Herramienta basada en computación evolutiva interactiva para arquitectos software

Aurora Ramírez, Rafael Barbudo, José Raúl Romero y Sebastián Ventura

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba, España
{aramirez, i22balur, jrromero, sventura}@uco.es *

Resumen Resolver problemas de optimización para los cuales es difícil definir una función de fitness, como ocurre en el desarrollo del software, requiere considerar criterios de evaluación subjetivos. En este contexto, la computación evolutiva interactiva propone involucrar al experto en la búsqueda, consiguiendo así que los resultados obtenidos realmente satisfagan sus expectativas. Este tipo de modelos plantean retos tales como la fatiga del usuario o el desarrollo de herramientas que permitan interactuar con el algoritmo. Este trabajo propone un framework evolutivo interactivo para la resolución de un problema multi-objetivo de diseño arquitectónico. Además, se ha desarrollado una herramienta con la que se ha realizado un estudio inicial sobre la adecuación del mecanismo de evaluación propuesto, basado en la valoración de aspectos cualitativos.

Keywords: computación evolutiva interactiva, optimización multi-objetivo, ingeniería del software basada en búsqueda, arquitecturas software, herramienta software

1. Introducción

La computación evolutiva ofrece métodos eficientes para la resolución de problemas de optimización, ya que realizan una búsqueda inteligente sobre el espacio de soluciones guiados por una o varias funciones objetivo. Sin embargo, existen aplicaciones reales donde resulta complejo definir con precisión una función de *fitness* que pueda ser calculada de forma meramente computacional. Es por ello que la computación evolutiva interactiva (*Interactive Evolutionary Computation*, IEC) [1] propone involucrar al experto en el proceso de búsqueda, de forma que sobre él recaen tareas como la evaluación de soluciones o incluso su transformación [2]. En el ámbito de la optimización multi-objetivo, este tipo de métodos suelen estar dirigidos a permitir que el experto establezca puntos de referencia como forma de dirigir la búsqueda hacia zonas concretas del frente de Pareto. Sin embargo, emitir este tipo de opiniones puede no ser suficiente en problemas en los que la evaluación de la calidad es principalmente un aspecto subjetivo.

* Trabajo financiado por el Ministerio de Economía y Competitivad, proyecto TIN2014-55252-P y el Ministerio de Educación, programa FPU (FPU13/01466).

La ingeniería del software presenta multitud de situaciones en las cuales los ingenieros deben tomar decisiones complejas y establecer compromisos entre factores conflictivos, tales como el coste y la calidad del software. Es por tanto habitual encontrar enfoques multi-objetivo en el ámbito de la ingeniería del software basada en búsqueda (*Search Based Software Engineering*, SBSE) [3], donde diferentes técnicas metaheurísticas se emplean para resolver tareas propias del ciclo de vida del desarrollo software. La generación o mejora de artefactos software producidos en las fases de análisis y diseño puede verse especialmente beneficiada por el uso de enfoques interactivos, ya que es donde las habilidades y experiencias del ingeniero resultan más determinantes [4]. En este contexto, aspectos como la fatiga o la inexperiencia en el uso de técnicas metaheurísticas también deben ser tenidos en cuenta tanto a la hora de diseñar el algoritmo interactivo como al construir la herramienta software que lo encapsule [5].

Dentro del área de SBSE, la optimización de arquitecturas software se centra en la aplicación de técnicas de búsqueda para especificar, modificar o reconstruir artefactos de diseño tales como componentes o servicios. En este contexto, el descubrimiento de arquitecturas software basadas en componentes ha sido recientemente formulado como un problema de búsqueda multi-objetivo [6]. Hasta el momento, su resolución se ha abordado mediante un algoritmo evolutivo guiado únicamente por medidas software que evalúan aspectos relacionados con la mantenibilidad. Sin embargo, las medidas software no siempre pueden capturar todos los aspectos relevantes a la hora de evaluar la calidad de un diseño software. Existen otros aspectos, como la elegancia, que influyen más si cabe en las decisiones arquitectónicas tomadas. La dificultad de evaluar automáticamente alternativas de diseño a un nivel de abstracción tan elevado, unido al hecho de que el arquitecto dispone de una visión global del análisis del sistema de la cual carece la máquina, hacen interesante considerar un enfoque interactivo.

Este trabajo presenta una herramienta basada en computación evolutiva multi-objetivo para abordar el descubrimiento de arquitecturas software de forma interactiva. Por un lado, la participación activa del arquitecto permitirá la inclusión de criterios de evaluación subjetivos encaminados a satisfacer sus preferencias de diseño. Por otro lado, el algoritmo evolutivo aportará mecanismos eficientes para discriminar soluciones. La combinación de criterios de evaluación cualitativos y cuantitativos representa un aspecto poco estudiado en el ámbito de la IEC, especialmente cuando se desea mantener la perspectiva multi-objetivo del problema. Para abordarlo, el algoritmo se basa en la definición de *preferencias arquitectónicas* que transforman la opinión del arquitecto sobre aspectos cualitativos de las soluciones en medidas evaluables sobre cualquier otra solución generada. La información extraída de dichas preferencias, junto con la adaptación de técnicas de la IEC para el control de la diversidad y la posibilidad de realizar acciones como la "congelación" de partes del genotipo de las soluciones, dotan al modelo propuesto de elementos suficientes con los cuales abordar el descubrimiento evolutivo de arquitecturas software de una manera más eficaz.

El resto del artículo se estructura como sigue. La Sección 2 describe los conceptos relacionados con el problema de optimización y la IEC. La Sección 3

describe el modelo evolutivo interactivo, cuya evaluación experimental se detalla en la Sección 4. Finalmente, la Sección 5 presenta las conclusiones.

2. Trabajo relacionado

2.1. Descubrimiento evolutivo de arquitecturas software

La tarea del descubrimiento de arquitecturas software basadas en componentes puede formularse como un problema de optimización multi-objetivo [6], en el cual se pretende encontrar la combinación de clases y sus relaciones que mejor representa la arquitectura de alto nivel del sistema software en términos de sus componentes e interfaces. Tomando como entrada un diagrama de clases UML 2, los componentes son especificados a partir de agrupaciones de clases muy relacionadas entre sí, mientras que las interfaces se crean a partir de las relaciones existentes entre clases alojadas en distintos componentes.

La calidad de las arquitecturas candidatas es evaluada mediante varias funciones objetivo formuladas en base a medidas software relacionadas con la mantenibilidad, tales como la modularidad o la reutilización. Las tres medidas empleadas aquí son: ICD (*Intra-modular Coupling Density*), que establece un compromiso entre cohesión y acoplamiento; ERP (*External Relations Penalty*), que penaliza la existencia de relaciones entre componentes que no pueden especificarse a través de interfaces; y GCR (*Groups Component Ratio*), que busca que todas las clases alojadas en un mismo componente estén conectadas entre sí.

2.2. Computación evolutiva interactiva

Los métodos de optimización interactivos engloban a todos aquellos algoritmos en los que un humano participa en el proceso de búsqueda [2]. Este significado amplía la idea original propuesta por la IEC, donde el humano únicamente actuaba como sustituto de la función de *fitness* [1]. Por tanto, es posible encontrar diferentes modelos interactivos dependiendo del rol que toma el humano (desde ajustar parámetros hasta actuar como un operador genético) o de los aspectos sobre los que influye (definición del problema, evaluación de soluciones, etc.) [2]. Por lo general, todo sistema interactivo consta de un mecanismo de búsqueda que, cada cierto periodo de tiempo, muestra resultados intermedios al experto. A continuación, éste retroalimenta al sistema con información subjetiva que debe ser integrada por el algoritmo para continuar la búsqueda en base a ella. Aspectos como la frecuencia con la que interviene el experto, la selección de las soluciones a mostrar o la vigencia de la información han de ser tenidos en cuenta al diseñar un modelo interactivo, si bien son factores que pueden estar condicionados por la naturaleza del problema que se pretende resolver.

Por su propia definición, la resolución de problemas multi-objetivo asume la existencia de un experto que elige, en función de sus propias preferencias, una solución de compromiso de entre el conjunto devuelto por el algoritmo. Por tanto, es lógico que el paso natural a la hora de plantear modelos interactivos haya sido

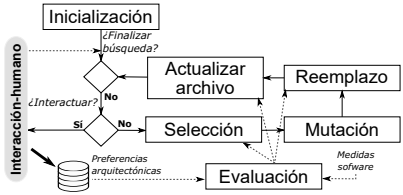

Figura 1. Diagrama de flujo del modelo evolutivo interactivo

el establecimiento de dichas preferencias durante el propio proceso de búsqueda, no sólo al final. De esta forma, el algoritmo es capaz de dirigir la búsqueda hacia uno o varios puntos de referencia, restringiendo así el conjunto final de soluciones. Un algoritmo que explora esta idea es iTDEA (*Interactive Territory Defining Evolutionary Algorithm*) [7], donde el experto debe seleccionar la mejor solución entre las mostradas por el algoritmo para, a partir de ella, definir en qué zonas del frente de Pareto se permite una mayor aglomeración de soluciones. Aunque efectivo, este tipo de propuestas se centra únicamente en el espacio de objetivos, asumiendo por tanto la existencia de funciones objetivo bien definidas.

Sin embargo, esto no suele ocurrir en el ámbito del diseño software, ya que los expertos se basan más en sus percepciones y experiencias a la hora de evaluar la calidad de sus diseños. Es por ello que, dentro de SBSE, la mayoría de enfoques interactivos traten de resolver tareas de diseño. Por ejemplo, en [4] se propone un modelo interactivo donde el usuario interviene para evaluar la elegancia de varios diagramas de clases. Dicha información es luego utilizada para ponderar las medidas software que componen la función de *fitness*. Los diagramas de clases son también el objeto de estudio en [8], si bien aquí representan la arquitectura de bajo nivel de un sistema software, definida en términos de clases y patrones de diseño. El arquitecto interviene sólo para congelar partes de las soluciones.

3. Modelo evolutivo interactivo

La Figura 1 ofrece una visión global del modelo propuesto, el cual parte de la estructura del algoritmo multi-objetivo utilizado en trabajos anteriores [6]. Su funcionamiento ha sido modificado para que, cada cierto número de generaciones, el arquitecto visualice un conjunto de soluciones sobre las que emitirá sus preferencias de diseño. Dicha información será utilizada, junto con las medidas software, para evaluar al resto de soluciones generadas, tal y como se explica en la Sección 3.1. La forma en la que las soluciones son seleccionadas y el resto de acciones que el arquitecto puede realizar son detalladas en la Sección 3.2. Una vez definidos todos estos elementos es posible describir cómo influyen en cada una de las fases del proceso evolutivo, tal y como se recoge en la Sección 3.3.

3.1. Evaluación de soluciones

El modelo interactivo propuesto evalúa las soluciones arquitectónicas atendiendo a criterios objetivos y subjetivos que, además, pueden ser fácilmente extendidos o configurados. Los primeros están formulados en base a las medidas software que se han venido aplicando al problema, obteniendo así una evaluación *objetiva*, f_{obj}. Los segundos hacen referencia a la información que suministra el arquitecto de forma interactiva, y por tanto determinan la calidad *subjetiva*, f_{sub}. La función de *fitness* para un individuo i, que debe ser minimizada, se define como la suma ponderada de ambos términos (ver Ecuación 1).

$$fitness(i) = w_{obj} \cdot f_{obj}(i) + w_{sub} \cdot f_{sub}(i) \quad (1)$$

Evaluación de medidas software. Cada medida software es considerada de forma independiente, por lo que cada individuo i está representado por un conjunto de k funciones objetivo, las cuales deben minimizarse y estar definidas en el mismo rango. Si alguna de las medidas debe maximizarse, como ocurre con ICD, los valores se invierten. A partir de ellas se calcula la función *maximin* [9], que evalúa tanto la dominancia entre las soluciones como su diversidad. La función *maximin* varía en el rango [-1,1], otorgando valores inferiores a 0 a las soluciones no dominadas y valores más próximos a los extremos conforme más distancia haya entre dicha solución y las demás, j. La Ecuación 2 obtiene dicho valor pero definido en el rango [0,1], donde t representa el tamaño poblacional.

$$f_{obj}(i) = \frac{1 + max_{j \neq i}(min_k(f_k^i - f_k^j))}{2} \ \forall\ j\ en\ [1,t] \quad (2)$$

Evaluación de preferencias arquitectónicas. Cabe recordar que las preferencias de diseño establecidas por el arquitecto hacen referencia a criterios cualitativos que éste observa sobre el fenotipo de una solución arquitectónica. Por tanto, cada preferencia, p, debe ser traducida a un valor cuantitativo que pueda ser calculado sobre cualquier otra solución generada por el algoritmo. Este valor se obtiene mediante la función $preferencia_p$, y siempre varía en el rango [0,1], donde valores cercanos a 1 representan que el individuo satisface dicha preferencia en gran medida. La componente subjetiva de la función de *fitness* se obtiene a partir del valor medio del grado de cumplimiento de las P preferencias acumuladas a lo largo de la evolución (ver Ecuación 3).

$$f_{sub}(i) = 1 - \frac{1}{P} \cdot \sum_{p=1}^{P} preferencia_p(i) \quad (3)$$

En primer lugar se ha definido una preferencia que busca promocionar aquellas soluciones que presentan componentes parecidos a los que más interesan al experto. Si, dada una solución arquitectónica j, el arquitecto ha seleccionado el componente C^* como el mejor, es posible evaluar cómo un componente C de

otra solución i se asemeja a él en cuanto a las clases que lo conforman. Para ello se utiliza el coeficiente de similitud de Jaccard, el cual no sólo considera los elementos en común entre dos conjuntos finitos, sino también en cuántos difieren (ver Ecuación 4). Una vez obtenido ese valor, la Ecuación 5 permite calcular la preferencia *mejor componente* ($preferencia_{mejor\_comp}$) para el individuo i, que es determinada por el máximo grado de similitud entre el componente ideal, C_j^* y los N componentes de la solución i, C_i^n.

$$similitud(C, C^*) = \frac{\#clases_{C \cap C^*}}{\#clases_{C \cup C^*}} \quad (4)$$

$$preferencia_{mejor\_comp}(i) = max(similitud(C_i^n, C_j^*)) \;\; \forall \; n \; en \; [1, N] \quad (5)$$

Del mismo modo, se ha definido una segunda preferencia, *mejor interfaz*, que considera el grado de similitud entre las interfaces de los componentes comparando qué operaciones tienen en común. Aparte, existe la opción de no emitir ninguna preferencia. Nótese que, siguiendo esta filosofía, sería posible definir multitud de preferencias de diseño con las que apoyar la toma de decisiones, incluyendo también el establecimiento de puntos de referencia.

3.2. Mecanismo de interacción

La IEC proporciona diferentes mecanismos para seleccionar qué soluciones mostrar al experto, desde elegir únicamente al mejor individuo hasta trabajar con la población completa. Dadas las características del problema planteado aquí, una alternativa interesante es considerar un algoritmo de *clustering*, como kMeans++, pues permite seleccionar un porcentaje de soluciones de la población lo suficientemente representativas. Además, con el fin de que el arquitecto vea la progresión de la búsqueda, también se le muestra aquella solución con el mayor grado de satisfacción de las preferencias establecidas hasta el momento.

El modelo interactivo propuesto también permite al arquitecto congelar componentes de las arquitecturas que le son mostradas, marcar soluciones para forzar su eliminación de la población durante el reemplazo, archivar las mejores soluciones en una población externa y finalizar la búsqueda cuando lo desee.

3.3. Algoritmo evolutivo

Una vez definidos los aspectos propios del enfoque interactivo, esta sección explica cómo esos elementos son integrados en un algoritmo evolutivo multiobjetivo. En concreto, se ha diseñado un algoritmo estacionario, pues este esquema permite un proceso de evolución más lento que resultará más intuitivo para el arquitecto y, además, ha demostrado su eficacia con anterioridad [6].

Codificación de individuos. Cada solución arquitectónica es representada en forma de árbol, donde cada nivel establece cómo unos elementos son conformados

por otros más concretos. Se trata pues de una estructura flexible que puede ser fácilmente extendida añadiendo nuevos tipos de elementos del modelo, como pueden ser los atributos de las clases. Aquí, cada componente contiene una serie de clases en su interior, mientras que sus interfaces proveídas se especifican a partir de los métodos públicos de las clases que, si bien forman parte del componente, mantienen relaciones con clases alojadas en otros componentes. Los conectores permiten establecer relaciones entre las interfaces proveídas y requeridas.

Inicialización de la población. El algoritmo propuesto parte de una población aleatoria de soluciones arquitectónicas, de forma que las clases del diagrama de entrada se distribuyen en un número aleatorio de componentes para, a continuación, determinar las interfaces candidatas. A continuación, las medidas software son evaluadas, mientras que la componente de evaluación subjetiva se inicializa a 0, esto es, no será utilizada hasta que no se realice la primera interacción.

Operador de mutación. A lo largo de la evolución, nuevos individuos son generados mediante un operador de mutación que permite aplicar diferentes transformaciones arquitectónicas sobre los individuos, tales como añadir, eliminar, dividir o mezclar componentes, así como mover clases de un componente a otro. La selección de qué operador aplicar se realiza mediante una ruleta probabilística. Cabe destacar que todas estas operaciones han sido adaptadas para tener en cuenta la posible existencia de componentes "congelados", los cuales son excluidos de cualquier modificación. Con el fin de mantener la aleatoriedad del proceso, únicamente se permite fijar un componente en cada solución.

Mecanismos de selección y reemplazo. Al tratarse de un algoritmo estacionario sin operador de cruce, en cada iteración se seleccionan dos individuos para ser mutados, uno perteneciente a la población regular y otro al archivo externo. En ambos casos se aplica un torneo binario considerando la función global de *fitness*. Por otro lado, el mecanismo de reemplazo consiste en sustituir a los dos peores individuos de la población por los dos descendientes creados. Sin embargo, si alguna solución ha sido marcada por el arquitecto para su eliminación, tendrá prevalencia sobre cualquier otra solución de la población.

Manejo del archivo externo. Como es habitual en los algoritmos multiobjetivo, el modelo propuesto dispone de un archivo externo donde se van a almacenar las mejores soluciones encontradas. El arquitecto puede seleccionar qué soluciones quiere guardar en el archivo, mientras que el propio algoritmo controla la inclusión de soluciones no dominadas. Para ello se ha adaptado la estrategia propuesta en el algoritmo iTDEA, de forma que aquellas soluciones con el mayor grado de cumplimiento de preferencias tendrán asociados territorios de tamaño menor para así permitir la introducción de soluciones parecidas.

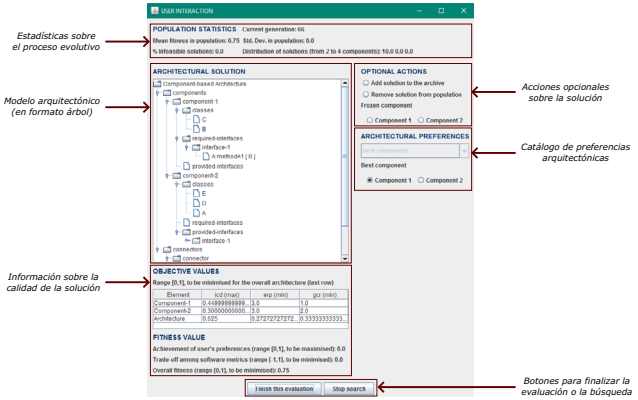

Figura 2. Interfaz gráfica de la herramienta prototipo

4. Marco experimental

El framework propuesto ha sido desarrollado en Java con la librería *JCLEC-MOEA* [10]. Otras librerías públicas se han utilizado para el preprocesamiento de datos[1] y para el algoritmo de *clustering*[2]. El framework dispone de un prototipo de herramienta gráfica desde la cual el arquitecto software puede interactuar (ver Figura 2). La herramienta permite la visualización del modelo arquitectónico en forma de árbol, mostrando a su vez los valores que dicha solución alcanza para las distintas medidas software. La herramienta dispone de un catálogo desplegable de preferencias, de forma que al seleccionar una concreta, por ejemplo *mejor componente*, la interfaz se actualiza para generar tantos botones de selección como componentes tenga la solución mostrada actualmente.

Para el estudio inicial del algoritmo se ha considerado un escenario de interacción real, donde tres usuarios han realizado sendas ejecuciones del algoritmo con el objetivo de descubrir la estructura de un sistema software real, *Datapro4j*. La configuración del algoritmo es la siguiente: 150 individuos en la población; un máximo de 12.000 generaciones como criterio de parada; soluciones de entre 2 y 6 componentes; 4 interacciones, mostrando 3 soluciones en cada una de ellas. Las interacciones son automáticamente espaciadas a lo largo de la evolución [7], mientras que $w_{obj} = w_{sub} = 0{,}5$. El resto de parámetros relativos al dominio del problema han sido fijados de acuerdo al trabajo anterior [6], escogiendo tres

[1] `http://www.uco.es/grupos/kdis/datapro4j/`
[2] `http://commons.apache.org/proper/commons-math/`

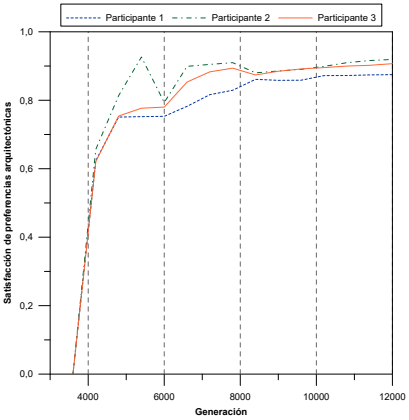

Figura 3. Evolución de la componente subjetiva de la función de *fitness*

medidas software (ICD, ERP y GCR) relacionadas con criterios de mantenibilidad. Por último, los parámetros requeridos por el mecanismo de actualización del archivo han sido fijados de acuerdo a las recomendaciones dadas en [7].

Durante la ejecución, la herramienta almacena varios ficheros *log* donde se registran las acciones realizadas por el usuario y la propia evolución del algoritmo. A partir de dicha información se puede analizar el funcionamiento del algoritmo y la influencia de las decisiones tomadas por el usuario. A modo de ejemplo, la Figura 3 muestra el grado de cumplimento medio de las preferencias en la población a lo largo de las generaciones. Cabe señalar que el algoritmo ejecuta un tercio del número máximo de generaciones antes de la primera interacción, con el fin de mostrar soluciones relativamente buenas. Como puede observarse, tras cada interacción (líneas discontinuas), el algoritmo genera soluciones que satisfacen las nuevas preferencias introducidas. Este incremento es más visible en las primeras interacciones que en las últimas, debido a que al final del proceso, los usuarios optaron por emitir menos preferencias. Respecto a las acciones realizadas por los usuarios, se pueden extraer las siguientes conclusiones:

- La preferencia *mejor componente* es más utilizada que la de *mejor interfaz*, posiblemente porque resulta más sencillo analizar la estructura de los componentes que las interacciones entre ellos.
- Los participantes aplicaron con frecuencia las acciones opcionales, especialmente la de añadir soluciones al archivo y la de congelar partes del genotipo.

5. Conclusiones

En este trabajo se ha presentado un framework basado en IEC para resolver un problema de optimización arquitectónica desde una perspectiva multiobjetivo. El enfoque propuesto combina criterios de evaluación cuantitativos (medidas software) y cualitativos (preferencias arquitectónicas), ya que éstos últimos se acercan más a la forma en la que los arquitectos software toman decisiones. La herramienta gráfica desarrollada da soporte a la interacción con el usuario, de forma que éste puede visualizar las características de la solución y tomar diferentes decisiones en base a criterios subjetivos.

La experimentación preliminar realizada muestra como el algoritmo, conforme avanza la evolución, es capaz de encontrar soluciones que cumplen la mayoría de preferencias establecidas sin por ello perjudicar el rendimiento evolutivo. En el futuro se pretende incorporar preferencias que permitan indicar qué aspectos del diseño deben ser mejorados, así como realizar un estudio con más participantes.

Referencias

1. H. Takagi, "Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.
2. D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, and N. Gaud, "A Review and Taxonomy of Interactive Optimization Methods in Operations Research," *ACM Transactions on Interactive Intelligent Systems*, vol. 5, no. 3, pp. 17:1–43, 2015.
3. M. Harman, S. A. Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications," *ACM Computing Surveys*, vol. 45, no. 1, pp. 1–64, 2012.
4. C. L. Simons and I. C. Parmee, "Elegant Object-Oriented Software Design via Interactive, Evolutionary Computation," *IEEE Transactions on Systems, Man and Cybernetics. Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1797–1805, 2012.
5. M. R. N. Shackelford, "Implementation issues for an interactive evolutionary computation system," *Proceedings of the Companion Publication 2007 Genetic and Evolutionary Computation Conference*, pp. 2933–2936, 2007.
6. A. Ramírez, J. R. Romero, and S. Ventura, "A comparative study of many-objective evolutionary algorithms for the discovery of software architectures," *Empirical Software Engineering*, 2015.
7. M. Koksalan and I. Karahan, "An Interactive Territory Defining Evolutionary Algorithm: iTDEA," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 702–722, 2010.
8. S. Vathsavayi, Hadaytullah, and K. Koskimies, "Interleaving human and search-based software architecture design," *Proceedings of the Estonian Academy of Sciences*, vol. 62, no. 1, pp. 16–26, 2013.
9. R. Balling and S. Wilson, "The maximin fitness function for multi-objective evolutionary computation: application to city planning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1079–1084, 2001.
10. A. Ramírez, J. R. Romero, and S. Ventura, "An Extensible JCLEC-based Solution for the Implementation of Multi-Objective Evolutionary Algorithms," in *Proceedings of the Companion Publication 2015 Genetic and Evolutionary Computation Conference*, pp. 1085–1092, 2015.