Mejorando la Usabilidad de las Herramientas de Floss Refactoring haciendo uso del Análisis del Comportamiento del Desarrollador

Raúl Naupari Quiroz, Alvaro Cuno

Facultad de Ingeniería de Sistemas e Informática. Universidad Nacional Mayor de San Marcos Lima, Perú
raulnq@gmail.com
alvaroecp@gmail.com

Abstract.

[Contexto] El refactoring es el proceso por el cual se cambia la estructura interna del código fuente sin alterar el comportamiento externo del software. Es una práctica común y recomendada pues permite mejorar la mantenibilidad y extensibilidad del software.

[Objetivo] Si bien muchos entornos de desarrollo proveen herramientas para efectuarlo, éstas no son tomadas en cuenta por la mayoría de los desarrolladores principalmente porque, según los estudios, rompen con el flujo normal de sus actividades. El objetivo del presente trabajo es evidenciar que analizando el comportamiento del desarrollador es posible proponer un nuevo esquema de invocación-configuración que mejore la usabilidad de dichas herramientas.

[Método] Se plantearon 4 preguntas de investigación y se efectuaron 3 actividades para responderlas: recopilación, construcción y evaluación. En la recopilación se buscó responder la pregunta de investigación PI1 mediante la selección de los refactorings a ser tomados en cuenta en la investigación y la caracterización del comportamiento del desarrollador, obteniendo como resultado un esquema invocación-configuración. En base al esquema de invocación-configuración, en la segunda actividad, se realizó la implementación de la herramienta 'B-Refactoring', que permitió responder la pregunta de investigación PI2. Finalmente, a fin de responder las preguntas de investigación PI3 y PI4 se evaluó el impacto de los lineamientos encontrados sobre la usabilidad de la herramienta y su uso.

[Resultado] La herramienta que implementa el esquema invocación-configuración fue validada mediante una evaluación heurística encontrando que el 80% de los participantes calificaron a esta como superior a las alternativas actuales. Y empíricamente mediante un estudio controlado donde se evidencio que el desarrollador se sintió más a gusto con el nuevo esquema.

[Conclusión] El presente estudio confirma que considerar el comportamiento del desarrollador en la definición de los esquemas de invocación-configuración de las herramientas de refactoring incrementa su intención de uso.

Keywords: Palabras Clave: Floss refactoring, usabilidad, herramientas automatizadas.

1 INTRODUCCIÓN

M. Fowler [21] define al refactoring como el proceso de cambiar la estructura del código preservando su comportamiento externo. Emerson Murphy-Hill [6] tipifica el floss refactoring como el que ocurre cuando el desarrollador siempre sabe lo que necesita ser refactorizado, dado que este solo esta refactorizando lo que está entorpeciendo su progreso. M. Fowler [21] menciona que el

refactoring mejora el diseño y hace más fácil entender el software además de ayudar a buscar errores y programar más rápido. Y. Kataoka y colegas [26] muestran cómo puede disminuir las métricas de acoplamiento en el código existente. J. Benn y colegas [27] determinan que la complejidad, tamaño, cohesión y acoplamiento son mejorados. R. Kolb y colegas [28] muestran que la mantenibilidad y reusabilidad son incrementados mediante el refactoring. Ratzinger y colegas [29] ven que la mantenibilidad y capacidad de evolución fueron mejorados, y que el código producido fue mucho más claro y fácil de usar. R. Moser y colegas [30] concluyen que el software se puede hacer significativamente más reusable mediante el refactoring. Como respuesta a los beneficios potenciales que trae, actualmente la mayoría de entornos de desarrollo le dan soporte mediante la implementación de herramientas automatizadas. Sin embargo se ha encontrado que los desarrolladores no las utilizan en el grado que deberían. E. Murphy-Hill [6] entrevistó a 16 colegas de los cuales solo 2 reportaron usar las herramientas entre un 20% y 60% de sus actividades de refactoring. En el marco de la conferencia 'Agile Open Northwest 2007', se encuestó 112 personas [24], obteniendo como resultado que los desarrolladores escogen usar la herramienta sólo un 68% de las veces, haciendo el resto del trabajo a mano. Ya en el 2009, [14] analiza el uso de las herramientas de refactoring incluidas en Eclipse dando como resultado que el 90% de los refactorings son realizados manualmente. En cuanto los motivos por los cuales se produce esta falta de uso, M. Vakilian y colegas [12] indican que el factor clave para su uso es el método de invocación y en cuanto a los motivos de su desuso menciona a la excesiva configuración de las herramientas como uno de los principales. En el 2013, [15] afirma que usar los menús conduce a un gran costo de aprendizaje y de invocación. Y. Young Lee y colegas [11] identifican al menos tres problemas de usabilidad en las herramientas de refactoring actuales. Primero, el programador tiene problemas para identificar las oportunidades para usar la herramienta. Segundo, los programadores tienen dificultad para invocar el refactoring correcto y tercero, los programadores encuentran la configuración del refactoring complicada. D. Campbell y colegas [5] se enfocan en tres barreras, que a su parecer evitan que los desarrolladores adopten las herramientas de refactoring: falta de detectabilidad, falta de confianza en los resultados y la falsa percepción de que hacer el refactoring manual es más productivo. V Raychev y colegas [13] reafirman estas ideas, en primer lugar recalcan la pobre detectabilidad que tienen las herramientas con el fin de iniciar el refactoring y después habla sobre la complejidad de las interfaces de usuario, para ciertos refactorings, que son controlados por complejos diálogos de configuración. Stephen R. Foster [1] indica que el número de precondiciones cognitivas necesarias para concluir un refactoring de manera exitosa es numeroso y más aún si se desea usar una herramienta de refactoring automático, concluyendo que el éxito está en evitar o aliviar al programador del mayor número de estas precondiciones. Emerson Murphy-Hill y colegas en [7] [8] y [24] vuelven a mencionar la falsa percepción que tienen los desarrolladores en pensar que el refactoring manual es más productivo que el automático y esto debido en gran parte a la dificultad de invocación y configuración de estas herramientas.

Por lo mencionado, es posible concluir que las herramientas de refactoring sufren de un problema de usabilidad, lo que se traduce en su desuso. Por lo que se trata de responder la siguiente pregunta de investigación: ¿En qué medida se puede mejorar la usabilidad de las herramientas de floss refactoring con el fin de aumentar su uso? Para responder la se propone el análisis del comportamiento del desarrollador para generar un paradigma invocación-configuración idóneo para cada tipo de refactoring, a fin de que se mejore la usabilidad de las herramientas. Las contribuciones de este trabajo de investigación son:

- Enfoque: Se introduce una nueva pauta para el desarrollo de paradigmas de invocación-configuración de las herramientas de floss refactoring automático basado en cómo los usuarios tienden a trabajar de manera manual estos.
- Mapa invocación-configuración: Para que el enfoque se lleve a cabo se necesitó realizar un mapeo entre cada refactoring y la o las maneras en que se invocara y configurará.
- Herramienta: Con el fin de que de dar vida al enfoque se implementó una herramienta haciendo uso del mapa de invocación-configuración llamada "B-Refactoring".
- Evaluación: Se muestra la efectividad de la herramienta en el aspecto de la usabilidad a través de una encuesta y un estudio controlado.

2 TRABAJOS PREVIOS

Y. Young Lee y colegas [11] proponen una solución para llenar la brecha entre lo que se desea refactorizar y cómo completarlo, mediante el uso del drag and drop. Permitiendo al desarrollador manipular los elementos a refactorizar directamente, eliminando la necesidad de usar menús y diálogos de configuración. Bajo este esquema el desarrollador solo tendría que identificar el elemento del código que servirá de fuente (drag source) y el destino (drag target). Usando este enfoque se obtienen dos ventajas: se elimina la necesidad de navegar entre complicados menús y se elimina la existencia de un paso extra de configuración que sobrecargue de trabajo al desarrollador. La limitación de esta propuesta recae en la dificultad de traducir algunos refactorings al paradigma drag and drop debido, principalmente, a la naturaleza y forma de estos. V. Raychev y colegas [13] proponen un nuevo enfoque llamado refactoring con síntesis para la aplicación de refactorings complejos, lo cuales generalmente están compuestos de otros más simples. Mediante tres simples pasos: El desarrollador primero indica el inicio de una fase de refactorización de código, luego realiza los cambios en el código manualmente (los cambios finales) y finalmente pide a la herramienta que complete con los refactorings necesarios para alcanzar los cambios manuales. La propuesta cae en el error de adicionar acciones que no son parte de flujo habitual de trabajo del desarrollador además de tener varios problemas en su prototipo inicial. Stephen R. Foster y colegas [1] proponen WitchDoctor un sistema que puede detectar sobre la marcha, cuando un desarrollador está realizando un refactoring de manera manual. El sistema puede completar el refactoring en segundo plano y proponérselo al usuario antes que este termine de completarlo. Un trabajo similar es el presentado por Xi Ge y colegas [10] quien proponen a BeneFactor como una herramienta que detecta el refactoring manual que pueda estar realizando el desarrollador , le indica que existe un refactoring automático disponible y puede completar su refactorización automáticamente. Ambos trabajos tuvieron problemas al momento de implementar sus enfoques, dado que tratan de cubrir todos los escenarios posibles al momento de identificar cuando se está realizando un refactoring. E. Murphy-Hill y colegas en [3][7][8][9] proponen el uso de un conjunto de herramientas que en si atacan diferentes ángulos del problema de usabilidad aunque no dan una solución que pueda ser generalizada, pero sí demuestran que pequeños cambios aumentan radicalmente la aceptación de las herramientas de refactoring.

3 METODO

Para entender como el análisis del comportamiento del desarrollador puede mejor la usabilidad de las herramientas de floss refactorings se desdobló la pregunta de investigación en las siguientes:

- PI1) ¿Qué método de invocación y configuración es el más adecuado para cada técnica de refactoring?
- PI2) ¿Cómo podemos implementar los requisitos de usabilidad?
- PI3) ¿El análisis del comportamiento del desarrollador puede mejorar la usabilidad de las herramientas de refactoring?
- PI4) ¿El aumento de la usabilidad de la herramienta de refactoring puede aumentar su uso?

A fin de responder las preguntas de investigación se dividió la investigación en tres etapas:

1. Recopilación: En esta primera etapa se tuvo como objetivo responder a la PI1 mediante selección de los refactorings a ser tomados en cuenta en la investigación y la caracterización del comportamiento del desarrollador al momento de realizar cada uno de estos, obteniendo como resultado de esta fase el Mapa invocación-configuración.
2. Implementación: Basados en el Mapa invocación-configuración, en esta etapa se realizó la implementación de la herramienta 'B-Refactoring', que nos permite responder a la PI2.
3. Evaluación: A fin de responder las PI3 y PI4 se evaluó el impacto de los lineamientos encontrados sobre la usabilidad de la herramientas y si este puede aumentar el uso de las herramientas de refactoring.

4 RECOPILACIÓN

Basados en los estudios previos de S. Negara y colegas [16] y E. Murphy-Hill [14], sobre las técnicas de refactoring que poseen mayor uso, tomamos como parte de esta investigación:

- Convert Local Variable to Field: Su objetivo es convertir una variable local a una de la clase.
- Encapsulate Field: Tiene como objetivo dar a una variable privada accesos públicos.
- Extract Local Variable: Su objetivo es crear una variable local usando un valor provisto por un fragmento del código.
- Inline Local Variable: Tiene como objetivo usar el valor de una variable en lugar de esta.
- Extract Method: Tiene como objetivo agrupar fragmentos de código en un método aparte.
- Inline Method: Su objetivo es sustituir la invocación de un método por la implementación.
- Introduce Parameter: Tiene como objetivo adicionar nuevos parámetros a un método.
- Move Class to File: Tiene como objetivo mover una clase a un nuevo archivo.
- Rename: Tiene como objetivo cambiar el nombre de algún elemento del código.

Para entender cómo el desarrollador se comporta ante una tarea de refactoring se usaron tres técnicas para la recolección de datos, una proveniente de la ingeniería de requisitos y dos propias del campo del refactoring. Estas dos últimas son mencionadas en trabajos previos como medios de recolección de datos para este campo, y fueron puestas en práctica en los trabajos de M. Vakilian [4] o en el de E. Murphy-Hill [33].

4.1 Encuesta

Participantes y Método.

La recopilación de datos fue de tipo exploratorio sobre una muestra no probabilística y un enfoque fundamentalmente cualitativo; dado que el objetivo es documentar las experiencias y pers-

pectivas de los participantes en relación a los tipos de refactoring [25]. La encuesta contó con un total de 32 participantes con diferentes grados de experiencia en la programación orientada a objetos así como en su grado de familiaridad con los conceptos de refactoring. Constó de una pregunta abierta por cada método de refactoring seleccionado: ¿Cómo le gustaría que se ejecute el refactoring 'Refactoring X' dentro de su entorno de desarrollo? que se realizó mediante un formulario web, enviado por medio de correo electrónico con una vigencia de dos semanas.

Resultados y Análisis.

De las 72 invitaciones enviadas se recolectó un total de 32 encuestas de las cuales no todas tuvieron respuestas válidas sobre la pregunta en cuestión o simplemente no se respondieron. A fin de interpretar las respuestas, éstas se generalizaron y agruparon con el objetivo de ver el grado de ocurrencia de cada una. Los resultados indican que para cada uno de los refactorings elegidos los métodos de invocación más populares son: "Posicionarse sobre/Clic derecho" y "Opción y Posicionarse sobre/Shortcut". En cuanto al método de configuración la respuesta con mayor frecuencia fue "Automática". De los datos obtenidos se observa que los métodos de invocación y configuración predominantes refuerzan los estudios previos [6][11][12][24] donde se mencionan las características que debería tener una herramienta de refactoring usable, con lo cual podemos definir los siguientes requerimientos como esenciales en la implementación de la herramienta:

- El método de invocación tiene que activar el refactoring de la manera rápida y simple.
- El cambio entre la herramienta y la edición de programa debe ser lo más fluido posible.
- La configuración explícita no debe ser requerida bajo ningún caso.

4.2 Observación en el laboratorio y Monitoreo de la codificación

Participantes y Método.

La observación en el laboratorio contó de 5 participantes y el monitoreo de la codificación contó con un total 10. Cada uno con diferentes niveles de experiencia y grado de familiaridad con los conceptos de refactoring.

La observación en el laboratorio consistió de la aplicación de cada uno de los refactorings seleccionados sobre tres proyectos open sources desarrollados en el lenguaje C# que fueron seleccionados debido a su tamaño y grado de madurez. A cada participante se le asignó un proyecto, a fin que aplique cada uno de los refactorings sin el uso de ninguna herramienta automatizada. Todo este proceso fue grabado, al finalizar se evaluó el resultado de cada refactoring como "correcto", "incorrecto" u "otro" según se haya alcanzado el objetivo de manera satisfactoria, se haya alterado el comportamiento del código o no se haya terminado de realizar este. El monitoreo de la codificación se realizó mediante un plugin (para Visual Studio .Net) llamado 'VSCodignTracker' (desarrollado para esta investigación) el cual almaceno durante dos semanas información acerca de su comportamiento a la hora de programar.

Resultados y Análisis.

De todos los participantes de la observación en el laboratorio se obtuvo un video y el código fuente de los proyectos que cada uno modifico. El tiempo promedio de los video fue de 15 minutos aunque pero no todos lograron el 100% de respuestas válidas. En cuanto al monitoreo de la codificación todos los participantes brindaron la información que en sus computadoras estuvo almace-

nado el plugin previamente instalado. En este caso no se presentaron problemas y se pudo recolectar el 100% de los datos esperados.

Al igual que las respuestas de la encuesta, las actividades realizadas por los desarrolladores al momento de efectuar un refactoring fueron generalizadas, tipificadas y agrupadas con lo que se pudo documentar la secuencia o secuencias de pasos que realizaron para lograr cada uno de esos. En base a estos resultados se observa que los desarrolladores tienden a usar una secuencia de pasos común para la ejecución de un tipo específico de refactoring. Haciendo uso de esta información se propuso que el método de invocación fuera justamente un paso dentro del comportamiento. La Tabla 1 muestra el Mapa invocación-configuración, el cual nos brinda la siguiente información:

- La fuente es el elemento al que se le aplica la primera acción del método de invocación
- El método de invocación, dependiendo del refactoring consta de uno o dos acciones, la primera siempre está relacionada con la fuente y la segunda con el objetivo.
- El objetivo es una condición que debe cumplir la segunda acción del método de invocación.
- Configuración, implícita derivada del contexto que se usará para el refactoring.

Refactoring	Fuente	Invocación	Objetivo	Configuración
Convert Local Variable to Field	Declaración de variable local	Cortar-Pegar	Sobre área válida para la declaración de variables de clase.	Nombre de la variable: El mismo de la variable local.
Extract Local Variable	Valor de la variable	Cortar-Pegar	Sobre área válida para la declaración de variables locales	Nombre de la variable: Derivado del valor de la variable.
Inline Local Variable	Valor de la variable	Cortar-Pegar	Sobre el uso de la variable a reemplazar	-
Extract Method	Cuerpo de método	Cortar-Pegar	Sobre área válida para la declaración de métodos de clase	Nombre el método: Derivado de lo retornado por el método.
Inline Method	Cuerpo de método	Cortar-Pegar	Sobre el uso del método a reemplazar	-
Introduce Parameter	Declaración de la variable	Cortar-Pegar	Sobre área válida para la declaración de parámetros del método.	Nombre de la variable: Derivado del valor de la variable.
Rename	Declaración de variable, clase, método	Modificar	-	-

Tabla 1. Mapa de Invocación-Configuración

Existen dos refactoring (Encapsulate Field y Move Class to File) que no poseen un método de invocación definido en el presente trabajo dado que por su naturaleza se considera que los métodos convencionales ya sean mediante menú contextual o shortcuts los ejecutan de la mejor manera.

5 IMPLEMENTACIÓN

Dada la problemática previamente identificada se plantea B-Refactoring una herramienta que está orientada a simplificar los mecanismos de invocación y configuración, llegando a implementar un total de 7 refactorings. Nuestra herramienta automáticamente detecta cuando un refactoring manual se está llevando a cabo y sugiere al desarrollador que uno automático está disponible con

el cual puede terminar esta acción. B-Refactoring tiene 2 componentes principales, la detección del refactoring y la ejecución del mismo.

5.1 Detección del Refactoring

Con el fin de asistir al desarrollador en reconocer que está realizando un refactoring de manera manual B-Refactoring, a diferencia de otras herramientas, detecta lo que se está intentando hacer mediante la presencia de la terna Fuente-Invocación-Objetivo dentro de las acciones del desarrollador. Este constante proceso corre en segundo plano y principalmente logra su objetivo en dos etapas. En la primera trata de encontrar la ocurrencia de la dupla Fuente-Invocación mediante:

- Detección de cambios sobre el código fuente mediante la comparación del código antes y después de efectuar el cambio.
- Detección de la acción que está causando el último cambio al código fuente como copiar, pegar o cortar código.
- Detección la fuente a la cual se le aplicó la acción.

Si se encuentra una ocurrencia válida se almacena y se procede a la detección de una nueva dupla Invocación-Objetivo que consta de los mismos pasos, más uno nuevo que verifica que la nueva dupla encontrada es efectivamente la que complementa la previamente identificada. Ésta debe ocurrir inmediatamente después de la primera. De ocurrir lo anterior el componente concluye que el desarrollador está queriendo hacer un refactoring manual del tipo R y mostrará un smart tag en el editor del desarrollador para que éste elija completar el refactoring de manera automática.

5.2 Ejecución del refactoring

Si el desarrollador decide usar B-Refactoring, este tendrá que completar las siguientes tareas a fin de concretar el refactoring.

- Recolección de la información necesaria para llevar a cabo el refactoring. Esta información vendrá principalmente del contexto antes y después de efectuado el cambio en el código fuente.
- Aplicar el refactoring automático, completando el código manual hecho hasta el momento.

6 EVALUACIÓN

La siguiente sección tiene como finalidad probar que usando el análisis del comportamiento del programador podemos mejorar la usabilidad de las herramientas de refactoring. Para lo cual sean se llevaron a cabo 2 experimentos, el primero consta de una evaluación heurística realizada por un grupo de expertos. El segundo realiza una comparación cuantitativa entre la herramienta de refactoring automático Resharper y la nuestra.

6.1 Experimento 1

Participantes y Método.

Se contó con un total de 15 evaluadores todos ellos con más de 4 años experiencia programando en el lenguaje CSharp y actualmente trabajando activamente en tareas de programación. Diez

de los participantes indicaron que poseían los conceptos básicos de refactoring de código fuente, no obstante, igual se les dio a todo el grupo un tutorial indicando en qué consisten cada uno de los tipos de refactoring que iban a evaluar.

Dentro de los sistemas de evaluación de la usabilidad llamados "de inspección", se encuentra la evaluación heurística, que consiste en que un determinado número de evaluadores revisen la interfaz siguiendo unos principios de usabilidad [31]. Para acometer esta parte del análisis se tomó como referencia los principios heurísticos de Jakob Nielsen [32], teniendo en cuenta el tipo de herramienta que estamos evaluando, se escogieron los siguientes aspectos a evaluar:

- Flexibilidad y eficiencia de uso: La herramienta debe ser fácil de usar para usuarios novatos y para usuarios avanzados.
- Diseño estético y minimalista: Cualquier tipo de información que no sea relevante para el usuario y que sobrecargue la interfaz debe ser eliminada
- Libertad y control por parte del usuario: El usuario debe tener el control del sistema, no se puede limitar su actuación. No se debe forzar al usuario a seguir un camino determinado.
- Visibilidad del estado del sistema: La herramienta siempre debe informar al usuario acerca de lo que está sucediendo.
- Es mejor reconocer que recordar: El usuario debe tener siempre toda la información a mano, y no verse obligado a usar su memoria para seguir el hilo de la interacción.
- Coherencia. Todas las acciones tienen que ser siempre consistentes con el efecto que producen.

Los principios: Relación entre el sistema y el mundo real, Prevención de errores, Ayudar a los usuarios a reconocer, Ayuda y documentación; no fueron tomados en cuenta por considerarse no aplicables para esta herramienta y/o por ser cubiertos de manera intrínseca por el diseño de la misma. A fin de poner en práctica la metodología, se empleó una lista de verificación con cada uno de los principios mencionados, para cada uno de los refactoring que implementa nuestra herramienta. La cual fue llenada tanto para nuestra herramienta, como para la herramienta Resharper. Para esto se tuvo que entregar a los evaluadores un manual de B-Refactoring y otro de Resharper a manera de vídeo tutorial a fin de hacerlos conscientes del funcionamiento de ambas herramientas. Se definió una escala de 1(no cumple) a 5(cumple totalmente) para medir cada una de las heurísticas seleccionadas.

Resultados.

Después de la evaluación heurística realizada por los expertos utilizando la lista de verificación, se obtuvieron los resultados que se presentan en la Tabla 2.

	Convert Local Var. to Field		**Extract Local Var.**		**Inline Local Var.**		**Extract Method**		**Inline Method**		**Introduce Par.**		**Rename**	
Refactoring	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**
Flexibilidad	2,07	4,07	2,07	4,07	2,07	4,07	1,00	4,07	2,00	4,07	3,00	3,27	4,73	3,80
Diseño estético	2,20	4,27	2,13	4,20	2,20	4,27	2,07	4,20	2,20	4,20	2,13	4,07	4,07	3,87
Libertad y control	1,13	4,47	1,87	4,47	1,87	4,47	1,07	4,47	1,87	4,47	1,93	4,33	4,07	4,07

Visibilidad	3,80	2,47	2,93	2,47	3,07	2,47	2,93	2,47	2,93	2,47	2,93	2,40	2,73	2,40
Es mejor reconocer	1,93	4,80	1,93	4,80	1,93	4,80	1,93	4,80	1,93	4,93	1,93	4,80	4,73	4,80
Coherencia	2,13	3,93	2,13	4,07	2,13	3,93	2,13	3,93	2,13	3,93	2,13	3,93	3,00	3,93
Promedio	2,21	4,00	2,18	4,01	2,21	4,00	1,86	3,99	2,18	4,01	2,34	3,80	3,89	3,81

Table 2. Resultados de la evaluacion Heuristica. (R) Resharper, (B) B-Refactoring

6.2 Experimiento 2

Participantes y Método.

Se condujo una observación en laboratorio con 5 participantes a fin de evaluar la usabilidad de B-Refactoring. Todos los participantes son ingenieros de sistemas graduados que tienen a lo menos 4 años de experiencia en programación bajo el lenguaje csharp, la participación fue estrictamente voluntaria y las invitaciones fueron hechas mediante email.

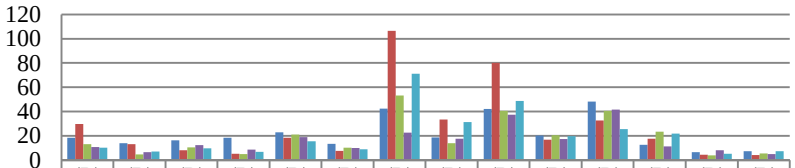
Se le pidió a cada participante que lleve a cabo dos veces cada uno de los 7 refactorings que implementa nuestra herramienta, una usando la herramienta Resharper y otro usando B-Refactoring. A fin de minimizar agentes externos que pudieran entorpecer la prueba esta fue llevada a cabo en la propia computadora de los participantes. Cada sesión fue grabada enteramente a fin de analizarla posteriormente. Las tareas de refactoring fueron extraídas de los 3 proyectos open source, 7 de cada uno. A cada participante se le entregó un manual de B-Refactoring y otro de Resharper. El tutorial consto con un pequeño video tutorial demostrando el uso de las herramientas, con un código de ejemplo totalmente diferente al que el participante evaluó. Se planeó recolectar los siguientes datos:

- Tiempo en realizar el refactoring: Para la herramienta Resharper consideraremos el momento de inicio cuando se active el refactoring y el momento de fin fue marcado con la actualización del código fuente con dicho refactoring. Para B-Refactoring se consideró como tiempo de inicio el momento en que se ejecuta el par Fuente-Invocación y el momento de fin fue marcado con la actualización del código fuente con dicho refactoring.
- Número de obstáculos detectados

Estas mediciones fueron hechas de manera manual después de terminada la observación haciendo uso de los videos grabados, con el fin de no afectar el rendimiento del participante al realizar las tareas.

Resultado.

La Figura 1 muestra el tiempo (en segundos) que le tomó a cada participante en realizar cada uno de los refactorings tanto para. B-Refactoring como para Resharper.



	(R)	(B)	(R)	(B)	(R)	(B)	(R)	(B)	(R)	(B)	(R)	(B)	(R)	(B)
	Convert Local Var. to Field		Extract Local Var.		Inline Local Var.		Extract Method		Inline Method		Introduce Par.		Rename	
Participante 1	18.5	13.8	16.2	18.5	22.9	13.4	42.3	18.6	42.1	20.2	48.3	12.7	6.4	7.4
Participante 2	29.8	13.2	8.2	5.2	18.3	7.5	106.4	33.5	80.1	16.8	32.5	17.6	4.4	4.1
Participante 3	13.1	4.7	10.5	4.9	21	10.1	53.3	13.8	40.6	20.4	40.5	23.5	4	5.6
Participante 4	10.7	6.4	12.3	8.7	18.9	9.9	22.6	17.6	37.4	17.3	41.5	11.4	8.2	4.9
Participante 5	10.2	7.1	9.7	6.9	15.4	8.8	71	31.3	48.6	19.6	25.5	21.8	5.2	7.3

Fig 1. Tiempo en segundos - que demora el desarrollador al efectuar los refactorings. (R) Resharper, (B) B-Refactoring

Tras evaluar las grabaciones de video con el fin de identificar los obstáculos que los participantes encontraron mientras realizaban las tareas de refactoring, obtuvimos como resultado una lista que los agrupa en categorías similares. La tabla 3 muestra las categorías y las ocurrencias de cada una de estas tanto para B-Refactoring como para Resharper. A continuación describimos las categorías que identificamos:

- Cancelaciones, cuando se cancela un refactoring durante la configuración o se deshace.
- Cambios manuales, cuando el programador decide hacer un cambio manual al código.
- Selección errónea, cuando el código al cual se aplica el refactoring fue mal seleccionado.
- No se puede escoger el refactoring, cuando se encontró dificultades para invocar el refactoring.
- Mala configuración, cuando el participante configuro mal la herramienta de refactoring.

	Partic #1		**Partic #2**		**Partic #3**		**Partic #4**		**Partic #5**	
Participante	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**	**(R)**	**(B)**
Cancelaciones	1	0	2	0	2	0	0	0	2	0
Cambios manuales	3	1	1	1	2	0	1	0	0	0
Selección errónea	0	0	1	0	0	2	0	1	3	1
No se puede escoger el refactoring	0	0	0	0	0	0	4	0	1	0
Mala configuración	0	0	0	0	1	0	2	0	0	0
Total	4	1	4	1	5	2	7	1	6	1

Table 3. Obstaculos que presenta el desarrollador al efectuar los refactorings. (R) Resharper, (B) B-Refactoring

Discusión.

Los resultados sugieren que B-Refactoring posee mejores características de usabilidad si se le compara con Resharper. Resharper es una herramienta de refactoring para Visual Studio .Net líder del mercado que usa los métodos tradicionales de invocación, como son los menú contextuales y/o shortcuts, y de configuración que agregan pasos extras al momento de realizar las tareas de refactoring. En la heurística inicial los evaluadores puntuaron a B-Refactoring sobre Resharper en todos los refactorings, a excepción del 'Rename'. Cuando hablamos cualitativamente y comparamos los tiempos que se gastan al usar una y otra herramienta, B-Refactoring reduce a la mitad el tiempo total empleado por los participantes en realizar todos las tareas de refactoring evaluadas, con lo que inferimos que el uso de nuestra herramienta aumenta la eficiencia del desarrollador y si consideramos a la eficiencia como un forma para medir la usabilidad [33] podemos decir también que es más usable.

Cabe mencionar que Resharper fue más eficiente sólo para el tipo de refactoring 'Rename' (por estrecho margen) al igual que en la heurística. Lo que nos llevó a prestar atención a dicho refactoring observando que este posee entre sus métodos de invocación uno exactamente igual al de nuestra herramienta, además de los comunes (menú contextual y shortcuts). Con lo que podríamos concluir que estamos comparando herramientas de similares características cuando hablamos de este tipo de refactoring.

Finalmente los participantes encontraron mayor número de obstáculos al momento de usar Resharper. La presencia de un mayor número de obstáculos impacta negativamente en la usabilidad de la herramienta, con lo también se reafirma que nuestra herramienta es superior en ese aspecto. Con lo que podríamos decir que si alineamos los métodos de invocación-configuración con el flujo normal de trabajo de los desarrolladores, gracias al análisis del comportamiento podemos aumentar la usabilidad de las herramientas de floss refactoring.

7 LIMITACIONES

7.1 Amenazas a la Validez

Validez Interna.

Permitimos a los participantes usar sus propias maquinas por un tema de familiaridad. Estas máquinas varían ampliamente tanto en hardware, software y en configuración. Estas diferencias podrían tener un efecto en la duración de las tareas de refactoring. La selección de los proyectos, clases y tareas de refactoring involucrados en los experimentos fueron escogidos de manera subjetiva y nos son representativos de las tareas que normalmente haría un desarrollador. Tenemos planeado realizar un estudio considerando los refactorings hechos por parte de los desarrolladores sobre los mismos proyectos pero respondiendo a requerimientos a fin de emular un entorno más real. Finalmente, dado que los participantes eran conscientes que la herramienta era prototipo, se pudieron haber parcializado en contra o a favor el enfoque según sus motivaciones personales.

Validez externa.

Nuestros participantes en la encuesta inicial, evaluación heurística y observación en laboratorio fueron profesionales de la ingeniería de sistemas y carreras afines. Aunque los participantes tienes diferentes grados de experiencia con Csharp y Visual Studio, ellos no podrían ser una muestra representativa de todos los desarrolladores que usan herramientas de refactoring. Sin embargo dentro de un grupo grande y heterogéneo, se detectaron aspectos y comportamientos comunes lo

que nos sugeriría que estamos dirigiéndonos por buen camino. El estudio tomo inicialmente nueve técnicas de refactoring como base para las investigaciones por lo que los resultados solo son válidos para estas. Si se quisiera generalizar estos para otras técnicas de refactoring se tendría que repetir los pasos descritos en el presente estudio. Finalmente, 'B-Refactoring' fue solo implementado para Visual Studio y comparado con la herramienta Resharper dejando de lado otras herramientas de refactoring que poseen un grado de uso semejante por parte de los desarrolladores. Pero debido a que la gran parte de estas herramientas posee el mismo modelo de interacción con los desarrolladores, esperamos que nuestros resultados se mantengan si fueran comparadas.

Confiabilidad.

Todos los materiales del experimento y datos recolectados están disponibles en: https://sites.google.com/site/refactoringtraining/b-refactoring.

7.2 Limitaciones de B-Refactoring

La actual implementación de B-Refactoring solamente implementa 7 tipos de refactoring diferentes y deja abierto someter a análisis los demás tipos de refactoring a fin de poder incorporarlos a la herramienta o construir una nueva que siga los mismos principios de diseño. Durante la realización del trabajo encontramos que el enfoque planteado, no es aplicable a todo los tipos de refactorings. Se presentaron los casos de los refactorings 'Encapsulate Field' y 'Move Class to File' los que posee una complejidad al momento de su realización manual que no permitió identificar un único método de invocación posible para este, y que posiblemente de haber forzado el trabajo para incluirlos, el resultado hubiera tenido menos usabilidad que las herramientas actuales. Dado que B-Refactoring está aún en fase de prototipo, el módulo de ejecución del refactoring aún no puede procesar código fuente de gran complejidad y alto grado de dependencia, como lo hacen las herramientas comerciales actuales. Con lo que, si bien se puede detectar que el refactoring manual que se está llevando a cabo, en ciertos caso el código refactorizado no es el deseado.

8 CONCLUSIONES

Con el presente trabajo se ha demostrado que mediante el análisis del comportamiento del programador podemos diseñar herramientas de refactoring automático que brindan un mayor grado de usabilidad que las herramientas actuales. Presentamos B-Refactoring una herramienta que usa como método de invocación las acciones que el propio desarrollador comúnmente efectúa al hacer un refactoring de manera manual sin entorpecer su flujo de trabajo cotidiano sino simplemente lo complementamos con la opción de realizar el refactoring de manera automática. Además B-Refactoring elimina la necesidad de configuraciones extras con lo que logramos un proceso más intuitivo y eficiente en términos de minimizar los tiempos de ejecución de los refactoring, si los comparamos con los enfoques tradicionales.

El método propuesto abre un abanico de posibilidades de diseño e implementación de una nueva generación de herramientas de refactoring más usables, que se basen en cómo el desarrollador realmente trabaja, y por ende deben lograr una mayor aceptación y uso. Si se logra esto se podrán disfrutar de los beneficios que la herramienta puede traer sobre el código fuente que, al fin y al cabo, repercutirá en un ahorro de tiempo, minimización de errores y aumento de la calidad del software desarrollado.

Se tiene planificado realizar un estudio para determinar el grado de confiabilidad de la herramienta en cuanto al grado de exactitud para detectar un refactoring manual (falsos positivos y falsos negativos) y la calidad del código generado. Finalmente, se planea realizar una investigación a largo plazo para analizar y evaluar la utilidad de B-Refactoring asistiendo a los desarrolladores, a fin de responder la pregunta ¿Teniendo disponible B-Refactoring en su entorno de trabajo el desarrollador incrementa el uso de refactorings automáticos en su día a día?

9 REFERENCIAS

1. Stephen R. Foster, William G. Griswold, and Sorin Lerner. 2012. WitchDoctor: IDE support for real-time auto-completion of refactorings. In Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012). IEEE Press, Piscataway, NJ, USA, 222-232.
2. Erica Mealy, David Carrington, Paul Strooper, and Peta Wyeth. 2007. Improving Usability of Software Refactoring Tools. In Proceedings of the 2007 Australian Software Engineering Conference (ASWEC '07). IEEE Computer Society, Washington, DC, USA, 307-318.
   Emerson Murphy-Hill and Andrew P. Black. 2012. Programmer-Friendly Refactoring Errors. IEEE Trans. Softw. Eng. 38, 6 (November 2012), 1417-1431.
3. Emerson Murphy-Hill and Andrew P. Black. 2008. Breaking the barriers to successful refactoring: observations and tools for extract method. In Proceedings of the 30th international conference on Software engineering (ICSE '08). ACM, New York, NY, USA, 421-430.
4. Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Roshanak Zilouchian Moghaddam, and Ralph E. Johnson. 2011. The need for richer refactoring usage data. InProceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU '11). ACM, New York, NY, USA, 31-38.
5. Dustin Campbell and Mark Miller. 2008. Designing refactoring tools for developers. InProceedings of the 2nd Workshop on Refactoring Tools (WRT '08). ACM, New York, NY, USA, Article 9 , 2 pages.
6. Emerson Murphy-Hill. 2009. Programmer Friendly Refactoring Tools. Ph.D. Dissertation. Portland State University, Portland, OR, USA.
7. Emerson Murphy-Hill and Andrew P. Black. 2007. High velocity refactorings in Eclipse. InProceedings of the 2007 OOPSLA workshop on eclipse technology eXchange (eclipse '07). ACM, NY, USA, 1-5.
8. Emerson Murphy-Hill. 2007. Activating refactorings faster. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion(OOPSLA '07). ACM, New York, NY, USA, 925-926.
9. Emerson Murphy-Hill and Andrew P. Black. 2006. Tools for a successful refactoring. InCompanion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA '06). ACM, New York, NY, USA, 694-695.
10. Xi Ge, Quinton L. DuBose, and Emerson Murphy-Hill. 2012. Reconciling manual and automatic refactoring. In Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012). IEEE Press, Piscataway, NJ, USA, 211-221.
11. Yun Young Lee, Nicholas Chen, and Ralph E. Johnson. 2013. Drag-and-drop refactoring: intuitive and efficient program transformation. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 23-32.
12. Mohsen Vakilian, Nicholas Chen, Stas Negara, Balaji Ambresh Rajkumar, Brian P. Bailey, and Ralph E. Johnson. 2012. Use, disuse, and misuse of automated refactorings. In Proceedings of the 2012 International Conference on Software Engineering (ICSE 2012). IEEE Press, Piscataway, NJ, USA, 233-243.
13. Veselin Raychev, Max Schäfer, Manu Sridharan, and Martin Vechev. 2013. Refactoring with synthesis. SIGPLAN Not. 48, 10 (October 2013), 339-354.

14. Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. 2009. How we refactor, and how we know it. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 287-297.
15. Mohsen Vakilian, Nicholas Chen, Roshanak Zilouchian Moghaddam, Stas Negara, and Ralph E. Johnson. 2013. A compositional paradigm of automating refactorings. In Proceedings of the 27th European conference on Object-Oriented Programming (ECOOP'13), Giuseppe Castagna (Ed.). Springer-Verlag, Berlin, Heidelberg, 527-551.
16. Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E. Johnson, and Danny Dig. 2013. A comparative study of manual and automated refactorings. In Proceedings of the 27th European conference on Object-Oriented Programming (ECOOP'13), Giuseppe Castagna . Springer-Verlag, Berlin, Heidelberg, 552-576.
17. Marija Katic and Kresimir Fertalj. 2009. Towards an appropriate software refactoring tool support. In Proceedings of the 9th WSEAS international conference on Applied computer science (ACS'09), Roberto Revetria, Valeri Mladenov, and Nikos Mastorakis (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 140-145.
18. Huiqing Li and Simon Thompson. 2012. Let's make refactoring tools user-extensible!. InProceedings of the Fifth Workshop on Refactoring Tools (WRT '12). ACM, New York, NY, USA, 32-39.
19. Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2012. A field study of refactoring challenges and benefits. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12). ACM, New York, NY, USA, Article 50 , 11 pages.
20. Zhenchang Xing and Eleni Stroulia. 2006. Refactoring Practice: How it is and How it Should be Supported - An Eclipse Case Study. In Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06). IEEE Computer Society, Washington, DC, USA, 458-468.
21. M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 1999.
22. W. Opdyke and R. E. Johnson, "Refactoring, An Aid in Designing Application Frameworks and Evolving Object-oriented Systems" in SOOPA, 1990.
23. Tom Mens and Tom Tourwé. 2004. A Survey of Software Refactoring. IEEE Trans. Softw. Eng. 30, 2 (2004), 126-139.
24. Emerson Murphy-Hill and Andrew P. Black. 2008. Refactoring Tools: Fitness for Purpose. IEEE Softw. 25, 5 (September 2008), 38-44.
25. Sampieri Hernández, Roberto. Metodología de la investigación. McGraw Hill Interamericana. México, D.F 2003.
26. Y. Kataoka, T. Imai, H. Andou, T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring," 2013 IEEE International Conference on Software Maintenance, p. 0576, 18th IEEE International Conference on Software Maintenance (ICSM'02), 2002.
27. J. Benn, C. Constantinides, H.K. Padda, K.H. Pedersen, F. Rioux and X. Ye. Reasoning on Software Quality Improvement with Aspect-Oriented Refactoring: A Case Study. In W.T. Tsai and M.H. Hamza, editors, Proceedings of the 2005 Software Engineering and Applications, Phoenix, AZ, USA, 2005.
28. R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi, "A Case Study in Refactoring a Legacy Component for Reuse in a Product Line," presented at ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005.
29. J. Ratzinger, M. Fischer, and H. Gall, "Improving evolvability through refactoring," presented at MSR '05: Proceedings of the 2005 international workshop on Mining software repositories, 2005.
30. R. Moser , A. Sillitti , P. Abrahamsson, and G. Succi, Does refactoring improve reusability?, Ninth International Conference on Software Reuse (ICSR-9), Turin, Italy, 11-15 June 2006 .
31. Jakob Nielsen, Usability Inspection Methods, published by John Wiley & Sons, New York 1994
32. Jakob Nielsen, Usability Engineering, published by Academic press limited, Londres 1993
33. Emerson Murphy-Hill, Andrew P. Black, Danny Dig, and Chris Parnin. 2008. Gathering refactoring data: a comparison of four methods. In *Proceedings of the 2nd Workshop on Refactoring Tools* (WRT '08). ACM, New York, NY, USA, Article 7, 5 pages.