas, así como todas las pruebas anteriores o preexistentes. Si alguna prueba falla se corrige el código de producción y el conjunto de pruebas se vuelve a ejecutar. Si las pruebas pasan se re-factoriza tanto el código de producción como las pruebas para que posean la mayor calidad como sea posible.

Este proceso se contrasta con la técnica de Test Last Development (TLD), que es habitualmente utilizada en los proyectos de desarrollo tradicionales. En TLD, se escribe en primer lugar el código, luego se escriben las pruebas, se ejecutan las pruebas y si fallan se corrigen los errores y se vuelven a ejecutar las pruebas. La figura 1, muestra el proceso de TDD, frente a TLD.

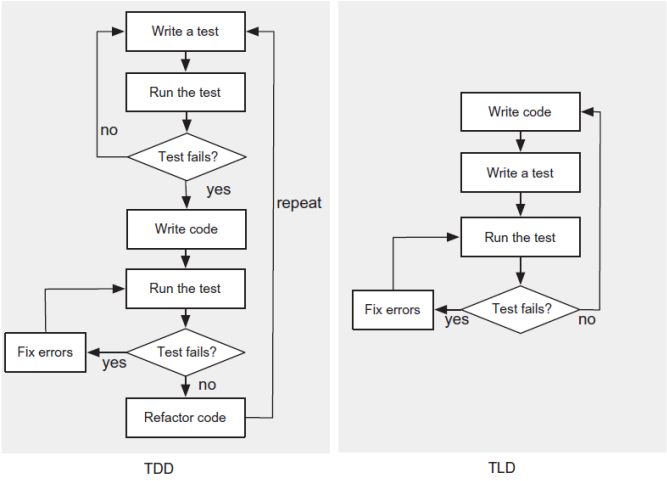

Fig. 1 Proceso de TDD frente a TLD (figura tomada de [9])

Entre las ventajas de TDD, de acuerdo a Beck [4], se consideran a la mejora en la comprensión del código, la eficiencia para encontrar el problema cuando se hallan nuevos defectos, la generación de bancos de pruebas que pueden incrementarse a medida que avanza el desarrollo y la posibilidad de reducir defectos introducidos en el código durante el mantenimiento y depuración.

Los estudios sobre la efectividad de TDD han sido sintetizados en diferentes revisiones sistemáticas de literatura [10], [11], [3], [12], [13] y [14]. Estas revisiones reportan que los estudios primarios se han efectuado tanto en la industria (la menor parte) como en el ámbito académico, utilizando para ello diferentes métodos de investigación (experimentos, casos de estudio, estudios piloto, o encuestas). Aunque los factores analizados son varios, es de nuestro interés el análisis de la calidad y de la productividad.

En cuanto a la calidad, la figura 2 muestra los efectos positivos, negativos o neutros de TDD, reportados en la revisión sistemática de Mäkinen & Münch [14] sobre 19 publicaciones.

Como se observa en la Figura 2, existen atributos de calidad que han sido menos o más estudiados (la mantenibilidad ha sido un aspecto poco estudiado, en tanto que el esfuerzo es uno de los aspectos más estudiados). Se reporta además que la mayoría de estudios presentan resultados inconclusos o no concluyentes en diferentes aspectos investigados.

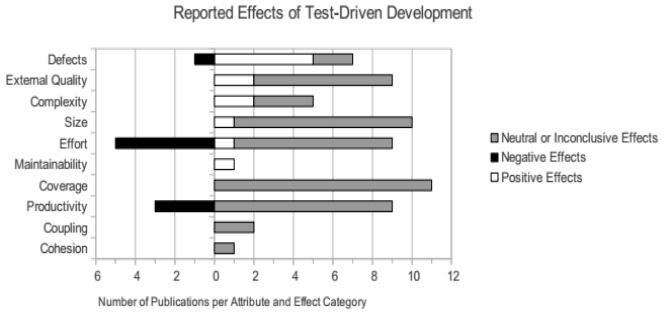

Fig. 2 Efectos de TDD en los atributos de calidad (Figura tomada de [14])

Estos resultados en general son similares a los hallazgos encontrados en [11], [10] y [3] donde además se puede notar que existe una tendencia a pensar que TDD mejora la calidad externa, sin embargo la calidad interna y la productividad presentan diferentes conclusiones.

Así podemos encontrar que, en la revisión sistemática de Kollanus [14], se analizan 40 artículos que incluyen 25 experimentos, 14 casos de estudio y una encuesta. Se reporta que hay una evidencia débil de que TDD mejora la calidad externa y que hay pruebas moderadas de que TDD disminuye la productividad. Los resultados son mucho más contradictorios si se toma en cuenta el tipo de investigación (por ejemplo, la mayoría de experimentos controlados no encuentran diferencia en la calidad externa). La evidencia sobre la calidad interna, conlleva también a resultados contradictorios.

En la revisión sistemática de Turhan et al. [11] se reunieron estudios de la industria y el mundo académico entre 2000 y 2008 y el panorama general es que los efectos del TDD son apenas demostrables. En 32 estudios primarios publicados en 22 artículos, se concluye que la calidad externa parece que se mejora específicamente cuando se analizan los datos obtenidos de estudios piloto y de estudios en la industria. En experimentos controlados los resultados no son concluyentes. Aunque en términos generales la evidencia sugiere que TDD puede mejorar la calidad externa. En cuanto a la productividad, al parecer mejora cuando se analizan datos de experimentos controlados. Sin embargo, los estudios piloto proporcionan evidencia mixta, algunos en favor y otros en contra de TDD. En los estudios en la industria, la evidencia indica que TDD disminuye la productividad, por lo que en forma general se puede indicar que los resultados no son concluyentes en este aspecto.

Shull et al. [10], resume los resultados de Turhan et al. [11] y concluye que existe moderada evidencia de que TDD mejora la calidad externa. En cuanto a la productividad los resultados difieren drásticamente en diferentes tipos de estudio. Los experimentos tienden a favorecer a TDD, en tanto que los estudios industriales tienden a favorecer el enfoque tradicional; finalmente los estudios de caso tienen resultados mixtos. Estos resultados además son contrastados con la opinión de expertos.

El meta-análisis de Rafique y Misic [3] con 27 estudios empíricos publicados hasta febrero de 2011, determina que en general, TDD tiene un pequeño efecto positivo en la calidad externa, pero poco o ningún efecto discernible sobre la productividad. En un análisis de subgrupos (estudios académicos y estudios industriales) se ha encontrado que la mejora de la calidad externa y caída de la productividad es mucho más grande en la industria, en comparación con los

estudios académicos; sin embargo no hay evidencia concreta que relacione la experiencia con la mejora de la calidad.

Munir [10] concluye que estudios de mayor rigor y relevancia muestran resultados de que TDD mejora en la calidad externa, pero parece que esto genera una pérdida de la productividad. Se encuentra además que estudios de menor rigor no muestran ninguna diferencia; y por lo tanto sostiene que estos hallazgos deben justificarse por un mayor número de experimentos en la industria y estudios de caso longitudinales.

La tabla 1 resume por orden cronológico los resultados de las revisiones sistemáticas citadas (los estudios que informaron sobre la calidad interna son omitidos). También se observa que Shull et al. [10] (*) resume la resultados de Turhan et al. [11], por lo que se muestra como un único estudio ya que sus conclusiones son similares.

TABLA 1. EFECTOS DE TDD EN LA CALIDAD EXTERNA Y PRODUCTIVIDAD DE ACUERDO A DIFERENTES ESTUDIOS.

Revisión	Año	Calidad Externa	Productividad
Kollanus [14]	2010	Experimento controlado: Sin diferencia Estudios de caso: Mejora Otros: Mejora **PROMEDIO: Mejora**	Experimento controlado: No concluyente Estudios de caso: reducción Otros: Mejora **PROMEDIO: Reducción**
Turhan et al. [11] (*) Shull et al [10]	2010	Experimento controlado: no concluyente Estudios piloto: Mejora Industria: Mejora **PROMEDIO: Mejora**	Experimento controlado: mejora Estudios piloto: no concluyente Industria: Disminución **PROMEDIO: No Concluyente**
Rafique & Misic [3]	2013	Experimento académico: sin diferencia Industria: Mejora Test Last: Mejora Iterative Test-Last: no concluyente (potencial disminución) **PROMEDIO: Mejora**	Experimento académico: mejora Industria: disminución Test Last: disminución Iterative Test-Last: no concluyente (potencial mejora) **PROMEDIO: No concluyente**
Munir et al. [9]	2014	Estudios de Alto Rigor y Alta Relevancia (A): Mejora Estudios de Bajo Rigor y alta Relevancia (B1): Mejora Estudios de Alto Rigor y baja Relevancia (B2): sin diferencia Estudios de Bajo Rigor y Baja Relevancia (C) : no concluyente **PROMEDIO: Mejora**	Estudios de Alto Rigor y Alta Relevancia (A): disminuye Estudios de Bajo Rigor y alta Relevancia (B1): disminuye Estudios de Alto Rigor y baja Relevancia (B2): sin diferencia Estudios de Bajo Rigor y Baja Relevancia (C) : no concluyente **PROMEDIO: No concluyente**

En síntesis podemos indicar que los estudios citados reportan diferentes resultados sobre los efectos de TDD en la calidad externa, la calidad interna y la productividad dependiendo de los grupos y subgrupos investigados, rigor y relevancia de los métodos de investigación utilizados, y otros factores que en general no han permitido llegar a conclusiones sólidas.

Creemos que lo que subyace en los resultados aparentemente contradictorios sobre la efectividad de TDD es una falta de consideración de las características personales de los sujetos como pueden ser: la experiencia y el conocimiento previo en TDD, su habilidad para realizar casos de pruebas, el conocimiento del dominio, la motivación, entre otros; sin que se evidencien estudios significativos sobre estos aspectos hasta la actualidad.

III. INFORMACIÓN ACERCA DEL EXPERIMENTO ORIGINAL

El experimento original fue realizado por N. Juristo y su equipo en el marco del proyecto ESEIL[1]. El objetivo de este experimento fue estudiar la efectividad de TDD en comparación con Iterative Test-Last (ITL). ITL es la aproximación más usada para el desarrollo de software apoyado por test automatizados. ITL consiste en el desarrollo de pequeñas porciones del código de producción, seguido inmediatamente por la realización de pruebas de unidad [15].

A. Factores y Variable Respuesta

El experimento original ensayó como factor principal la *aproximación de desarrollo*, con los niveles ITL y TDD. Se usó como factor secundario la *tarea* que los sujetos debían resolver. La *tarea* tuvo cuatro niveles, que correspondían con cuatro *katas*[2] ampliamente usados en experimentos acerca de TDD: *MarsRover* (MR), *MusicPhone* (MP), *BowlingScoreKeeper* (BSK) y *Sudoku* (SDKU). La web del proyecto ESEIL contiene la descripción de dichas tareas.

La efectividad de la *aproximación de desarrollo* (ITL y TDD) puede ser estudiada desde varias perspectivas. En el experimento original se han estudiado dos variables: la *calidad externa* (QLTY) y la *productividad* (PROD), junto con otras variables no relevantes para este trabajo. QLTY representa el grado de corrección del código desarrollado por los sujetos, y se define como:

$$QLTY = \frac{\sum_{i=1}^{\#tus} QLTY_i}{\#TUS} \quad (1)$$

donde $QLTY_i$ es la calidad de la historia de usuario i-ésima implementada por el sujeto. $QLTY_i$ se define como:

$$QLTY_i = \frac{\#Assert_i(Pass)}{\#Assert_i(All)} \quad (2)$$

mientras que #TUS es:

$$\#TUS = \sum_{i=1}^{\#us} \#Asser_i(Pass) \geq 0 \rightarrow True \quad (3)$$

En ambos casos, $\#Assert_i(Pass)$ representa el número de aserciones de jUnit (ya que el experimento se realizó utilizando el lenguaje de programación Java) correctamente ejecutados en la historia de usuario i-ésima. PROD representa la cantidad de trabajo realizada por los sujetos, y se define como:

$$PROD = \frac{\#Assert(Pass)}{\#Assert(All)} \quad (4)$$

B. Hipótesis

El experimento original posee dos hipótesis experimentales; la primera hace referencia a que la calidad del producto software no se ve alterada por el uso de ITL o TDD:

H_{10}: $QLTY_{ITL} = QLTY_{TDD}$
H_{11}: $QLTY_{ITL} <> QLTY_{TDD}$

Mientras que la segunda hipótesis afirma lo mismo respecto a la productividad:

H_{20}: $PROD_{ITL} = PROD_{TDD}$
H_{21}: $PROD_{ITL} <> PROD_{TDD}$

C. Diseño

Debido al previsible reducido número de sujetos experimentales, los investigadores originales decidieron utilizar un diseño de medidas repetidas para aumentar el poder estadístico [8,9]. Dicho diseño se muestra en la Tabla 2. Este diseño puede calificarse como ABBB, ya que el nivel de interés (TDD) se aplica repetidas veces para mejorar las habilidades de los sujetos y poder detectar más fácilmente sus efectos. El factor secundario *tarea* fue contrabalanceado en las

[1] *Empirical Software Engineering Industrial Lab*, por sus siglas en inglés. La información acerca de los experimentos realizados está disponible en https://sites.google.com/site/fidiproeseil/

2 Las katas corresponden a la especificación y código base de programas muy conocidos en el ámbito de TDD, usados como objetos experimentales

cuatro sesiones experimentales para evitar confundir los factores *tarea* y *aproximación de desarrollo*.

TABLA 2. DISEÑO EXPERIMENTAL

Secuencia Temporal			
Sesión 1	***Sesión 2***	***Sesión 3***	***Sesión 4***
ITL	TDD-1	TDD-2	TDD-3

D. Amenazas a la Validez

Los diseños de medidas repetidas poseen generalmente las siguientes amenazas a la validez [16]: fatiga, práctica, carry over y orden/periodo. En el presente experimento operó sin duda la amenaza de fatiga, ya que las sesiones son contiguas en el tiempo. Sin embargo, creemos que las restantes amenazas no aplican, por las siguientes razones:

- **Práctica:** TDD es una aproximación nueva para la mayoría de los sujetos experimentales. Para poder identificar los potenciales efectos beneficiosos de TDD, los sujetos deben lograr cierto grado de pericia. Por lo tanto, la práctica obtenida mediante la aplicación repetida del nivel TDD no representa una amenaza a la validez sino una condición necesaria para alcanzar los objetivos experimentales.
- **Carry over:** ITL utiliza estrategias parecidas a TDD, por lo que el carry-over, al igual que la práctica, resulta favorable para el experimento.
- **Orden/periodo:** Las sesiones experimentales son contiguas en el tiempo. Descontados los efectos de práctica y carry-over, no existe ninguna razón que sugiera la existencia de un efecto de orden/periodo.

E. Ejecución del Experimento Original

El experimento original se realizó en academia, utilizando como sujetos experimentales 16 estudiantes de maestría de la UPM. Antes de realizar el experimento se aplicó un cuestionario demográfico, cuyos resultados se muestran en la Tabla 3 de forma simplificada. Todos los sujetos poseen titulaciones relacionadas con la informática (Ingeniero de Sistemas y Computación, o similar), y una experiencia profesional media-baja (menor a 4 años, con pocas excepciones). En la actualidad, ninguno de los sujetos se encuentra trabajando en industria. Todos han usado lenguajes procedurales y orientados a objetos. A pesar de la poca experiencia, la inmensa mayoría de los sujetos han realizado labores de programación en la industria. Un 50% de los sujetos reportan poseer experiencia intermedia en programación. La mayoría de los sujetos conocen Java, aunque no pruebas de unidad ni jUnit. Tres sujetos reportan haber usado TDD como metodología de desarrollo por un breve lapso de tiempo.

El experimento original se realizó en la UPM en el mes de Marzo de 2014. La duración total del experimento fue de 5 días, realizándose sesiones experimentales a partir del segundo día. Para la realización del experimento se utilizó el lenguaje de programación Java y jUnit 4.

La descripción detallada de los resultados del experimento original está disponible en el sitio web del proyecto ESEIL[3] En lo que sigue reportaremos únicamente los resultados principales con el objetivo de permitir la comparación del experimento original y la replicación en la Sección III literal F.

El experimento original ha sido incapaz de obtener efectos significativos de la *aproximación de desarrollo* tanto para la variable respuesta calidad como productividad, si bien en esta última se aprecia una cierta tendencia a la significación estadística (*p-valor* = 0.116). Por el contrario, se ha podido constatar la influencia de la tarea tanto en calidad como en productividad (*p-valor* < 0.000 en ambos casos). La Figura 3 muestra el gráfico de perfil para la variable respuesta calidad, mientras que la Figura 4 muestra la misma información para la productividad. Se observa claramente que, para ambas variables respuesta, las *tareas* BSK y SDKU registran más altas cotas de calidad y productividad que las *tareas MP y SDKU.*

TABLA 3. DATOS DEMOGRÁFICOS.

ID	Experiencia			
	Profesional (4 años)	***¿Prog. Profesional?***	***Programación***	***Java***
3	4 años	Si	Novice	Novice
4	4 años	Si	Intermediate	Intermediate
5	1.15 años	Si	Intermediate	Novice
6	.	Si	Intermediate	Intermediate
7	2 años	Si	Novice	Novice
8	0.5 años	Si	Intermediate	Intermediate
9		No		
10	1.5 años	Si		
12	6 años	No		
13	0.15 años	Si	Novice	Novice
14	2 años	Si	Novice	Intermediate
15	4 años	Si	Intermediate	Intermediate
16	10 años	Si	Intermediate	Intermediate
17	6 años	Si	Intermediate	Novice
20	14 años	Si		
21	12 años	Si	Intermediate	Novice

NOTA: Las celdas con un *'.'* corresponden a datos no reportados por los sujetos. Todas las celdas en blanco corresponden con el valor *"Sin experiencia"*. La experiencia se valora de la siguiente forma: No experience=0-2 años; Novice=2-5 años; Intermediate=5-10 años; Expert>10 años.

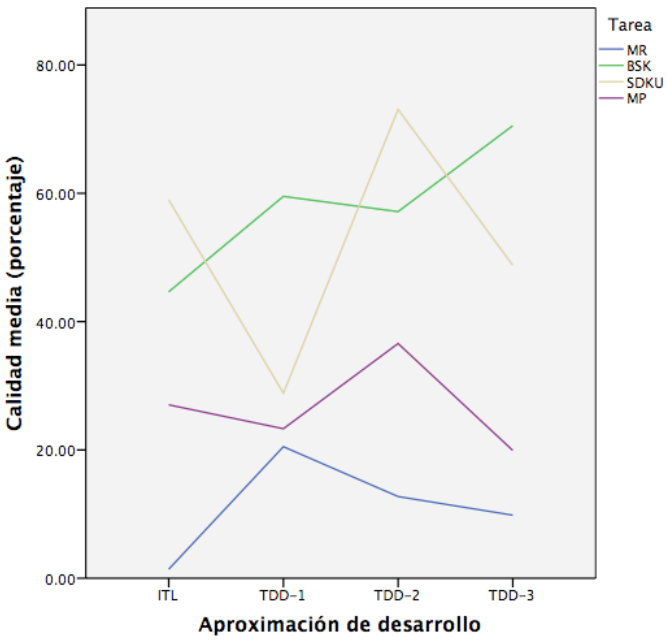

Fig. 3 Gráfico de perfil (variable respuesta calidad)

IV. INFORMACIÓN ACERCA DEL LA REPLICACIÓN

La replicación fue realizada en la Universidad de las Fuerzas Armadas ESPE de Ecuador - Sede Latacunga (ESPEL en lo que sigue), en el marco del curso de Verificación y Validación de Software de la Maestría en Ingeniería de Software. El experimento consistió en realizar ejercicios en la práctica, que fueron evaluados como parte de la asignatura.#

[3] https://sites.google.com/site/_diproeseil/

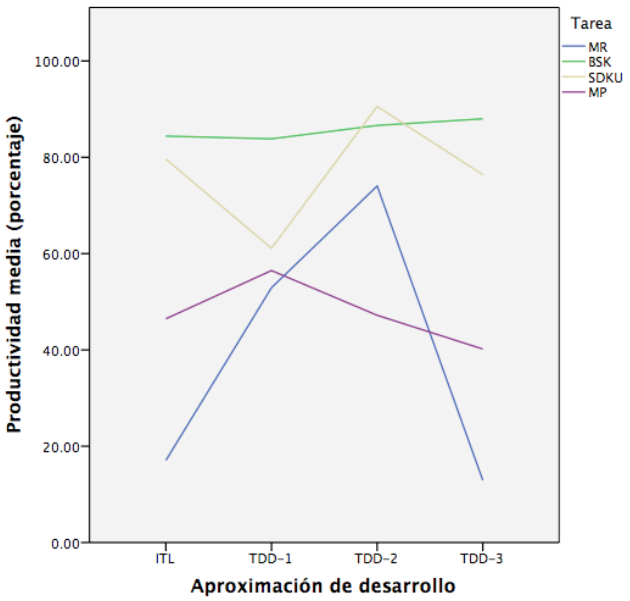

Fig. 4 Gráfico de perfil (variable respuesta productividad)

A. Motivación para la Realización de la Replicación

La razón principal que motivó la realización de la replicación fue conformar los resultados del experimento original o, en el caso de encontrar diferencias, identificar los factores o parámetros que podrán haber causado las desviaciones. Asimismo, esperamos que la realización de replicaciones nos permita identificar qué influencia ejerce el programador, en función de su experiencia (novato, profesional, etc.) en la efectividad de las estrategias ITL y TDD. No obstante, no esperamos que la presente replicación arroje resultados al respecto, siendo necesarios más experimentos para identificar efectos con relativa seguridad.

B. Nivel de Interacción con los Experimentadores Originales

La replicación fue guiada por uno de los experimentadores originales (O. Dieste) durante todo el ciclo experimental, tanto en la fase de entrenamiento como en la colección de datos y análisis. La replicación puede calificarse como literal (es decir, la replicación se asemeja al experimento original tanto como sea posible), conjunta (algunos de los experimentadores originales participaron en la replicación) y externa (la replicación se llevó a cabo en un sitio diferente) [17].

C. Cambios Respecto al Experimento Original

En general, la replicación se llevó a cabo de acuerdo al experimento original en todos los aspectos. Las hipótesis, factores, variables respuesta, materiales, procedimiento experimental no fueron cambiados. La población experimental fue muy similar (estudiantes de postgrado, de la UPM en el experimento original y de la ESPE en la replicación). La orientación general de la docencia también fue muy similar, aunque en el experimento original el profesor fue B. Turham (de la Universidad de Oulu, Finlandia) y en la replicación el profesor fue O. Dieste (UPM).

La única diferencia, y aun así no sustancial, reside en el diseño experimental. El experimento original tuvo una duración de 5 días, por lo que fue posible realizar 4 sesiones experimentales, lo que resultó en un diseño ABBB (ver Sección II literal C). La replicación tuvo una duración de 4 días, lo que exigió eliminar una de las sesiones TDD. El diseño de la replicación fue ABB. Las amenazas a la validez son las mismas del experimento original (ver Sección III literal D).

D. Ejecución de la Replicación

17 estudiantes tomaron parte en el experimento. Sus datos personales se obtuvieron, de forma previa al experimento, mediante una encuesta implementada en Google Forms. La información demográfica se muestra en la Tabla 3. Todos los sujetos poseen titulaciones relacionadas con la informática. La experiencia profesional parece asimismo considerable, aunque no todos los sujetos han reportado dicha información. En lo tocante a las habilidades en programación, todos los sujetos han usado lenguajes procedurales y orientados a objetos (por brevedad, el detalle se ha omitido de la Tabla 3). No obstante, un 50% de los sujetos se califican a sí mismos como sin experiencia o novatos en programación, lo cual va habitualmente unido al hecho de que no han trabajado como programadores en la industria. La mayoría de los sujetos desconoce Java, pruebas de unidad y jUnit. Ninguno de los sujetos recibió formación específica en TDD con anterioridad al experimento, aunque uno de ellos (#3) reporta haber usado TDD en entornos ágiles durante 1 año.

El experimento se realizó en paralelo con un curso en Verificación y Validación. La duración total del curso es de 32 horas, divididas en cuatro sesiones presenciales de 8 horas de jueves a domingo. Durante cada sesión se impartió docencia acerca de prueba de unidad, ITL y TDD. La docencia estuvo apoyada por ejercicios prácticos. Las sesiones experimentales tuvieron lugar de Viernes a Domingo, en horario de mañana. Esta agenda fue respetada sin que tuvieran lugar desviaciones relevantes.

Los sujetos #12 y #13 no entregaron sus soluciones a las tareas experimentales, probablemente a causa de su pobre desempeño, el cual pudo ser observado por los experimentadores durante la realización de las sesiones experimentales. El sujeto #22 tampoco entregó sus soluciones, aunque su desempeño fue aparentemente satisfactorio.

E. Resultados

En lo que respecta a la aproximación de desarrollo *ITL* supera a *TDD-1* y *TDD-2* tanto en calidad como en productividad. En lo tocante a las tareas, BSK alcanza las mayores cotas de calidad y productividad, seguido por SDKU, MP y, finalmente, MR. Las dispersiones son notables tanto para la aproximación de desarrollo como para la tarea. En muchos casos, la mediana de los niveles se sitúa en cero. Esto implica que varios sujetos han tenido dificultades para solucionar los problemas planteados. El diseño experimental exige la aplicación de un análisis de medidas repetidas.

Debido a la complejidad del diseño y a la existencia de covariables, el método estadístico más apropiado es un modelo mixto. Hemos especificado como factores fijos la aproximación de desarrollo y la tarea, así como su interacción. La aproximación de desarrollo se ha incluido también como factor aleatorio en el análisis, para modelar los efectos de aprendizaje. Esto es posible, ya que el orden de realización de las sesiones y la aplicación de los tratamientos están confundidos. Como covariables, se ha añadido la experiencia en programación y la experiencia en pruebas de unidad, que parecen ser los dos aspectos en los que los sujetos experimentales se diferencian en mayor medida (ver Tabla 3). La estructura de covarianzas escogida ha sido AR(1), la cual es a menudo adecuada para experimentos realizados en intervalos temporales consecutivos. Los resultados del análisis se muestran en la Fig. 5 y Fig. 6 para la calidad y productividad, respectivamente.

Los residuos de modelo son normales tanto para la productividad como para la calidad (test Shapiro-Wilks, p-valores 0.215 y 0.071 respectivamente). Los resultados del modelo mixto son, pues, confiables. El factor aproximación de desarrollo ha resultado no significativo, así como también la interacción aproximación de desarrollo por tarea, y las covariables experiencia en programación y la experiencia en pruebas de unidad. No pueden descartarse errores de tipo II debido al reducido número de sujetos experimentales.

En contrapartida, el factor tarea ha resultado significativo para las dos variables respuesta. En lo que respecta a la productividad, la comparación por pares indica que BSK >SDKU >MR (p-valores 0.0009 y 0.04, respectivamente). En lo tocante a la calidad, BSK >MP y BSK >MR (p-valores 0.011 y 0.004, respectivamente).

Type III Tests of Fixed Effects[a]

Source	Numerator df	Denominator df	F	Sig.
Intercept	1	12.753	5.253	.040
Tarea	3	17.603	4.316	.019
Aproximación D.	2	17.339	1.023	.380
Tarea * Aproximación D.	6	15.191	.882	.531
Exp. Prueba	1	9.418	.005	.943
Exp. Program.	1	9.677	.390	.547

a. Dependent Variable: Calidad (porcentaje).

Fig. 5 Modelo mixto (variable respuesta calidad)

Type III Tests of Fixed Effects[a]

Source	Numerator df	Denominator df	F	Sig.
Intercept	1	11.398	9.374	.010
Tarea	3	17.393	3.326	.044
Aproximación D.	2	16.979	1.636	.224
Tarea * Aproximación D.	6	14.462	1.398	.281
Exp. Prueba	1	7.855	.231	.644
Exp. Program.	1	8.111	.125	.733

a. Dependent Variable: Productividad (porcentaje).

Fig. 6 Modelo mixto (variable respuesta productividad)

La gran diferencia existente entre BSK y las restantes tareas experimentales es incluso más evidente en los gráficos de perfil que se muestran en las Figuras 7 y 8.

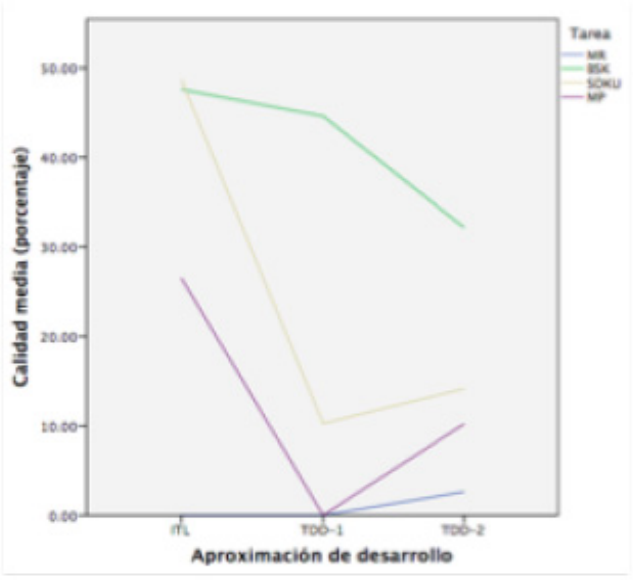

Fig. 7 Gráfico de perfil (variable respuesta calidad)

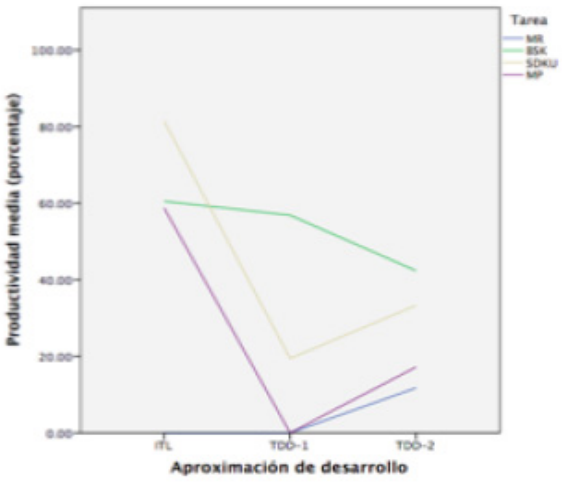

Fig. 8 Gráfico de perfil (variable respuesta productividad)

F. Comparación de Resultados

Considerando que la replicación es del tipo literal respecto al experimento original, los resultados son comparables en todos sus aspectos y coincidentes en su mayor parte.

Ni el experimento original ni la replicación fueron capaces de obtener efectos significativos de la *aproximación de desarrollo* tanto para la variable respuesta calidad como productividad. En lo que respecta a la *tarea*, los resultados son significativos en ambos casos, y con unas tendencias muy similares. BSK y SDKU obtienen mayores valores de productividad y calidad que MP y MR, aunque SDKU en ESPEL se destaca menos que en el experimento original.

La mayor diferencia entre ambos experimentos son los valores absolutos de las variables respuesta. Los datos de ESPEL son claramente más bajos que en UPM. Esta situación no es nueva, ya que en experimentos anteriores realizados en ESPEL [18] los resultados han mostrado una tendencia similar. La razón del menor rendimiento podría deberse a las características de la población (los sujetos UPM exhiben mayor experiencia de programación que ESPEL), así como a problemas de motivación y cansancio por el carácter intensivo del curso en ESPEL.

Finalmente, se observa una caída muy fuerte de productividad y calidad en ESPEL el primer día que los sujetos aplicaron TDD. (*TDD-1*) Esto no ocurre en UPM. La productividad y calidad aumentan en *TDD-2*. Ello podría indicar la necesidad de mayor entrenamiento.

V. CONCLUSIONES

TDD no produce beneficios en calidad o productividad, o al menos no de forma inmediata. Parece necesario que los sujetos experimentales reciban training intensivo para que los efectos de TDD sean evidentes. Se necesita una mayor cantidad de evidencias empíricas para establecer con seguridad los efectos de TDD.

REFERENCIAS

[1] B. Cartaxo, A. Araujo, A. Sa Barreto y S. Soere, «The impact of scrum on customer satisfaction: An empirical study,» Software Engineering (SBES), pp. 129-136., 2013.

[2] J. E. Hannay, T. Dyba, E. Arisholm y D. I. S, «The e□ectiveness of pair programming: A meta-analysis Information and Software Technology,» special Section: Software Engineering for Secure Systems Software Engineering for Secure Systems., vol. 51, nº 7, p. 1110 – 1122, 2009.

[3] Y. Rafique y V. Misic, «The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis,» Software Engineering, IEEE Transactions, vol. 39, nº 6, p. 835–856, 2013.

[4] K. Beck, Test Driven Development: By Example, Addison-Wesley Professional, 2002.

[5] J. C. Carver, «Towards reporting guidelines for experimental replications: A pro- posal,» Proceedings of the 1st International Workshop on Replication in Empir- ical Software Engineering Research (RESER), 4 May 2010.

[6] B. e. al., «agilemanifesto.org,» 2001. [En línea]. Available: http://agilemanifesto.org/. [Último acceso: 12 Agosto 2014].

[7] K. Beck y A. Cynthia, Extreme Programming Explained: Embrace Change., Addison-Wesley Professional., 2004.

[8] M. Hussan, M. Misagh y P. Kai, «Considering rigor and relevance when evaluating test driven development: A systematic review.,» Information and Software Technology, vol. 56, pp. 375-394, 2014.

[9] F. Shull, G. Melnik, B. Turhan, L. Layman, M. Diep y H. Erdogmus, «What do we know about test-driven development?,» IEEE Software, vol. 27, nº 6, pp. 16-19, 2010.

[10] B. Turhan , L. Layman , M. Diep , H. Erdogmus y F. Shull, «How Effective is Test-Driven Development?,» Making Software: What Really Works, and Why We Believe It, pp. 399-412, 2010.

[11] H. Munir y M. Moayyed , «Systematic Literature Review and Controlled Pilot Experimental Evaluation of Test Driven Development (TDD) vs.Test-Last Development (TLD),» 2012.

[12] A. Causevic, D. Sundmark y S. Punnekkat, «Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review,» Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference, pp. 337-346, 21-25 Marzo 2011.

[13] M. Simo y M. Jürgen , «Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies,» Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering, vol. 166, pp. 155-169, 2014.

[14] B. George y L. Williams, «An initial investigation of test driven development in industry.,» Proceedings of the 2003 ACM symposium on applied computing, pp. 1135-1139 , 2003.

[15] D. Janzen y H. Saiedian, «On the Influence of Test-Driven Development on Software Design,» Software Engineering Education and Training, pp. 141-148, 2006.

[16] T. Cook y D. Campbell, Quasi-Experimentation: Design & Analysis Issues for Field Settings, Boston: Houghton Mifflin, 1979.

[17] O. S. Gómez Gómez, «Tipología de Replicaciones para la Síntesis de Experimentos en Ingeniería del Software,» de Ph.D. dissertation, Universidad Politécnica de Madrid, 2012.

[18] R. E. Fonseca, E. G. Espinosa y O. Dieste, «Effectiveness for detecting faults within and outside the scope of testing techniques: An independent replication,» Empirical Software Engineering, vol. 18, nº 15, pp. 1-40, 2013.

Oscar Dieste. Es investigador científico de la Universidad Politécnica de Madrid. Anteriormente, fue becario Fulbright con la Universidad de Colorado en Colorado Springs y profesor asistente con las universidades Complutense de Madrid y de Alfonso X el Sabio. Sus intereses de investigación incluyen: la ingeniería de software empírica, la ingeniería de requisitos y sus intersecciones. Recibió su B.S. en Informática en la Universidad de La Coruña y su Ph.D. en la Universidad de Castilla-La Mancha

Efraín R. Fonseca C. Se recibió como doctor en julio de 2014 en la Universidad Politécnica de Madrid. Tiene diez años de experiencia como consultor en la industria de las TI. Es profesor investigador a tiempo completo en la Universidad de las Fuerzas Armadas ESPE de Ecuador. Entre sus temas de investigación de interés están: el proceso experimental en ingeniería de software, métodos de investigación en ingeniería de software, análisis y diseño orientado a objetos y representaciones ontológicas en ingeniería de software.

Geovanny Raura. Es Ingeniero de Sistemas e Informática con un grado de maestría en Ingeniería de Software obtenido en la Universidad Politécnica de Madrid. Es profesor investigador a tiempo completo en la Universidad de las Fuerzas Armadas - ESPE de Ecuador y Candidato a Phd. por la Universidad Nacional de La Plata de Argentina. Entre sus temas de investigación de interés están la ingeniería de software empírica, la ingeniería de requisitos, y técnicas de desarrollo de software basadas en metodologías ágiles.

Priscila Rodríguez. Es Ingeniera en Sistemas Informáticos de la Escuela Superior Politécnica de Chimborazo ESPOCH, cuenta con cinco años de experiencia en la industria de desarrollo de software bancario, obtuvo una Maestría en Gerencia de Proyectos de Ingeniería en la Universidad de Melbourne - Australia, actualmente se desempeña como Docente Tiempo Completo de la Universidad de las Fuerzas Armadas ESPE. Entre sus temas de investigación de interés están eGovernment, y el uso de metodologías para la Planificación Estratégica de Tecnologías de la Información.