6. Conclusiones.

6.1 Conclusión.

El trabajo de nuestra investigación siguió tres diferentes direcciones. Por una parte se centró en desarrollar un API útil que estuviera a cargo de automatizar y facilitar la modificación de ensamblados binarios. Por otro lado, también se concentró en proveer un framework con la suficiente flexibilidad que permitirá a terceros investigadores crear nuevas reglas de refactorización con el objetivo de mejorar las existentes o continuar la investigación en otras direcciones. Y finalmente y más importante, se centró en crear reglas de refactorización que pudieran romper dependencias entre ensamblados. Estos tres caminos convergieron al final de la investigación, produciendo una aplicación que fuera capaz de proveer una solución final para el escenario de dependencias contenidas, y más

aún una infraestructura que facilita la investigación de otros retos como continuar con la investigación de las dependencias basadas en API ó incluso dependencias ocultas.

Afterex fue el nombre asignado para esta aplicación, el cual es un juego de palabras de Extensibility after-the-fact. Su arquitectura fue basada en un diseño de tres capas: El framework o API, la capa de implementación y la capa de aplicación. El diseño permitió al framework que fuera reusable, y que las otras capas fueran extensibles. A través de la aplicación basada en consola ofrecida, es posible que los desarrolladores puedan aplicar las reglas de refactorización creadas en la capa de implementación. Esta aplicación también tomó la responsabilidad de generar los archivos de salida.

Usando la aplicación Afterex, un conjunto de experimentos y aplicaciones ejemplo fueron realizados, con el propósito específico de evaluar su uso así como destacar sus características más importantes. Se puso una atención especial a los escenarios de solo-implementación, dado que fue el único al que se pudo lograr una solución completa. El primer ejemplo que se desarrolló fue una aplicación que mostraba la fecha actual del sistema en la consola. La librería principal contenía una dependencia de tipo contenida al tipo que contenía la implementación de cómo se mostraba la fecha. Después de la refactorización, la aplicación funcionó exactamente igual, pero ahora se podía satisfacer esta dependencia con otras implementaciones. Para este caso en específico, se mostró como se podía cambiar la implementación para que en lugar de imprimir la fecha en consola, lo hiciera en una ventana. Todo esto fue realizado sin editar código fuente existente, ó recompilar. El otro ejemplo fue uno más real. Se trató de un componente muy usado en la mayoría de las aplicaciones, un componente que genera bitácoras a partir de mensajes de error. El componente original era uno que guardaba los mensajes en un archivo de texto.

Después de ejecutar Afterex, fue posible crear nuevos comportamientos como generar archivos XML o mostrar los mensajes en tiempo real en la consola. Este ejemplo principalmente mostró, que tan útil puede ser la herramienta propuesta al momento de crear nuevos componentes e inyectar estos automáticamente.

Al final, los tres objetivos fueron alcanzados, ya que ahora tenemos una herramienta que será la base para investigación a futuro que relacione la refactorización de ensamblados binarios.

6.2 Trabajo a futuro.

El trajo realizado durante nuestra investigación nos dio una visión de la complejidad de los distintos tipos de dependencias, basados en las características de cada uno de los escenarios. Tomando en cuenta los resultados de nuestros experimentos, podemos decir que creamos una prueba de concepto de una aplicación que puede dar una solución total al escenario de solo-implementación. Aprendimos que la mayoría de las veces este tipo de dependencias pueden ser rotas, incluso si no hay una clase base. También aprendimos que este escenario es muy común en situaciones reales.

En caso del escenario de dependencias directas aprendimos que es un escenario mucho más difícil que uno de dependencias contenidas. Además es más difícil que pase en la vida real. En este caso no pudimos llegar a una solución final, sin embargo, investigamos algunas ideas de cómo atacar algunos de sus sub-casos. Hay más trabajo que hacer en este escenario, como por ejemplo tratar a los métodos estáticos, herencia, tipos genéricos, etc.

Además la infraestructura que creamos puede ser utilizada para continuar con la investigación de otro tipo de dependencias, como las escondidas o las indirectas. Pero en

general el framework/API puede ser utilizado en cualquier proyecto que involucre la modificación de ensamblados o la generación de código intermedio.

Por otro lado, gracias a la investigación realizada nos dimos cuenta que puede ser posible crear un contenedor de inyección de dependencias para aplicaciones existentes, basado en las ideas estudiadas.

Finalmente, el API puede ser extendido para soportar operaciones que estén relacionadas y que puedan enriquecer su funcionamiento, para que finalmente se llegue a una madurez para poder en fusionarlo con el API provisto por CECIL.