Yazılım Mimarisi Geri Kazanımı : Hiyerarşik Kümeleme Yöntemi

Ozan ZORLU

Hava Harp Okulu, İstanbul, Türkiye
oznzrl@gmail.com

Özet. Yazılım geliştirme süreçlerinin iyi işletilmemesi sonucu çeşitli problemlerle karşılaşılmaktadır. Bu problemlerin en önemlilerinden bir tanesi yazılım mimarisinin oluşturulmamış olmasıdır. Yazılım mimarisi olmayan bir sistemde analiz, geliştirme ve test işlemleri zorlaşacak ve belki imkansız hale gelecektir. Bu nedenle; kaynak kodundan yazılım mimarisinin çıkarsanmasına yoğunlaşılmış ve çeşitli yöntemler ile denenmiştir. Bu çalışma ile; eksikliği hissedilen yazılım mimari geri kazanımı (software architecture recovery) alanındaki kaynak eksikliği giderilmeye çalışılırken, mevcut çalışmaların incelenerek, anlaşılabilir bir şekilde ifade edilmesi ve bundan dolayı araştırmalara kaynak sunulması amaçlanmıştır. Bu kapsamda hiyerarşik kümeleme ile yazılım mimarisinin geri kazanımı çalışmaları incelenmiş ve bu araştırmaların temelini oluşturan; kümeleme ölçütleri, algoritmaları ve yazılım alanındaki uygulamaları anlatılmıştır.

Anahtar Kelimeler: Yazılım mimarisi, Tersine mühendislik, Mimari geri kazanımı, Hiyerarşik kümeleme.

Abstract. As a result of not running appropriately the software development process, many problems occurs. One of important these problems is not forming software architecture. Analysis, development and testing will become harder, maybe impossible with a system without software architecture.For this reason, focused on software architecture recovery from source code and tested various methods. While lack of source about software architecture recovery domain is trying to be satisfied, also, explanation of past studies with a comprehensible way and for this reason representing a source material for future research is aimed.With this perspective; hierarchical clustering for software architecture recovery examined and fundamental principals, clustering measures, algorithms and studies on software domain described.

Anahtar Kelimeler: Software architecture, Reverse engineeering, Architecture recovery, Hierarchical clustering.

1 GİRİŞ

Yazılım sisteminin mimarisi nesneye dayalı sistemlerde sınıflar ve ilişkileri ifade eder. Bu yapıların kullanılarak yazılım mimarisinin yeniden oluşturulabiliyor ol-

ması çoğu durumda hayati önem arz etmektedir. Günümüzde bazı yazılımların, kaynak kodundan başka hiç bir dokumanı bulunmamaktadır. Bu nedenledir ki yazılımda değişiklik yapılması, geliştirmeler için ihtiyaçların analiz edilmesi veya hata giderimleri için gereksinim duyulan ön bilginin olmayışı nedeniyle zaman ve para kayıpları büyük miktarlarda olabilmektedir. Bu problemleri aşabilmek için yazılım kaynak kodundan yazılım mimarisinin çıkarılması ihtiyacı ortaya çıkmış ve otonom olarak çalışan çeşitli yöntemler geliştirilmiştir [1], [2], [3], [4].

Çeşitli alanlarda mimari geri kazanımı (architecture recovery) için bir çok teknik geliştirilmiş ve uygulanmıştır. Bu yöntemlerden bazıları, yazılım mimarisinde yapı analizi, bağıntı ve ilişki çıkarımı ile kümeleme için kullanılmıştır [5],[6]. Pollet ve diğerlerinin sunmuş oldukları mimari geri kazanımı yaklaşımında, yazılım mimarisi tekrar inşasında (reconstruction) kümeleme, "quasi-auto- matic" (otomatik gibi) olarak tanımlanmıştır [7].

Bu alandaki tüm çalışmalar göstermiştir ki; mimari geri kazanımı yaklaşımlarının temelinde kümeleme bulunmaktadır. Kümelemenin bu alanda kullanılması ve ilgi çekiyor oluşu, Jain ve diğerlerinin veri kümeleme incelemesiyle [8] büyük oranda artış göstermiştir [9]. Ayrıca bu çalışmada kümeleme yaklaşımlarında kullanılan algoritmalar, bölümlemeli ve hiyerarşik olmak üzere iki sınıf olarak tanımlanmıştır. Bölümlemeli algoritmalar genel olarak, başlangıçta giriş bölümlemelerinden oluşan belirli sayıdaki kümelerden oluşur. Her bir adımda, başlangıçta belirlenen küme sayısı sabit kalacak şekilde, bölümlemeler belirlenerek kriterlere göre düzenlenirler. Bu algoritmanın dezavantajı kümeleme yapılacak olan sistemde kaç adet küme olduğunun bilinmemesi durumunda ortaya çıkmaktadır. Diğer taraftan, yazılım mimarisi düşünüldüğünde ise genellikle oluşturulacak olan küme sayısı bilinmemektedir. Bunun yanında, bölümlemeler içerisindeki n nesne k küme içerisinde aranmaktadır ve genellikle çok fazla hesaplama yükü getirmektedir. Bu nedenlerden dolayı araştırmacılar bulgusal (heuristic) tabanlı yaklaşımları benimsemişlerdir [10], [11], [12].

Yazılım sisteminin küme sayı bilgisi başlangıçta edinilebiliyorsa, yazılım mimari geri kazanımı için bölümlemeli kümeleme kullanılabilir. Ancak bu durumda bile yazılımların genel yapısı itibari ile hiyerarşik yapıda tasarlanmaları nedeniyle, düz yapıda olan bölümlemeli kümeleme çoğu zaman uygun olmaz [5], [13]. Diğer bir kümeleme yöntemi olan hiyerarşik kümeleme de ise seviyeli yapı desteklenir. Hiyerarşik kümelemede önceki iterasyonlar mimariyi detaylı olarak ortaya çıkarırken, sonraki iterasyonlarla yüksek seviye görünüm elde edilir ve başlangıçta küme sayı bilgisine ihtiyaç duyulmaz.

Çeşitli alanlarda kullanılan kümeleme yöntemlerinin temel yapısı, ancak kümeleme değerleri ve algoritmalarının anlaşılması ile yazılım mimarisinde geri kazanım için uyarlanabilir. Bu nedenle, hiyerarşik kümeleme yöntemiyle yazılım mimarisi geri kazanımında, bu çalışmada da olduğu gibi diğer çalışmalarda da, kullanılan temel kaynaklardan biri Maqbool ve Babri tarafından yayınlanmıştır [9]. Yaptıkları çalışmada kısaca:

1. Çeşitli benzerlik ve mesafe ölçütleri yazılımlar için, analiz edilerek benzerlik/mesafe grupları çıkarılmış, ayrıca aynı grup üyeleriyle yapılan incelemelerin aynı sonuçları verdiği,

2. Yayınlanmış olan Weighted Combined Algorithm (WCA) [14] ve LIMBO [15] algoritmaları incelenmiş ve adım yaklaşım benzerlikleri açıklanmış, ayrıca bu yaklaşımların uyarlanmasıyla daha az sayıda keyfi karar verildiği ve daha iyi sonuçlar elde edildiği,

3. Bilinen hiyerarşik kümeleme algoritmaları analiz edilerek ve bağıntılarının güçlü ve zayıf yanları değerlendirilerek, kümeleme algoritmalarının performansının sadece kendi karakteristiğine değil aynı zamanda uygulandığı yazılıma göre de değişiklik gösterdiği sunulmuştur [9].

Geri kazanım işlemi için geliştirilmiş olan Dali [16], PBS [17], Imagix4D [18] and Bauhaus [19] gibi yardımcı araçlar bulunmaktadır. R. Naseem ve diğerleri tarafından yayınlanan çalışmada ise bu araçların mimari geri kazanımındaki tüm adımların gerçeklenmesinde tam anlamıyla kullanılamaması nedeniyle DRT isimli özelleştirilmiş araç geliştirilmiştir [20]. Bunun yanında tersine mühendislik için yaklaşımlar açıklanmıştır.

Otomatik olarak yazılım mimarisinin kazanılması alanında birçok çalışma sunulmuştur [1], [2], [3], [4]. Bu çalışmalardan T. Lutellier ve diğerlerinin sunduğu çalışmaya göre önceki çalışmaların geliştirildiği ve 10 milyon seviyesindeki satırdan oluşan kodlarda sunulan alt modül tabanlı teknik kullanarak çalışabildiği, diğer yöntemlerle bunun yapılamadığı sunulmuştur [4].

Bu çalışmada ise amaç; hiyerarşik kümeleme yapısının, açık bir şekilde incelenmesi ve geçmiş çalışmaların açıklanması ile yazılım mimarisi geri kazanımı için uygulanabilirliğinin araştırılarak, bu alanda eksik olan kaynak doküman problemine çözüm bulmaktır. Bu nedenle; konu kapsamına giren çalışmalar incelenmiş, özellikle Maqbool ve Babri tarafından yapılan çalışma ana referans olarak alınmış ve ilgili bilgiler kullanılmış [9], temel oluşturacak bilgiler anlaşılır bir şekilde ifade edilmeye çalışılmış ve araştırmalara kaynak olması hedeflenmiştir. Bu kapsamda; Bölüm' 1 de yazılım mimarisi ve kümeleme alanlarındaki mevcut problemler ile yapılan çalışmalara değinilmiş, Bölüm 2' de kümeleme benzerlik ölçüleri ve karşılaştırmaları kısaca anlatılmış, Bölüm 3'te hiyerarşik kümeleme adımları yazılım mimarisi boyutuyla anlatılmış, kullanılan algoritma ve benzerlik ölçülerinin karşılaştırmalı analiz çalışmaları incelenmiş ve Bölüm 4'te ise hiyerarşik kümeleme algoritmaları anlatılarak karşılaştırılmıştır.

2 Kümeleme

Kümeleme işleminde; gruplar içerisindeki madde veya varlıklar birbirleriyle benzerlik, diğer gruplar ile farklılık gösterecek şekilde biçimlendirilir. Yapılan bu ayrımda benzerlik veya farklılıklar madde/varlıkların karakteristik veya özelliklerine göre olur. Bunun yanında, benzerliklerin belirlenmesi için formal ve nonformal özellikler kullanılmış, bazı araştırmacılar tarafından ise beraber kullanılmıştır [15], [21]. Formal özeliklere; global değişkenleri, fonksiyonları, nonformal özelliklere ise yorum veya açıklama satırları örnek olarak verilebilir.

Kümelemede özelliklerin ifade şekli genellikle binary olarak yapılır ve '1-0' şeklinde ifade edilir. Benzerliklerin ölçülmesi için kullanılan binary tanımlamada karşılaştırılan iki farklı madde/varlık aynı özelliklerinin durumuna göre incelenir.

Eğer ikisinde de özellik '1' olarak ifade ediliyorsa benzerlik olma olasılığı yüksek demektir. Diğer bir taraftan ise iki özelliğin de '0' olması '1' olmasına göre daha düşük olasılıkla benzerlik olabileceğini gösterir. Bu nedenle benzerlikte özellik değerlerinin '0' olması değil, '1' olması kümeleme için daha yüksek öneme sahiptir.

Kümeleme işleminde öncelikli olarak varlıklar ve özellikler tanımlanarak her bir varlık için özellik vektörü oluşturulur. Özellik vektörleri genellikle '1' ve '0'lardan oluşur. Devamında benzerlik ölçütü belirlenerek her bir varlığın diğerleriyle olan benzerlikleri kare matris şeklinde ifade edilir. Benzerlik matrisinin oluşturulması sonucunda ise kümeleme algoritması kullanılarak uygun gruplandırma işlemi yapılır. Önemli olan grup içerisindeki benzerliklerin yüksek olması ve gruplar arası benzerliklerin daha düşük olmasıdır.

2.1 Benzerlik Ölçüleri

Kümeleme işleminde kullanılacak olan benzerlik değerleri olarak, mesafe ölçüleri (distance measures) ve bağıntı katsayıları (correlation coefficient) kullanılır. Bir çok benzerlik ölçüsü vardır ve bunları mesafe katsayıları, ilişki katsayıları, bağıntı katsayıları, olasılık katsayıları şeklinde gruplandırabiliriz. Bilinen mesafe ve benzerlik ölçüleri Tablo 1' de verilmiştir [9].

Tablo 1. Mesafe ve Benzerlik Ölçüleri

Mesafe Ölçüsü	**Formül**
Minkowski	$(\sum \lvert x_i - y_i \rvert^r)^{1/r}$
Canberra	$\sum \lvert x_i - y_i \rvert / \left(x_\lvert i \rvert + \lvert y_i \rvert \right)$
Bray-Curtis	$\frac{\sum \lvert x_i - y_i \rvert}{\sum \left(x_\lvert i \rvert + \lvert y_i \rvert \right)}$
Chord	$\sqrt{2\left(1 - \frac{\sum \lvert x_i y_i \rvert}{\sum x_i^2 \sum y_i^2}\right)}$
Hellinger	$\sqrt{\sum \left[\sqrt{\frac{x_i}{x_+}} - \sqrt{\frac{y_i}{y_+}} \right]^2}$

Benzerlik Ölçüsü	**Formül**
Simple Matching	$\frac{(a+d)}{(a+b+c+d)}$
Jaccard	$\frac{a}{(a+b+c)}$
Sorenson-Dice	$\frac{2a}{(2a+b+c+d)}$
Rogers ve Tanimoto	$\frac{(a+d)}{(a+2(b+c)+d)}$
Sokal ve Sneath	$\frac{a}{(a+2(b+c))}$

2.2 Benzerlik Ölçülerinin Karşılaştırılması

Jaccard ve Sorenson-Dize ölçüleri karşılaştırıldığında aynı kümeleri buldukları görülmüştür [21]. Ancak; Sorenson-Dice ölçülerinde ekstradan ağırlık verme işlemi olması nedeniyle, daha uygun sonuçlar vermesi beklenirdi [9]. Jaccard, Sorenson-Dice ve Sokal-Sneath değerleri ile yaptıkları çalışma sonucu, aynı kümeleri bulmalarının nedeninin monotonik bağıntılar olduğunu gördüler [22]. Monotik bağıntıda varlıklar $(E_1 - E_4)$ arasındaki benzerlik ilişkisi $benzerlik(E_1, E_2) > benzerlik(E_3, E_4)$ ve bu sonuca $s1$ benzerlik değeri ile hesaplama sonucu ulaşıldıysa E_3 ve E_4 değerleri $s1$ değeriyle monotik olarak ilişkili ise diğer metrikleri içinde her zaman $benzerlik(E_1, E_2) > benzerlik(E_3, E_4)$ olacaktır [9].

Bağıntı katsayısı yaklaşımında, olmayan özellik değerlerinin hesaplamalarda kullanılmasına rağmen, yazılım kümeleme için iyi sonuçlar elde etmesi beklenmeyen bir durumdur ve nedeni olarak ise benzerlikleri ifade eden '1' değerlerinin '0' olan değerlere oranla yüksek seviyede öneme sahip olmasıdır ve formülsel açıklaması da çeşitli çalışmalarla yapılmıştır [9].

Ayrıca bu çalışmada görülmüştür ki farklı ölçülerle alınan sonuçlar yazılım karakteristiklerine ve özellik ifade şekline bağımlıdır [9] .

Tablo 1'de yer alan mesafe ölçülerinde x ve y Euclidian uzayındaki noktaları temsil etmektedir. Tablo 1'deki benzerlik ölçülerinde; karşılaştırılan varlıkların ikisinde de '1' olan özellik sayısı a, ikisinde de '0' olan özellik sayısı d, birisinde '1' iken diğerinde '0' olan özelliklerin sayısı ise sırasıyla b ve c ile ifade edilmektedir. Yazılım mimarisinde özelliklerin asimetrik olması nedeniyle mesafe değerlerinden d içermeyenlerin, içerenlere oranla daha iyi sonuçlar verdiği gözlemlenmiştir [21], [23].

Hiyerarşik kümelemede birden fazla varlık birbirine benzer çıkabilir ve keyfi seçim yapılması söz konusudur. Bu durumdan sakınmanın daha iyi sonuçlar verdiği çalışmalar ile gösterilmiştir [14].

Temel hiyerarşik kümeleme algoritmaları Tablo 2'te verilmiştir [9]. Bu algoritmalar ile; belirlenmiş olan yeni küme ve diğer varlıkların benzerlikleri, belirtilen hesaplamalar ile yapılır. E_i, E_m ve E_o varlıkları E_{mo} ise varlıkların birleştirilerek küme oluşturulmasını ifade eder. Tablo 2'de verilen algoritmalara göre CLA yazılım kümelemede diğerlerine göre daha başarılıdır. Bunun yanında; SLA daha başarısız ve WLA ile ULA ise bu iki algoritmanın arasında başarıya sahiptir [9].

Tablo 2. Hiyerarşik Kümeleme Algoritmaları

Algoritma	**Küme Benzerliği**
Single Linkage (SLA)	$sim\left(E_i, E_{mo}\right) = Max\left(sim\left(E_i, E_m\right), sim\left(E_i, E_o\right)\right)$
Complete Linkage (CLA)	$sim\left(E_i, E_{mo}\right) = Min\left(sim\left(E_i, E_m\right), sim\left(E_i, E_o\right)\right)$
Weighted Average Linkage (WLA)	$sim\left(E_i, E_{mo}\right) = 1/2\left(sim\left(E_i, E_m\right)\right) + 1/2 sim\left(E_i, E_o\right)$
Unweighted Average Linkage (ULA)	$sim\left(E_i, E_{mo}\right) = \left(sim\left(E_i, E_m\right) + size\left(E_m\right) + sim\left(E_i, E_o\right) * size\left(E_o\right) t\right) / \left(size\left(E_m\right) + size\left(E_o\right)\right)$

3 Yazılım Mimarisinde Kümeleme Adımları

Yazılım mimarisi geri kazanımına uyarlanmış olan kümeleme araştırmalarının temelini oluşturan adımlara, bu bölümde değinilecektir. Ancak ve ancak bu adımların incelenip, iyi analiz edilmesi sonucu başarıya ulaşılabileceği vurgulanması gereken bir gerçektir.

3.1 Özellik Şeçimi

Kümeleme işleminin temelinde benzerliklerin çıkarılması ve çeşitli kümeleme algoritmaları yardımıyla gruplamanın yapılması yer almaktadır. Benzerliklerin çıkarılabilmesi için ise her bir varlığa ait özelliklerin tespit edilmesi, seçilmesi, belirlenmesi gerekmektedir. Şüphesiz ki, varlıkların özelliklerinin çıkarılması işlemi ne kadar başarılı olursa, o denli başarılı sonuçlar elde edilebilir.

Özellik çıkarımında formal ve nonformal özelliklerin ayrık veya birlikte kullanıldığı çeşitli araştırmalar bulunmaktadır [15], [23]. Formal özelliklerde; varlık tarafından çağırılan fonksiyonlar, global değişkenler ve kullanıcı tanımlı değişkenler yer alırken nonformal özellikler içerisinde; yorumlar, tanımlar, fonksiyon-dosya-geliştirici isimleri, son düzenleme zamanı gibi bilgiler yer alır. Performansın artırımı için özelliklerin ağırlıklandırılması yöntemini kullanarak, daha önemli ve/veya kümelemede daha yüksek öneme sahip olan özelliklerin, ön plana çıkarılmasını amaçlayan bir çok çalışma da bulunmaktadır [6], [15].

3.2 Yazılım Genelinde Kullanılan Modüllerin Tespiti

Yazılım sistemlerinde, bazı fonksiyonlar diğer fonksiyonlar tarafından sıklıkla çağırılır ve *omnipresant modules* (OM) olarak adlandırılırlar [24]. Bu fonksiyonların karakteristiklerinin farklı olması nedeniyle kümeleme işlemi yapılmadan önce sistemden çıkarılmalı ve karakteristiklerine uygun ayrı bir küme olarak veya dağıtık bir şekilde diğer kümeler içerisinde yer almaları önerilmektedir [5]. Öcelikli olarak bu fonksiyonların tespit edilmesi ihtiyacı karşılanmalıdır. Bu amaçla geiştirilen yöntemlerden bir tanesi; belirli bir eşik değerinin tespit edilmesi ve fonksiyonların çağırılma sayılarının bu eşik değerinin üzerine çıkması soucu OM olarak değerlendirilmesidir. Bunun yanısıra; Wen ve Tzerpos bu fonksiyonların tespitini, fonksiyonların, belirlenen kümelere olan bağlılıklarına göre değerlendirilmesi incelemesi yapmışlardır. [25].

3.3 Küme İsimlendirmesi

Kümeleme işlemi yapılarak gruplandırılan yazılımın anlaşılır olabilmesi için grupların etiketlenmesi gerekir. Bu nedenle; etiketleme işlemi, özelliklerin özetlerinden yararlanarak manuel olarak [6], rota tabanlı tanıma ile benzer alt sistemlerin değerlendirilmesiyle otomatik olarak [26] veya özel kelimelere dayalı olarak [27] dinamik olarak yapılabilmektedir.

3.4 Ölçülerin Değerlendirilmesi

Kümeleme işlemi tamamlandıktan sonra tahmin edilen grupların değerlendirilmesi, performansın ölçülmesi gerekmektedir. Bu amaçla; dışarıdan uzmanların müdahalesi ile belirlenen grupların, kümeleme sonucu belirlenen gruplarla karşılaştırılarak ölçüldüğü dış değerlendirme yöntemi, içsel verilerin kullanılarak değişik özelliklerle kümeleme işleminin yapılmasıyla iç değerlendirme yöntemi ve farklı kümeleme algoritmalarının uygulanması sonucu elde edilen verilerin karşılaştırılarak veya aynı algoritmayla farklı özelliklerin değerlendirilerek kullanıldığı göreceli değerlendirme yöntemleri kullanılabilmektedir [9].

4 Hiyerarşik Kümeleme Algoritmaları

Kümeleme algoritmaları, yazılımlar için özelliklerin kullanılmasıyla gruplama işlemi için kullanılırlar. Yazılım mimarisinin hiyerarşik yapıda olması nedeniyle de kullanılacak olan kümeleme algoritmalarının hiyerarşik kümeleme algoritmaları olmaları gereklidir. Bu nedenle temel olarak CA [28], WCA [14] ve LIMBO [15] geliştirilmiş, yazılım mimari geri kazanımı için kullanılmışlardır. Bu algoritmalar benzerlik matrisi kullanarak varlık veya küme benzerliklerini ölçerler.

Ayrıca; benzerlik yaklaşımları, başlangıçta yeni oluşturulan kümenin özellik vektörü çıkarılmasıyla, mevcut varlıklar ile bu kümenin benzerliğinin yeniden hesaplanmasına dayanır. Kümeleme algoritmalarındaki önemli değerlerden bir tanesi aynı değerlere sahip iki varlıktan hangisinin kümeye dahil edileceği (keyfi seçim), diğeri ise kullanılacak özellik sayısının belirlenmesidir. Yapılan çalışmalarda bu keyfi seçimin azaltılmasıyla ve kullanılan özellik sayısının belirli bir noktaya kadar artırılmasıyla daha iyi sonuçlar elde edildiği, daha da artırılması sonucu daha kötü sonuçlar elde edildiği görülmüştür [9].

4.1 Combined ve Weighted Combined Algoritmaları

CA çalışması Şekil 1' de [9] anlatıldığı gibi olup keyfi seçim sayısının düşürülmesine karşın yeni özellik vektörü çıkarıldığında varlığın eriştiği özellik bilgisi kaybolmaktadır. WCA' da ise bu bilgiler saklanarak CA' daki dezavantaj giderilmektedir. WCA' nın çalışmasının CA'dan farkı Şekil 2' de [9] gösterilmiştir.

Şekil 1. Hiyerarşik kümeleme için Combined Algorithm (CA)

1. Her bir varlık E_i bileşenleri $f_{i1}, f_{i2}...f_{in}$ olan f_i özellik vektörleri çıkarılarak tanımlanır.
2. Her bir varlık arasındaki benzerlik hesaplanır.
3. Aşağıdaki adımlar tekrar edilir;
   (a) En çok benzeyenleri kümelenir,
   (b) Yeni oluşturulan kümeyi oluşturan varlıkların özellik vektörleriyle binary OR yapılarak yeni özellik vektörü oluşturulur. E_i ve E_j kümelenecek varlıklar, f_i ve f_j ise bu varlıkların özellik vektörleri, f_{ij} de yeni özellik vektörü olmak kaydıyla $f_{ij} = f_i \cup f_j$ şeklinde oluşturulur.
   (c) Yeni üretilen küme varlık olarak değerlendirilerek diğer varlıklarala olan bezerliği hesaplanır,

   Tekrarla
   Gerekli sayıda küme oluşturulana veya tek küme kalana kadar

4.2 LIMBO

WCA'ya göre çeşitli temel farkları vardır. Bunlardan birincisi, doğruluk kaybı indirgenirken hesaplama ihtiyaçlarını en aza indirmek için Summary Artifacts

Şekil 2. Weighted Combined Algorithm WCA ile Combined Algorithm (CA) Farkı

3b. Yeni oluşturulan kümeyi oluşturan varlıkların özellik vektörleriyle binary OR yapılarak yeni özellik vektörü oluşturulur. E_i ve E_j kümelenecek varlıklar, f_i ve f_j ise bu varlıkların özellik vektörleri, f_{ij} yeni oluşturulacak olan özellik vektörü, E_i nin varlık sayısı n_i ve E_j nin varlık sayısı n_j olarak tanımlanırsa;
$f_{ij} = (f_i + f_j)/(n_i + n_j)$ şeklinde oluşturulur.

(SA) denen metod kullanılmasıdır. İkincisi, birleştirilmiş küme için yeni özellik vektörü oluşturulması ve son olarak ise bilgi kaybı ölçüsü (Information Loss) ile varlıklar arası benzerliklerin çıkarılmasıdır [29].

Kümeleme işlemi sırasındaki bilgi kaybını en aza indirgemeyi amaçlayan AIB algoritmasına dayalı olarak LIMBO algoritması geliştirilmiş ve temel amaç olarak, aktarılabilen maksimum seviyede bilginin bir sonraki işlem için aktarılmasını temel alınmıştır [15]. Algoritmanın WCA ile olan farkı Şekil 3'te [9] gösterilmiştir.

Şekil 3. Hiyerarşik kümeleme için LIMBO.

1. Her bir varlık E_i bileşenleri $f_{i1}, f_{i2}...f_{in}$ olan f_i özellik vektörleri çıkarılarak tanımlanır.
2. Her bir varlık arasındaki benzerlik hesaplanır.
3. Aşağıdaki adımlar tekrar edilir;
   (a) En çok benzeyenleri kümelenir,
   (b) Yeni oluşturulan kümeyi oluşturan varlıkların özellik vektörleriyle binary OR yapılarak yeni özellik vektörü oluşturulur. E_i ve E_j kümelenecek varlıklar, f_i ve f_j ise bu varlıkların özellik vektörleri, f_{ij} de yeni özellik vektörü olmak kaydıyla $f_{ij} = f_i \cup f_j$ şeklinde oluşturulur.
   (c) Yeni üretilen küme varlık olarak değerlendirilerek diğer varlıklarala olan bezerliği hesaplanır,

   Tekrarla
   Gerekli sayıda küme oluşturulana veya tek küme kalana kadar

Normalized WCA ve LIMBO sınıfı algoritmalar ile geleneksel kümeleme algoritmaları karşılaştırıldığında WCA ve LIMBO'nun kümeleme işleminde daha az özellik bilgisi kaybettiği, ayrıca daha az keyfi seçimler yaptığı, bundan dolayı da daha iyi sonuçlar alındığı testler sonucu görülmüştür [9].

4.3 Kümeleme Algoritmaları ve Beraberinde Kullanılan Benzerlik Değerlerinin Analizi

Araştırmacılar tarafından, yazılım kümeleme kalitesinin artırılması amacıyla; algoritmalar, benzerlik değerleri ve özellikler karşılaştırılarak analiz edilmiştir. Bu kapsamda ilk hiyerarşik kümleme yönteminin yazılım sistemlerine uyarlanması, 1985 yılında Hutchens ve Basili tarafından yapılmış ve Single Linkage Algorithm(SLA) sunulmuştur [5]. Çalışmalarına göre Recomputed binding, Expected binding, ve Weighted binding algoritmalarına göre küçük test sistemlerinde SLA daha başarılı sonuçlar verirken Recomputed binding algoritması büyük sistemlerde daha iyi sonuç vermektedir [9].

Anquetil and Lethbridge tarafından, formal-nonformal özellikler beraber kullanılmış [23] ve Jaccard-Sorenson-Dice benzerlik değerlerinin en iyi sonuçları verdiği belirtilmiştir. Bunun yanında Complete linkage algortimasının (CLA), SLA'ya göre daha iyi sonuçlar elde ettiği belirtilmiştir. Çeşitli çalışmalar ile CLA algoritmasının daha iyi sonuçlar verdiği desteklenmiştir [21].

Saeed ve diğerleri hiyerarşik kümeleme için Combined Algorithm (CA) sunmuştur [28]. Maqbool and Babri kümeleme kalitesinin, varlıkların küme içerisinde eriştikleri özelliklerle değerlendirilerek daha iyilenebileceğini, Weighted combined algorithm (WCA) sunarak göstermiştir [14]. İncelenen CA, WCA, SLA, CLA, Weighted linkage algorithm (WLA) ve Unweighted linkage algorithm (ULA) içerisinde WCA ile elde edilen sonuçların diğerlerine göre daha iyi olduğu sunulmuştur. Ayrıca WCA'nın CLA'ya göre otomatik isimlendirme işleminde daha dengeli ve anlaşılabilir mimari yapıda olduğu, sunulan isimlendirme şemalarının kullanılmasıyla yapılan analizde tespit edilmiştir [30].

Tzerpos çalışmasında Mojo metric sunarak, aynı sistemdeki bölümlemeler arasındaki mesayi ölçmeyi amaçlamış ve kararlılıklarını incelemiştir [26]. Yaptığı karşılaştırmada ACDC, SLA ve CLA arasında, en kararlı olanı SLA iken en kararsız olanı ACDC olarak tespit etmiştir.

Kümeleme işlemi esnasında bilgi kaybının en aza indirgenmesine dayalı LIMBO algoritması sunulmuş ve bu algoritma ile yapısal olmayan özelliklerin, kullanışlı olup olmadıklarının ağırlaklandırma ile ölçülmesi yaklaşımı incelenmiştir [15]. Linux, Mozilla ve Tobey sistemleri üzerinde LIMBO, SLA, CLA, WLA, ULA, ACDC, NAHC, SAHC [31] algoritmaları karşılaştırmasına göre LIMBO diğer algoritmalarla aynı veya daha iyi sonuçları elde etmiştir.

Yapılan bir başka çalışmada ise Bunch, ACDC, SLA ve CLA karşılıştırılmış olup, uzman değerlendirmesine göre Bunch ve CLA sonuçlarının daha uygun olmasına rağmen ACDC ve SLA'nın daha kararlı olduğu gözlemlenmiştir [32].

5 Sonuç

Bu çalışma ile; yazılım mimarisi geri kazanımı için hiyerarşik kümeleme yönteminin nasıl, hangi parametrelerle kullanılması gerektiği ve algoritma, ölçülerin incelenmesi-karşılaştırılması ile bu alana genel bir bakış yapılmıştır. Bu nedenle alandaki önemli çalışmalardan bazıları temel referans kaynağı olarak ele alınmıştır [9].

Böylece yazılım mimarisi geri kazanımı alanında eksik olan kaynak eksikliği doldurulmaya çalışılmıştır.

Öncelikli olarak kümelemenin anlaşılması için algoritmalar ve ölçüler sunulmuştur. Ayrıca incelenen karşılaştırma çalışmaları da sunularak yazılım alanına en uygun algoritma ve ölçülerin tespit edilmesi amaçlanmıştır. Aynı zamanda kullanılan algoritma ve ölçülerin uygulandıkları yazılım karakteristiklerine göre performans farkı gösterdiği de göz ardı edilmemelidir. Gelecek çalışmalarda yazılım mimarisi geri kazanımı alanında, kümeleme ile daha iyi sonuçlar elde edilmesi amacıyla, büyük ölçekli yazılımlar çeşitli algoritmalar ve mesafe ölçüleriyle analiz edilerek sunulacaktır.

References

1. C. Sun, J. Zhou, J. Cao, M. Jin, C. Liu, Y. Shen, "ReArchJBs: a tool for automated software architecture recovery of JavaBeans-based applications," Proc. Software Engineering Conference, pp. 270-280, 2005.
2. H. Sajnani, "Automatic software architecture recovery: A machine learning approach" Proc. 20th International Conference on Program Comprehension (ICPC), pp. 265-268, 2012.
3. H.M. Fahmy, R.C. Holt, and J.R. Cordy, "Wins and Losses of Algebraic Transformations of Software Architectures," Proc. 16th Ann. Int'l Conf. Automated Software Eng., pp. 51-62, 2001.
4. T. Lutellier, D. Chollak, J. Garciaz, L. Tan, D. Rayside, N. Medvidovicy, and R. Kroeger, "Comparing Software Architecture Recovery Techniques Using Accurate Dependencies", https://ece.uwaterloo.ca
5. D.H. Hutchens and V.R. Basili, "System Structure Analysis: Clustering with Data Bindings," IEEE Trans. Software Eng., vol. 11, no. 8, pp. 749-757, Aug. 1985.
6. R.W. Schwanke and M.A. Platoff, "Cross References Are Features," Proc. Int'l Conf. Software Configuration Management, pp. 86-95, 1989.
7. D. Pollet, S. Ducasse, L. Poyet, I. Alloui, S. Cimpan, and H. Verjus, "Towards a Process-Oriented Software Architecture Reconstruction Taxonomy," Proc. 11th European Conf. Software Maintenance and Reeng., pp. 137-148, 2007.
8. A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering:A Review," ACM Computing Surveys, vol. 13, no. 3, pp. 264-323, Sept. 1999.
9. O. Maqbool and H.A. Babri, "Hierarchical Clustering for Software Architecture Recovery," IEEE Trans. Software Eng., vol. 33, no. 11, pp. 759-780, Nov. 2007.
10. K. Sartipi and K. Kontogiannis, "A User-Assisted Approach to Component Clustering," J. Software Maintenance and Evolution: Research and Practice, vol. 15, no. 4, pp. 265-295, July-Aug. 2003.
11. B.S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool," IEEE Trans. Software Eng., vol. 32, no. 3, pp. 193-208, Mar. 2006.
12. A. Shokoufandeh, S. Mancoridis, T. Denton, and M. Maycock, "Spectral and Meta-Heuristic Algorithms for Software Clustering," J. Systems and Software, vol. 77, no. 3, pp. 213-223, 2004.
13. M. Shtern and V. Tzerpos, "A Framework for the Comparison of Nested Software Decompositions," Proc. 11th IEEE Working Conf. Reverse Eng., pp. 284-292, 2004.

14. O. Maqbool and H.A. Babri, "The Weighted Combined Algorithm: A Linkage Algorithm for Software Clustering," Proc. Eighth European Conf. Software Maintenance and Reeng., pp. 15-24, 2004.
15. P. Andritsos and V. Tzerpos, "Information-Theoretic Software Clustering," IEEE Trans. Software Eng., vol. 31, no. 2, pp. 150-165, Feb. 2005.
16. O'Brien, L., "Dali: A Software Architecture Reconstruction Workbench", Software Engineering Institute, Carnegie Mellon University, May 2001.
17. P.J. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, et al. "The software bookshelf". IBM Systems Journal, 36(4):564–593, November 1997.
18. Imagix4D, http://ww.imagix.comImagix Corp.
19. Bauhaus group, "Tour de Bauhaus", http://www.Bauhausstuttgart.de/demo/index.html.
20. R. Naseem, O. Maqbooly, S. Muhammad, "Improved Similarity Measures For Software Clustering, 15th European Conference on Software Maintenance and Reengineering (CSMR)",pp. 45-54, 2011.
21. J. Davey and E. Burd, "Evaluating the Suitability of Data Clustering for Software Remodularization," Proc. Seventh Working Conf. Reverse Eng., pp. 268-277, 2000.
22. B.S. Everitt and S. Landau, "Cluster Analysis", fourth ed. Arnold Publishers, 2001.
23. N. Anquetil and T.C. Lethbridge, "Experiments with Clustering as a Software Remodularization Method," Proc. Sixth Working Conf. Reverse Eng., pp. 235-255, 1999.
24. S. Mancoridis, B. Mitchell, Y. Chen, and E. Gansner, "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures," Proc. Int'l Conf. Software Maintenance, pp. 50- 62, 1999.
25. Z. Wen and V. Tzerpos, "Software Clustering Based on Omnipresent Object Detection," Proc. 13th IEEE Int'l Workshop Program Comprehension, pp. 269-278, 2005.
26. V. Tzerpos, "Comprehension-Driven Software Clustering," PhD dissertation, Univ. of Toronto, 2001.
27. P. Tonella, F. Ricca, E. Pianta, and C. Girardi, "Using Keyword Extraction for Web Site Clustering," Proc. Fifth Int'l Workshop Web Site Evolution, pp. 41-48, 2003.
28. M. Saeed, O. Maqbool, H.A. Babri, S.M. Sarwar, and S.Z. Hassan, "Software Clustering Techniques and the Use of the Combined Algorithm," Proc. Seventh European Conf. Software Maintenance and Reeng., pp. 301-306, 2003.
29. J. Garcia, I. Ivkovic, N. Medvidovic,"A Comparative Analysis of Software Architecture Recovery Techniques", 28th International Conference on Automated Software Engineering (ASE), 2013.
30. O. Maqbool and H.A. Babri, "Automated Software Clustering: An Insight Using Cluster Labels," J. Systems and Software, vol. 79, no. 11, pp. 1632-1648, 2006.
31. B.S. Mitchell, "A Heuristic Search Approach to Solving the Software Clustering Problem," PhD dissertation, Drexel Univ., 2002.
32. J. Wu, A.E. Hassan, and R.C. Holt, "Comparison of Clustering Algorithms in the Context of Software Evolution," Proc. Int'l Conf. Software Maintenance, pp. 525-535, 2005.