Identificación de los principios de segmentación y extracción de TRIZ en patrones de diseño

María Erika Auryly García-Cantú, Ulises Juárez-Martínez, Guillermo Cortés-Robles

División de Estudios de Posgrado e Investigación
Instituto Tecnológico de Orizaba
merika.gc@gmail.com, ujuarez@ito-depi.edu.mx,
gc_robles@hotmail.com

Resumen. Los patrones de diseño permiten la encapsulación y reutilización de componentes de software debiendo permitir la sustitución entre patrones. Sin embargo, dicha sustitución muchas veces se dificulta ya que no se tienen principios de diseño definidos para hacerla, muchas veces se hace por la intención del patrón o de acuerdo a la experiencia del diseñador. Por otro lado, TRIZ ofrece, entre otras herramientas, los principios inventivos que permiten la solución de problemas resolviendo contradicciones de forma no convencional. El presente artículo tiene como objetivo presentar las analogías entre los primeros dos principios inventivos de TRIZ (segmentación y extracción) y los patrones de diseño de software.

Palabras clave: Diseño de patrones, Sustitución, TRIZ, Principios inventivos.

1. Introducción

Diseñar software orientado a objetos puede resultar difícil, y aún más diseñar software orientado a objetos que sea reutilizable. Hay que encontrar los objetos pertinentes, factorizarlos en clases, definir interfaces y jerarquías de herencia y establecer relaciones entre clases y objetos [1]. Además, hay que tomar en cuenta los principios de diseño orientados a objetos: Principio de responsabilidad única (SRP), Principio Abierto/Cerrado (OCP), Principio de Sustitución de Liskov (LSP), Principio Inversión de la Dependencia (DIP), Principio de Segregación de Interfaz (ISP), los cuales ayudan a evitar síntomas de mal diseño [2]. Sin embargo, los diseñadores expertos saben que no hay que resolver cada problema desde cero, sino que usan una y otra vez una solución que les funcionó en el pasado. De esta forma se encuentran patrones que resuelven problemas concretos de diseño y hacen que los diseños orientados a objetos sean más flexibles, elegantes y reutilizables. Por lo tanto, un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico [1].

Tendencias Tecnológicas en Computación
Research in Computing Science 66, 2013, pp. 17-26
Paper received 14-05-2013 and accepted 02-08-2013

Los patrones de diseño permiten la definición de elementos reutilizables a nivel de diseño y algunos de ellos a nivel de código. Partiendo de aquellos patrones encapsulados, es necesario exhibir interfaces requeridas e interfaces esperadas para su acoplamiento con otros componentes. La sustitución o intercambio de patrones de diseño es factible, sin embargo, muchas veces dicha sustitución está basada en la intención del patrón y no en la interfaz o algún otro principio de diseño, dificultando una verdadera modularidad y evolución de las arquitecturas de software.

TRIZ son las siglas rusas para la Teoría de la Resolución de Problemas Inventivos. El desarrollo de esta metodología fue iniciada en 1946 por Genrikh Altshuller (1926-1998). Se trata de una metodología de resolución de problemas basada en un acercamiento lógico y sistemático desarrollada a partir del análisis de miles de patentes y con el análisis de la evolución de la tecnología. TRIZ se emplea como instrumento intelectual para solucionar problemas técnicos y tecnológicos, sencillos y difíciles, con mayor rapidez y mejores resultados [3].

Para que sea posible resolver un problema con TRIZ, el problema necesita tener por lo menos una contradicción. Si no hay ninguna contradicción técnica, entonces no se trata de un problema inventivo (no es un problema TRIZ) [3].

Una de las herramientas de TRIZ son los principios inventivos, los cuales se utilizan para la solución de contradicciones técnicas y resolver problemas de forma no convencional. El trabajo realizado por Elías Beltrán [4, 5] muestra que es factible establecer mediante los principios inventivos, una forma diferente de aplicar y sustituir patrones de diseño para el software.

Este trabajo presenta las analogías identificadas en los principios inventivos de segmentación y extracción de TRIZ con los 23 patrones de diseño de software, las cuales refinan el trabajo de Beltrán, utilizando principios de objetos, mostrando porqué caen los patrones en cada principio inventivo.

Se presentan en primer lugar los trabajos relacionados, seguidos de la descripción de los principios inventivos y patrones con los que se trabajó para, posteriormente, mostrar los resultados obtenidos. Asimismo, se presenta una sección de discusión donde se explican las diferencias con los trabajos anteriores, además de las conclusiones y trabajo a futuro.

2. Trabajos relacionados

A continuación se describe el conjunto de trabajos relacionados con el tema.

La aplicación de TRIZ en el ámbito del software es un campo no muy experimentado, hay relativamente pocas publicaciones sobre ello. Se ha aplicado en diferentes áreas como la arquitectura de software [6, 7], en [8] se utiliza como apoyo para el diseño de un software de innovación asistida por computadora. También se toman las soluciones inventivas de TRIZ en algoritmos de procesos de innovación [9]. En [10] presentan 2 casos de TRIZ aplicado a los Servicios de ITs y a la Inteligencia de Ne-

gocios. Asimismo hay un estudio de la teoría de TRIZ para resolver problemas de calidad del software [11]. En [12] se presenta una aplicación de TRIZ y Six Sigma para mejorar el proceso de desarrollo de software. En [13] Nakagawa revisa aspectos de ingeniería de software para ser utilizados con TRIZ; de igual forma, en [14] Kevin Rea estudia la solución de concurrencia mediante TRIZ. Sin embargo, se reportan pocas publicaciones sobre la aplicación de TRIZ en el diseño de software.

En [15] Aarti Goyal et al., describen un método basado en herramientas y principios de TRIZ que aplican para mantener la calidad de los modelos orientados a objetos. Los criterios de calidad fueron mapeados en un conjunto de reglas y prácticas de buen diseño. Se menciona que las herramientas UML no tienen el soporte adecuado para los modelos evolutivos, lo cual resulta en modelos menos mantenibles y reutilizables, y en muchos casos solo se usan como documentación. La evaluación mejorada que proporciona TRIZ será útil en la evaluación de la calidad con anticipación para desarrollar conciencia acerca de los problemas de calidad.

Ma JianHong et al., presentan una investigación relativa al Diseño Orientado a Objetos (Object-Oriented Design, OOD) aplicando TRIZ [16]. Se expresa que la dependencia de las relaciones entre los objetos afecta características importantes de los sistemas como la capacidad de prueba, fiabilidad o facilidad de mantenimiento. Inspirándose en la matriz de contradicciones de la teoría de TRIZ, el documento propone una matriz de contradicción enfocada en el diseño orientado a objetos basándose en el estudio de los patrones de diseño, así como los problemas del diseño de software orientado a objetos. Se construyó una matriz de 22x22 usando 22 parámetros encontrados. Además, se presenta un breve ejemplo de cómo utilizar la matriz para solucionar un problema de diseño. Éste trabajo da la pauta para aplicar TRIZ en el diseño orientado a objetos a partir del análisis de los patrones de diseño, sin embargo, no relaciona los principios inventivos de TRIZ con las características de los patrones, más bien propone una matriz de contradicción parecida a la de TRIZ para solucionar los problemas de OOD.

Una observación temprana de las similitudes entre los patrones de diseño y TRIZ es que ambos sistemas son un conjunto de heurísticas, derivadas de la observación de soluciones exitosas a problemas comunes. Se encuentran fuertes relaciones entre los patrones estructurales de la Gang of Four (GoF) y conjuntos específicos de los principios de TRIZ. En [17] Ellen Domb y John Stamey examinan los siete patrones estructurales, en su nivel conceptual más alto, como instancias de alguno de los 40 principios inventivos de TRIZ y su correspondiente relación con los principios. Aún y cuando sólo se analizaron los patrones estructurales, se hace mención de la intención que se tiene de completar los 23 patrones. Éste documento fue de los primeros en relacionar TRIZ con patrones de diseño, a pesar de que la idea de utilizar TRIZ en el ámbito del software surgió varios años antes.

En [4] Elías Beltrán expone cómo acercar la implementación de patrones de diseño usando principios inventivos de TRIZ, analizando las relaciones que existen entre ambos conceptos abstractos. Pretende generar una herramienta que asista el diseño de software, seleccionando los patrones más convenientes según los principios que se

implementen, lo cual permite mejorar la adaptación y evolución del software. Como hay principios inventivos que se refieren a características físicas y/o químicas, es más difícil encontrar una relación con los patrones de diseño, puesto que el software es lógico. Beltrán explica el análisis que se llevó a cabo para identificar las cinco correspondencias más complicadas entre los conceptos de TRIZ y patrones de diseño, listando los patrones de diseño que se adaptan a la descripción de cada principio inventivo. Posteriormente, el autor presenta una tabla con las relaciones entre los patrones y los principios inventivos, además de un ejemplo de aplicación que muestra cómo utilizar dicha tabla.

De igual forma, Beltrán en [5] propone el uso de los principios inventivos de TRIZ con un enfoque hacia los patrones de diseño de software. El objetivo es probar la capacidad de sustitución entre los patrones de diseño y así fomentar la reutilización de los mismos. Beltrán analiza los 23 patrones de diseño de GoF y los 8 más significativos del paradigma de Programación Orientada a Aspectos (POA). Para lograr el objetivo planteado, primero se analizaron detenidamente cada uno de los principios inventivos de TRIZ así como los patrones de diseño, para después establecer correspondencias con el desarrollo de software. Se logró establecer una relación entre los problemas en el desarrollo de software y los principios inventivos más frecuentes que los solucionan, también se establece una relación entre los principios inventivos y las sugerencias para un buen diseño. Como los patrones de diseño se relacionan entre sí dependiendo de las características que tengan entre ellos, se logró determinar las relaciones existentes. El documento toma dos enfoques: se relacionaron los principios inventivos con los patrones de diseño tomando como base los principios para encontrar qué patrones se relacionaban entre ellos y porqué, pero también se hizo a la inversa, tomando como base los patrones de diseño, para encontrar los principios que se relacionan con cada patrón, así se identificaron los patrones que mantienen en común los mismos principios. De igual forma, se encontraron las similitudes que permiten la sustitución entre patrones usando principios inventivos. El trabajo realizado también proporciona una base para determinar qué patrón de diseño es adecuado para utilizarse de acuerdo a los principios y sugerencias obtenidos con TRIZ.

3. Principios inventivos y patrones de diseño estudiados

Beltrán [4, 5] realizó un trabajo previo identificando las analogías entre los principios inventivos de TRIZ y los 23 patrones de diseño según GoF [1], sin embargo, dicho trabajo no consideró conceptos como encapsulación, herencia y polimorfismo, y debe ser refinado y documentado para comprobar que el intercambio es factible y así, una vez que se documenten las analogías, desarrollar una guía que facilite y favorezca el intercambio de patrones.

El trabajo de identificar las relaciones entre principios inventivos y patrones requiere un largo análisis, es por esto que se presentan las analogías de sólo 2 principios inventivos.

A continuación se describen los conceptos necesarios para la mejor comprensión del presente trabajo.

Los principios inventivos de TRIZ se usan para resolver contradicciones, se basan en el mismo estudio de patentes y tecnología que desarrolló los patrones de evolución. Por esto, es posible aplicar los mismos principios a problemas de diferentes ámbitos. Existen 40 principios inventivos, sin embargo, sólo se describen los dos primeros, de los cuales se tienen resultados [3].

1. ***Segmentación***. Fragmentación. Divide un objeto o sistema en partes independientes. Hace un objeto fácil de desarmar. Este principio tiene 3 casos: A - divide un objeto en partes independientes; B - hace un objeto desmontable para un fácil ensamblado y desensamblado y, C - incrementa el grado de segmentación de un objeto.

2. ***Extracción***. Separa la parte (o propiedad) necesaria o elimina una parte que interfiere con el objeto o sistema. El principio presenta 2 casos: A – extraer la parte o propiedad "inquietante" de un objeto; B – extraer sólo la parte o propiedad necesaria de un objeto.

Un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico [1]. A continuación se describen los patrones de diseño de los cuales se identificaron, refinaron y complementaron las analogías hechas por Beltrán [4, 5] con alguno de los dos principios inventivos (los resultados de éstas analogías se muestran en la sección 4).

- *Builder.* Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- *Bridge.* Desacopla una abstracción de su implementación, de modo que ambas puedan variar de forma independiente.
- *Flyweight.* Usa compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.
- *Facade.* Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.
- *Chain of responsibility.* Evita acoplar el emisor de una petición a su receptor, dando a más de un objeto la posibilidad de responder a la petición. Encadena los objetos receptores y pasa la petición a través de la cadena hasta que es procesada por algún objeto.
- *Mediator.* Define un objeto que encapsula cómo interactúan una serie de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.
- *Proxy.* Proporciona un representante o sustituto de otro objeto para controlar el acceso a éste.
- *Factory Method.* Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

- *Abstract Factory*. Proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.
- *Memento.* Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.

4. Relación entre principios inventivos y patrones de diseño

En ésta sección se presentan los resultados de la relación entre los principios inventivos de TRIZ con los patrones de diseño de software obtenidos hasta el momento. Estos resultados logran complementar el trabajo de Beltrán en [4, 5], ya que se realizó un análisis más profundo de cada patrón para conseguir una relación con los principios inventivos 1 y 2, además de quedar documentada la razón por la cual cada patrón presentado tiene relación con alguno de los principios.

Segmentación. El concepto de segmentación se refiere a que sin importar que estén separadas, todas las partes de un conjunto permiten lograr el propósito que se busca. Todas las partes son necesarias. En la tabla 1 se muestran las relaciones obtenidas entre el principio de segmentación y los patrones de diseño que lograron identificarse.

Tabla 1. Relación del principio de segmentación en patrones de diseño.

Patrón de diseño	Características
Builder	Entra en el principio de segmentación en el caso B porque separa el proceso de construcción. El algoritmo para crear un objeto debe ser independiente de las partes que se compone y cómo se ensambla.
Bridge	Entra en el caso A porque hace la abstracción e implementación independientes y trabajan en conjunto en el nivel más alto de la jerarquía.
Flyweight	Entra en el caso A porque divide los estados de un objeto en intrínseco y extrínseco. Hace las partes independientes, pero a la vez, cada objeto trabaja en conjunto con el objeto intrínseco.

Extracción. El principio de extracción se refiere a la acción de extraer, recuperar o remover un objeto. En la tabla 2 se muestran las relaciones identificadas entre el principio de extracción y los patrones de diseño.

Tabla 2. Relación del principio de extracción en patrones de diseño.

Patrón de diseño	Características
Facade	Entra en el caso A porque reubica las invocaciones que existen entre las clases que utilizan un subsistema. Las llamadas se pasan de un lugar a otro, a nivel del intercambio de mensajes entre los objetos o clases.
Bridge	Conceptualmente cae en el principio de extracción, estructuralmente en realidad no hay separación, hay una introducción de nuevos elementos para lograr el propósito del patrón.
Mediator	Se encuentra en el caso A, ya que las llamadas se pasan de un lugar a otro, a nivel del intercambio de mensajes. La interacción de objetos es a través del intercambio de mensajes.
Proxy	Se asocia al caso B porque extrae el objeto a sustituir y el proxy controla el acceso a éste. Se considera el concepto de remover porque pasa el objeto de un lugar a otro dando el efecto de posponer el acceso a él.
Chain of responsibility	Cae en el caso B porque extrae la petición (la encapsula) y la mueve entre objetos para que alguno de esos objetos la procese.
Flyweight	Se relaciona con el caso B ya que extrae el estado intrínseco de un objeto, cambiándolo de posición (encapsulación al tenerlo en otro objeto) y lo recupera al momento de compartir ese estado intrínseco con cada uno de los objetos que lo requiera.
Factory method	Se encuentra en el caso B. El patrón maneja una jerarquía paralela lo que permite mover un subconjunto del comportamiento de la jerarquía existente a la nueva. Este comportamiento es la instanciación.
Abstract Factory	Condición: se observa que si en el manejo de la creación de la familia de productos se hace de manera paralela, tiene elementos para considerarse dentro de éste principio. Si no lo hace, no cumple con el principio.
Memento	Presenta extracción como estrategia interna del patrón. Extrae y recupera el estado.

A pesar de que es complicado encontrar una relación entre los principios inventivos de TRIZ con los patrones de diseño, se puede observar que efectivamente existe una correlación entre ambos. Para ello, es necesario comprender correctamente los conceptos de ambas áreas.

5. Discusión

En esta sección se explican las diferencias encontradas en relación con el trabajo de Beltrán [4, 5] y con el de Stamey y Domb [17].

Beltrán reporta *Builder, Bridge* y *Visitor* para el principio de Segmentación, sin embargo, de acuerdo al análisis realizado, se descartó el patrón *Visitor* porque no cumple con el objetivo del principio de segmentación; en cambio, se añadió el patrón *Flyweight* debido a que presenta características relativas al propósito del principio. En [17] se reporta el patrón *Bridge* ligado solamente al principio de Extracción, y al *Flyweight* con el principio de transición a una nueva dimensión (principio 17) del cuál aún no se tiene un análisis realizado. No obstante, se demuestra que dichos patrones se pueden asociar con otros principios si se logra comprender a fondo la estructura y propósito del patrón.

De igual forma, Beltrán relaciona los patrones *Facade, Flyweight, Proxy* y *Chain of Responsibility* con el principio de Extracción. A diferencia de Beltrán, Domb y Stamey relacionan con este principio únicamente al patrón Bridge. No obstante, se logró identificar que los patrones *Mediator, Factory Method, Abstract Factory* y *Memento* también presentan una relación con el principio.

Es importante mencionar que los trabajos anteriores [4, 5, 17] abordan el tema de forma superficial. Sin embargo, el análisis realizado en el presente artículo se hizo desde una perspectiva más a fondo en la parte de Orientación a Objetos, ya que se consideraron conceptos de objetos como encapsulación, herencia y polimorfismo, además de tomar en cuenta la estructura, función y forma en que trabajan los patrones de diseño, lo cual no se considera en los trabajos anteriores. Esto permitió un análisis y comprensión más profundos de los principios, de manera que el trabajo presentado refina y complementa el trabajo previo [4, 5].

6. Conclusiones

Como se comentó a lo largo del presente artículo, los patrones de diseño permiten la reutilización de componentes de software [1]. Por otro lado, TRIZ ofrece los principios inventivos para solucionar contradicciones de forma no convencional [3].

Con base en el trabajo realizado por Beltrán [4, 5] donde se demuestra que los patrones de diseño de software tienen una correlación con los conceptos de los principios inventivos de TRIZ, el presente trabajo toma relevancia ya que encamina al refinamiento en las analogías entre principios inventivos y patrones de diseño al considerar conceptos de objetos que los otros trabajos no contemplan. Esto permite además, obtener un análisis con mayor profundidad de los patrones desde una perspectiva no convencional de diseño y promover su utilización a partir de un enfoque sistemático con los principios inventivos, además de documentar dicho análisis. Asimismo, presenta un alto grado de innovación puesto que aún y cuando TRIZ se usa en diferentes áreas de ingeniería [14], en el ámbito de los sistemas de software no se reportan trabajos realizados con la profundidad ni el alcance que tiene el presente trabajo.

7. Trabajo a futuro

Analizar y documentar la correlación de los 38 principios inventivos de TRIZ restantes (principios 3 al 40) con los patrones de diseño de software.

Determinar qué principios son congruentes con las tendencias de evolución estáticas, dinámicas y/o cinemáticas con el objetivo de plantear tendencias de evolución en la sustitución de patrones.

Agradecimientos

Se agradece al Consejo Nacional de Ciencia y Tecnología (CONACyT) y a la Dirección General de Educación Superior Tecnológica (DGEST) por el apoyo brindado para la realización del presente trabajo.

Referencias

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
2. R. C. Martin, M. C. Feathers, T. R. Ottinger, J. J. Langr, B. L. S. J. W. Grenning, and K. D. Wampler, Clean Code: A Handbook of Agile Software Craftsmanship. Robert C. Martin Series, Upper Saddle River, NJ: Prentice Hall, August 2008.
3. K. Rantanen and E. Domb, Simplified TRIZ. , Auerbach Publication: Auerbach Publication, 2008.
4. Elías Beltrán, Ulises Juárez, Guillermo Cortés. Patrones de diseño de software con principios inventivos. 7o Congreso Iberoamericano de Innovación Tecnológica & Desarrollo de Productos 2012, Orizaba, Veracruz. Noviembre 2012.
5. Elías Beltrán, Ulises Juárez, Guillermo Cortés. TRIZ en el desarrollo de arquitecturas de software. 6° Congreso Iberoamericano de Innovación Tecnológica 2011, Querétaro, Querétaro. Octubre 2011.
6. Daniel Kluender. TRIZ for software architecture. ScienceDirect, Procedia Engineering, Elsevier Ltd. vol. 9, pp. 708-713, 2011.
7. Song-Kyoo Kim. Design of Enhanced Software Protection Architecture by Using Theory of Inventive Problem Solving. In Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management, 2009. IEEM 2009. IEEE Computer Society, Hong Kong, 8-11 Dec. 2009, pp.978-982.
8. Cao Guozhong, Guo Haixia, Yu Jiang, Tan Runhua. Computer Aided Product Innovation Software Based on Extended-Effect and its Application. In Proceedings of the Software Engineering, 2009. WRI World Congress WCSE '09, vol.2, IEEE Computer Society, Xiamen, China, 19-21 May 2009, pp.63-67.

9. Brad, M. Fulea, B. Mocan, A. Duca, E. Brad. Software Platform for Supporting Open Innovation. In Proceedings of the IEEE International Conference on Automation, Quality and Testing, Robotics, AQTR 2008, vol.3, IEEE Computer Society, Cluj-Napoca, Romania, 22-25 May 2008, pp.224-229.
10. Kas Kasravi. Applications of TRIZ to IT: Cases and Lessons Learned. In Proceedings of the TRIZCON 2010. Altshuller Institute, Dayton, Ohio. Oct. 2010.
11. Su-Hua Wang, Durgesh Samadhiya, Dengjie Chen. Software Development and Quality Problems and Solutions by TRIZ. In Proceedings of the International Symposium on Frontiers in Ambient and Mobile Systems (FAMS-2011). Vol. 5. Procedia Computer Science, Ontario, Canada**,** 2011, 730-735.
12. Garikapati Pavan Kumar. Software Process Improvement –TRIZ and Six Sigma (Using Contradiction Matrix and 40 Principles). TRIZ Journal. CTQ Media. April 2005.
13. Toru Nakagawa. Software engineering and TRIZ – Structured Programming Reviewed with TRIZ. In Proceedings of TRIZCON 2005, Altshuller Institute, April 2005.
14. Kevin C. Rea. Using TRIZ in Computer Science – Concurrency. TRIZ Journal, CTQ Media. August 1999.
15. S. B. Goyal, Aarti Goyal, Pratima Sharma, Neha Singhal. Analyzing Object Models with Theory of Innovative Solution. In Proceedings of the 2012 Second International Conference on Advanced Computing & Communication Technologies (ACCT '12). IEEE Computer Society, Washington, DC, USA, 46-50.
16. Ma JianHong, Zhang Quan, Wang Yanling, Zhang Wei. Research and Application of the TRIZ Contradiction Matrix in OOD. In Proceedings of the WRI World Congress on Software Engineering, 2009. WCSE '09, vol.3, IEEE Computer Society, Xiamen, China, 19-21 May 2009, pp.247-251.
17. John W. Stamey, Ellen Domb. Workshop: communicating design patterns with TRIZ. In Proceedings of the 24th annual ACM international conference on Design of communication (SIGDOC '06). ACM, New York, NY, USA, 2006, 131-133.