Görselleştirme Tekniklerinin Yazılım Metriklerine Uygulanması

Muhammet Sebul BERATOĞLU[1] Feza BUZLUCA[2]

[1]Bilgisayar Bilimleri Bölümü, Bilişim Enstitüsü, İstanbul Teknik Üniversitesi, İstanbul
[2]Bilgisayar Mühendisliği Bölümü, İstanbul Teknik Üniversitesi, İstanbul

[1]e-posta: beratoglu@itu.edu.tr [2]e-posta: buzluca@itu.edu.tr

Özetçe

Görselleştirme karmaşık verilerin anlaşılabilir kılınmasında kendini kanıtlamış bir tekniktir. Yazılımdan elde edilen metrikleri incelerken, bu metrik değerlerini görselleştirerek analiz etmek, görselleştirme tekniklerinin yazılımdaki uygulamalarının bir alt bölümü olarak karşımıza çıkmaktadır. Bu çalışmada görselleştirme tekniklerini metriklere uygulayan çalışmalardan örnekler verilmiş, bu çalışmaların eksik kalan yanlarını tamamlayan bir görselleştirme sistemi önerilmiş, bu sistemin açık kaynak kodlu yazılımlara uygulanmasıyla elde edilen sonuçlar sunulmuş, modelin ve sonuçların nasıl iyileştirilebileceği tartışılmıştır. Geliştirilen renk tabanlı, iki boyutlu görselleştirme yöntemi; çok büyük ölçekli yazılımların metriklerini dahi rahatlıkla görselleştirip, birden fazla metriğin etkileşimini aynı anda analiz etme imkânı sunmaktadır.

1. Giriş

Yazılımın bakım, iyileştirme ve tekrar kullanma süreçlerinden sorumlu olan mühendisler öncelikli olarak üzerinde çalıştıkları yazılımı kavramak durumundadırlar. Endüstriyel kullanıma açık büyük ölçeklerdeki yazılımlarda böyle bir kavrayışa ulaşabilmek, özellikle belgelemenin yeteri kadar iyi olmadığı durumlarda, yoğun bir uğraş ve uzun zamanlar gerektirecektir. Yapılan araştırmalar bütün kaynakların %50 ila %75'lik kısmının yazılımın bakımı için ayrıldığını [1,2] ayrılan bu kaynağın büyük bir kısmının ise yazılımın kavranabilmesi için harcandığını göstermiştir.

Tersine mühendislik sayesinde yazılımlardan elde edilen yazılım metrikleri yazılımların anlaşılabilmesi, kontrol edilmesi ve yazılım kalitesinin iyileştirilmesine olanak sunmaktadır [3,4]. Ancak çok büyük ölçekli yazılımlar için elde edilen yazılım metriklerinden yazılım kalitesi ile ilgili çıkarsamalar yapmak zaman alacaktır. Ayrıca birden fazla metriği aynı anda değerlendirmek gerektiğinde tablolar halinde sunulan metriklerden gerekli bilgileri elde etmek kolay olmayacaktır [5].

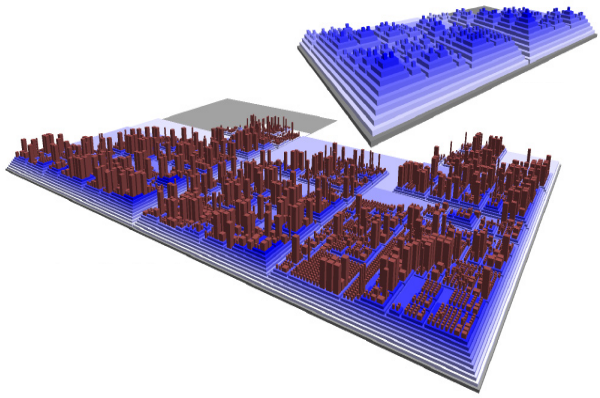
Görselleştirme karmaşık verilerin anlaşılabilir kılınmasında kendini kanıtlamış bir tekniktir. İnsan beyninin karmaşıklığın birçok farklı yönünü paralel olarak işleyebilmesine olanak tanır [6].

Bu çalışmada görselleştirme teknikleri kullanılarak, yazılım metriklerinden yazılım kalitesi hakkında fikir verebilecek bir görselliğe nasıl ulaşıldığı anlatılmıştır. 2. bölüm bu konuda yapılmış benzer çalışmalardan örnekler içermektedir. 3. bölüm geliştirilen görselleştirme yönteminin tanıtımına ayrılmıştır. 4. bölümde elde edilen sonuçlar sunulmuş ve son bölümde bu tekniğinin nasıl geliştirilip daha faydalı hale getirilebileceği tartışılmıştır.

2. Benzer Çalışmalar

Yazılımdan elde edilen metrikleri, çeşitli metaforlar kullanarak görselleştiren ve bu görselliği, yazılımı anlamak ve geliştirmek için kullanan çalışmalar daha önceden de yapılmıştır. Bu bölümde benzer çalışmalardan örnekler verilmiştir.

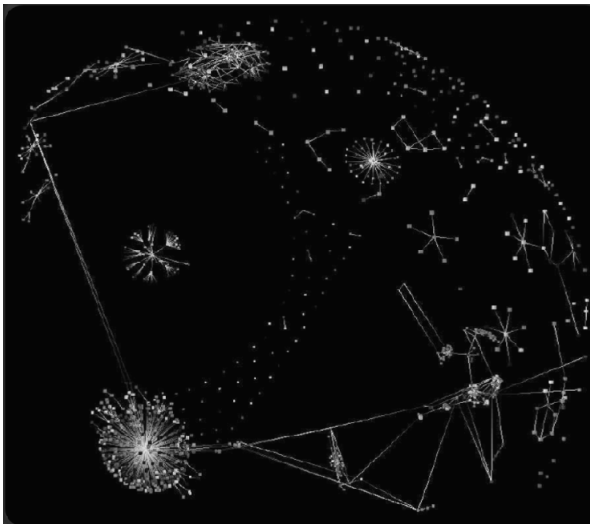
[7]'deki çalışmada şehir metaforu kullanılarak gerçeklik hissi yüksek bir görselliğe ulaşılmıştır (Şekil 1). Yazılımdaki her bir sınıf kahverengi bir çubukla temsil edilmiş, çubuğun yüksekliği metot sayısı (NOM) ile çubuğun eni ve boyu sınıfın nitelik sayısı (NAO) ile orantılı yapılmıştır. Aynı paket içindeki sınıfların bir bölge içinde şehrin mahallerini oluşturması sağlanmıştır. Kullanılabilen metrik sayısının sınırlı olduğu bu çalışmada görselliğin oluşturulması sınıf sayısı arttıkça zorlaşmaktadır, çok büyük ölçekli yazılımların görselleştirmesinde elverişli değildir. [5]'teki çalışmada da benzer bir şehir metaforu önerilmiş, programın akışının şehrin içinden akan araçlar ile temsil edilmesi düşünülmüştür ancak çalışma fikir bazında kalmıştır.



Şekil 1 Şehir modeli

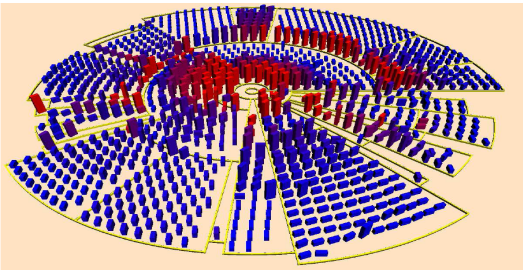
[8]'deki çalışmada tanıdık bir metafor olan güneş sistemi kullanılmış; kalıtım ağacının tepesindeki sınıflar evrenin merkezindeki gezegenler ve bu sınıflardan türetilmiş sınıflar uydular ile temsil edilmiştir. Gezegenlerin çapları sahip oldukları metot sayısı ile orantılı tutulmuştur. Az sayıda sınıfa sahip yazılımları görselleştiren bu çalışmanın endüstriyel bir yazılım için uygulaması bulunmamaktadır.

[9]'deki çalışmada yazılım metrikleri evren benzeri bir modelle temsil edilmiştir. Boyut indirgeme algoritmaları kullanılarak metrikler üç boyuta indirilmiştir. 3 boyutlu bir görsellik sunan bu yöntemde kullanıcının evren içinde kaybolma riski vardır. Şekil 2'de sınıfların kalıtım hiyerarşisi görünmektedir. VRML aracının temel gezinme teknikleriyle kullanıcının model içinde gezinme olanağı vardır, ancak daha derin ölçeklerde kullanıcının görsellik içinde kaybolma durumu oluşabilmektedir.



Şekil 2 Evren Modeli

[10]'deki çalışma iki boyutlu bir görsellik sunarken, insanın görsel algılama yetisinin karmaşık görsellikleri de rahatça kavrayabileceği varsayarak doku ve rengi birlikte kullanmıştır. Sınıfın dışa bağımlığı (Fan-Out) arttıkça daha karmaşık dokular, sınıfa olan bağımlılık (Fan-In) arttıkça daha koyu renkler kullanılmıştır. Bu iki metriğin aynı anda yüksek olması yazılım tasarımında istenilen bir durum değildir. Çalışmada potansiyel olarak problemli olan bu sınıfların hızlıca fark edilmesini sağlayan bir görselleştirme önerilmiştir. Yazılım boyutu arttıkça her bir sınıfa karşı düşen dokunun hesaplanması zaman alacağı ve sınıf sayısı arttıkça görsellikte her bir sınıfa ayrılan alan azalacağı için büyük ölçekli yazılımlar için uygun bir çalışma değildir.

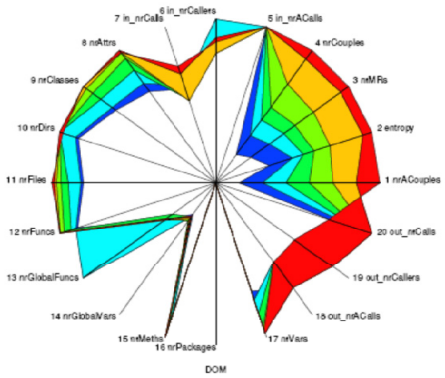


Şekil 3 Üç boyutlu kutular

Şekil 3'te bir örneği gösterilen çalışmada 3 boyutlu kutular görselleştirme yapıtaşları olarak kullanılmıştır [11]. Kutuların en, boy ve yüksekliklerinde birlikte dizilişlerinde yaptıkları açı ve renkleri de ayrı birer metriği temsil etmek üzere kullanılmıştır. Sınıf sayısı arttıkça görselleştirmenin yapılması ve bu üç boyutlu görsellik içerisinde gezinilmesi zorlaşmaktadır.

[12] deki çalışmada UML diyagramlarındaki her sınıfa karşı düşen dikdörtgenler üzerine sınıfa ait metrikleri gösteren üç boyutlu bir kutu eklemişlerdir. Yazılım boyutunun büyümesi UML diyagramların anlaşılabilirliğini azalttığı için bu görselliğin de algılanması zorlaşacaktır.

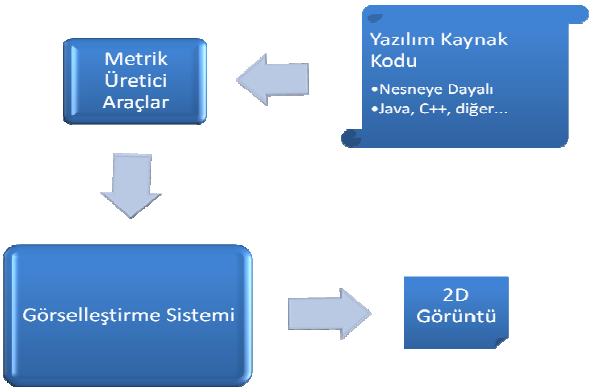
[13] deki çalışma yazılımın farklı sürümlerinden elde ettiği 20 farklı metriği yazılımın 7 farklı sürümü için Kiviat diyagramı kullanarak bir arada vermiştir. Şekil 4'te görüleceği üzere yazılımın farklı sürümleri için metriklerinin aldığı değerler tek bir görsellikle özetlemiştir. Bu çalışmada sürümler arttıkça azalan metrik değerlerinin ara sürümlerde aldığı değerleri görmek mümkün olamamaktadır.



Şekil 4 Yazılım Evrimi

3. Geliştirilen Yöntem

Literatürdeki çalışmaların büyük ölçekli yazılımları desteklemede ve görselliğe hızlı bir şekilde ulaşmada eksiklikleri olduğu görülmüştür. Büyük ölçekli yazılımları kolaylıkla gösterebilmek üzere geliştirilen yöntemde renk tabanlı bir görsellik tasarlanmıştır. Bu amaçla geliştirilen Görselleştirme Sisteminin girdisi nesneye dayalı yazılımların metrikleri, çıktısı ise iki boyutlu görüntüdür (Şekil-5).



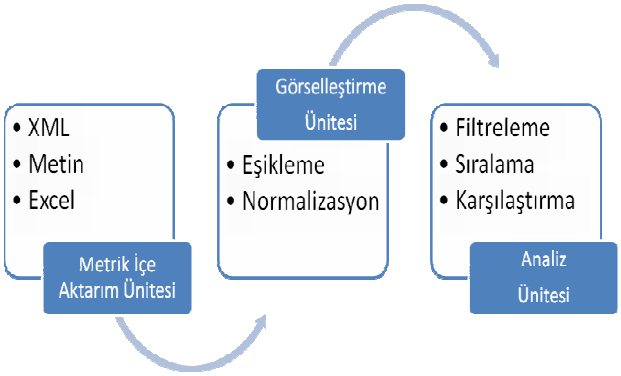
Şekil 5 Görselleştirme Sistemi

CodeGear RAD Studio 2007 [1]ortamında geliştirilen uygulama Windows 2000, XP ve Vista platformlarında herhangi ek bir bileşene gereksinim duymadan çalıştırılabilir. Görselleştirme sistemi üç ana üniteden oluşur (Şekil 6).

1. Metrik Aktarım Ünitesi
2. Görselleştirme Ünitesi
3. Analiz Ünitesi

3.1. Metrik Aktarım Ünitesi

Bu çalışmada yazılım için yeni metrik kümeleri tanımlamak ya da farklı metrikleri elde etmek için yeni yöntemler geliştirmek değil, var olan metrik üretici uygulamalarla bilinen metrikleri kullanarak yeni bir görselliğe ulaşmak hedeflenmiştir. Bu doğrultuda yazılımların kaynak kodlarından üretilen metrikler veri olarak kabul edilmiştir. Ünite basit metin, XML ve Excel formatlarındaki metrikleri destekleyecek genellikte yapılmıştır. Metrik Aktarım Ünitesi her bir sınıf için *Paket, Sınıf, Metrik Tanımı, Metrik Değeri* kümesinden oluşan bir bilgi kümesi oluşturur.



Şekil 6 Görselleştirme Üniteleri

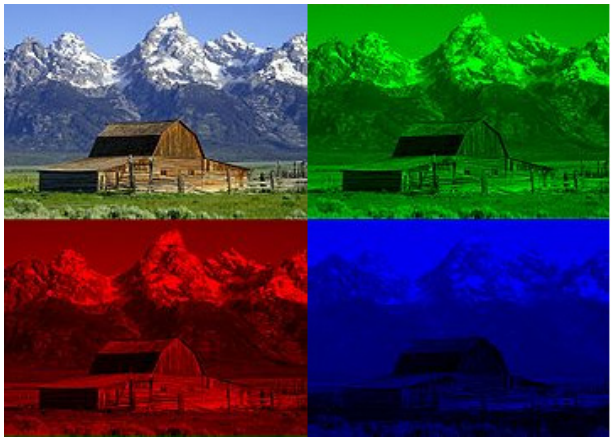
3.2. Görselleştirme Ünitesi

Renk, ışığın değişik dalga boylarının gözün retinasına ulaşması ile ortaya çıkan bir algılamadır. Bu algılama, ışığın maddeler üzerine çarpması ve kısmen soğurulup kısmen yansıması nedeniyle çeşitlilik gösterir ki bunlar renk tonu veya renk olarak adlandırılır. Gördüğümüz renkler değişik ışık frekanslarının kombinasyonlarıdır[2].

Görselleştirme için kullandığımız temel prensip metrik değerlerini farklı ışık frekansları olarak değerlendirip, birleşimlerine karşı düşen renkler toplamını bir görüntüde sunmaktır.

Renkleri elde edebilmek ve tanımlayabilmek için değişik renk uzayı modelleri oluşturulmuştur. RGB uzayı bütün renkleri Kırmızı, Yeşil ve Mavi renklerinin birleşimi olarak tanımlar (Şekil 7).

Görselleştirme ünitemiz bu modellerden RGB ve HSV renk uzaylarını kullanmıştır. Metrik aktarım ünitesinden gelen metrikler R,G,B ya da H,S,V renk bileşenlerine atanmıştır.



Şekil 7 Yeşil, Kırmızı, Mavi bileşenleri ile RGB renk uzayı modeli

Metriklerin renk bileşenlerine atanması esnasında dikkat edilmesi gereken önemli noktalardan birisi de normalizasyon işlemidir. Metriklerin aldığı değerler ile renk bileşenlerinin alabileceği değerler uyumlu olmayabilir. Bazı metrikler $[0,\infty)$, bazıları sonlu aralıkta gerçel ya da tamsayı üretirken bazıları sınırlı bir aralıkta gerçek sayılar üretebilirler [14]. Öte yandan RGB modelinde her bir bileşenin alabileceği değerler 0-255 arasında değişirken, HSV modelinde H, S, V bileşenlerinin alabileceği değerler sırasıyla [0,360], [0,100], [0,100] aralığında olabilmektedir.

Görselleştirme ünitesi doğrusal normalizyon metodu ile metrik değerlerini renk bileşenlerinin alabileceği değerlere uyumlu hale getirir. Kullanıcı farklı eşik değerleri belirleyerek bu değer aralıklarını kısıtlayabilir.

3.3. Analiz Ünitesi

Görselleştirme ünitesinin çıktısı olan iki boyutlu görüntü, analiz ünitesinin aşağıda özetlenen özellikleri yardımıyla analiz edilir.

Odak: Bu özellik ortaya çıkan görsellik içerisinde yazılımın tüm sınıflarını birden görme ile sadece tek bir sınıfı görme arasındaki tüm düzeylere odaklanabilme imkânı sunar.

Bilgi: Sistem görsellik üzerinde dolaşırken üzerinde bulunan her bir beneğin karşı düştüğü sınıfı, sınıfın içinde bulunduğu paketi, sınıfın yer aldığı kaynak kodunu ve bu piksel değerini oluşturan renk bileşenlerinin değerleri ile metrik değerlerini gösterir.

Sıralama: Analiz esnasında görüntü istenilen metrik değerine göre sıralanabilir. Örneğin DIT ve NOM metriklerinin oluşturduğu bir görsellikte DIT metriğine göre yapılan sıralama sınıfları kalıtım seviyesine göre dizerek sınıfların sahip oldukları metot sayılarının kalıtım seviyelerine göre dağılımını görebiliriz.

Filtreleme: Birden fazla metriğin renk bileşenlerine atanmasıyla oluşan görsellikte herhangi bir bileşenin etkisi

[1] http://www.embarcadero.com/products/delphi/

[2] http://en.wikipedia.org/wiki/Color

açılıp kapatılabilir. Böylelikte bileşenin tek başına ya da farklı ikili kombinasyonlarla analiz edilebilir.

Görüntü saklama ve karşılaştırma: Elde edilen sonuçlar analiz ünitesinde saklanabilir ve yazılımın farklı sürümlerindeki sonuçlarla ya da farklı yazılımların görsellikleriyle karşılaştırılabilir.

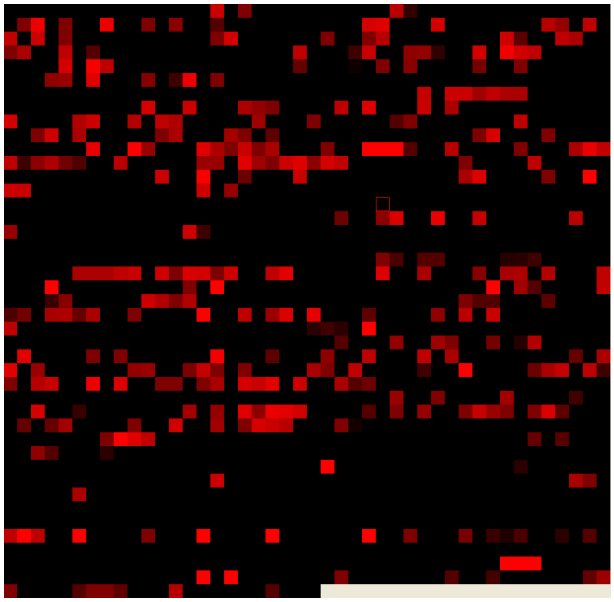
4. Sonuçlar

Görselleştirme için örnek uygulama olarak açık kaynak kodlu modelleme aracı ArgoUML [3] 2.6.0 sürümü seçilmiştir. Diğer çalışmaların da örnek olarak seçtiği bu yazılımın metrikleri ise yine herkesin erişimine açık olan Eclipse [4]ortamındaki Metrics [5]eklentisi kullanılarak elde edilmiştir.

Görselleştirme ile elde edilen sonuçlar üç ayrı kategoride ele alınmıştır.

4.1. Özet Bilgi

ArgoUML yazılımına ait LCOM metriklerinin görselleştirilmesiyle oluşan görüntü Şekil 8'de gösterilmiştir. LCOM (Lack of Cohesion in Methods) bir sınıfa ait metotlarının birbiriyle uyumluluğunun bir göstergesidir. Sınıfın uyumluluğunun düşük olması, sınıfın 2 veya daha fazla alt parçaya bölünmesi gerektiğini gösterir. Düşük uyumluluk karmaşıklığı artırır, bu nedenle geliştirme aşamasında hata yapılma ihtimali yükselir [14].



Şekil 8 Özet Bilgi – LCOM metriği

Görselliği elde edebilmek için LCOM metriği 0-255 aralığına doğrusal olarak normalize edilmiş ve elde edilen değerler görüntünün R bileşenine atanmıştır.

[3] http://www.argouml.tigris.org

[4] http://www.eclipse.org

[5] http://metrics.sourceforge.net

Şekil 8'deki görsellikte yazılımdaki sınıfların uyumluluk dağılımı görülmektedir. Sınıf içi uyumluluk ne kadar düşük ise sınıfa karşı düşen piksel o oranda kırmızıdır. Dikkatimizi çeken kırmızı beneklerin üzerine gidilerek uyumluluğu düşük olan sınıfa çabucak ulaşabiliriz.

Analiz ünitesinin sıralama özelliği kullanılarak Şekil 8'deki görüntü R bileşenine göre sıralanmış ve Şekil 9'daki görüntü elde edilmiştir. Bu görsellik bize uyumlu sınıflar ile uyumsuz sınıflar arasında nasıl bir dağılım olduğunu ve uyumsuz sınıfların bütün içindeki oranını gösterir.

Şekil 9 Özet Bilgi - LCOM metriği - sıralı

Görselliği kısa bir süre inceleyerek dahi 1871 sınıftan oluşan bir yazılımda, sınıfların ne kadar uyumlu olarak tasarlandığı hakkındaki bir fikir edinebiliriz. Görselleştirmemiz yazılım hakkında hızlı bir şekilde özet bir bilgi sunar.

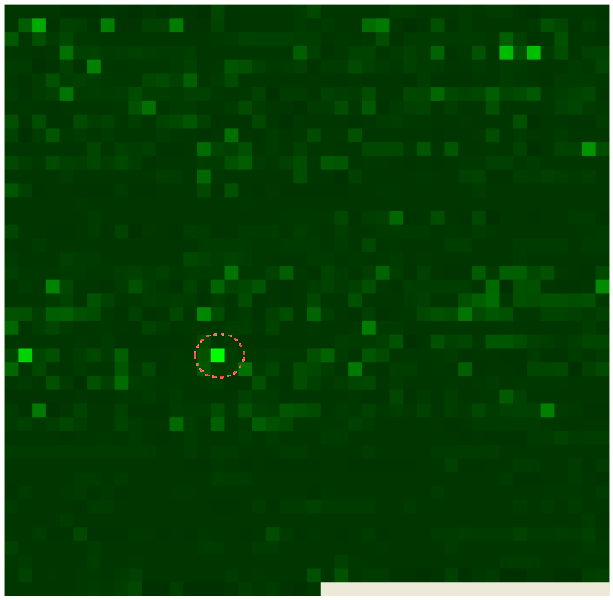
4.2. Görsel Kıyaslama

NOM (Number of Methods) bir sınıfın toplam metot sayısına karşı düşen metriktir. Sınıfın metot sayısı sınıfın karmaşıklığının bir göstergesidir [14]. ArgoUML yazılımına ait sınıfların NOM metriklerinin görselleştirmesiyle elde edilen görüntü Şekil 10'da gözükmektedir.

Görselliği elde edebilmek için NOM metriği 0-255 aralığına doğrusal olarak normalize edilmiş ve elde edilen değerler görüntünün G bileşenine atanmıştır.

Görüntüde kimi piksellerin diğerlerine kısayla daha parlak olduğu kolaylıkla görülmektedir. Görselliğin bize sunduğu önemli avantajlardan birisi bu kıyaslama olanağıdır. Metrik değerlerini sayısal olarak inceleyerek de bir kıyaslama yapabiliriz ancak burada sayısal metrik değerlerini teker teker ele alıp karşılaştırmak yerine görsel algı mekanizmamızın görüntüyü paralel olarak işleyebilen yetenekleri kullanarak daha kapsamlı bir sonuca daha efektif bir yolla ulaşabiliriz. Görsel algı mekanizmamız, şekildeki (1871 sınıfa ait) bütün pikselleri paralel olarak kıyaslayıp sadece baskın olan pikselleri hızlı bir şekilde bulmakla kalmaz aynı zamanda baskınlığın derecesi ve yoğunluğu hakkında da bir fikir sahibi olmamızı sağlar. Böyle bir özellik bize örneğin beneklerin sık

bir şekilde baskın olduğu bir görselliğinde sahip yazılımın tasarımının çok sayıda karmaşık sınıfları barındırdığı hakkında bir uyarı elde etmemizi sağlayacaktır.



Şekil 10 Görsel Kıyaslama - NOM metriği

Şekil 10'da en parlak olarak algıladığımız beneği, Analiz ünitesinin Odak ve Bilgi özelliğini kullanarak incelediğimizde karşı düşen sınıfın metrik değerinin diğer bütün sınıflardan daha yüksek dolayısıyla en fazla metoda sahip sınıf olduğunu görebiliriz (Tablo 1).

Tablo 1 UMLModelElementListModel2 renk değerleri

NOM	R	G	B
117	0	255	0

4.3. Metriklerin Bir Arada Gösterilimi

Kimi durumlarda tek bir metrik değeri bakmak yerine birden fazla metriğin aldığı değere bir arada bakmak gerekebilir. Bu bölümde birden fazla metrik değerini aynı anda ele alıp görselleştirdiğimiz durumlara örnekler verilecektir

4.3.1. Bakım Maliyeti Yüksek Olan Sınıfların Bulunması

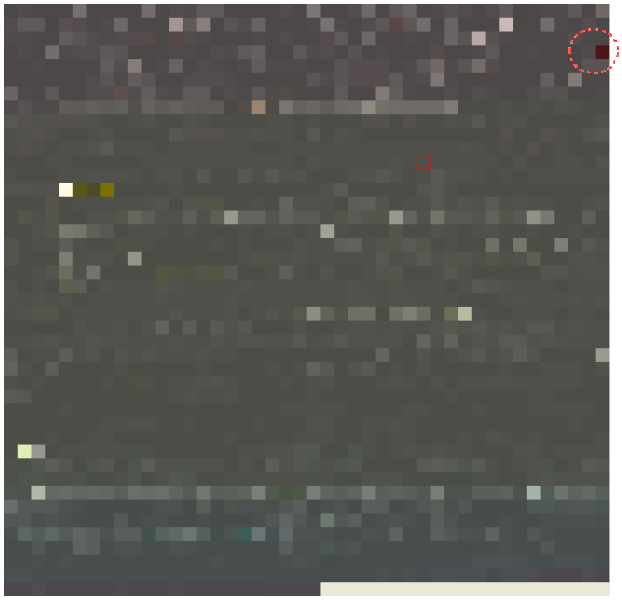
WMC (Weighted Methods per Class) metriği, bir sınıfın tüm metotlarının karmaşıklığının toplamıdır [4]. WMC metriği yazılımdaki sınıfların geliştirilmesine ve bakımına ne kadar zaman harcanacağı hakkında fikir verebilecek bir metriktir. Metot sayısı çok olan taban sınıflar, çocuk düğümlerde daha çok etki bırakırlar. Çünkü tanımlanan tüm metotlar türetilen sınıflarda da yer alacaktır [14]. O halde WMC değerini tek başına değerlendirmek yerine çok sayıda çocuk düğüme sahip taban sınıflar için aldığı yüksek değerleri analiz etmek daha doğru olacaktır.

DIT (Depth of Inheritance Tree) sınıfın, kalıtım ağacının köküne olan uzaklığıdır. NSC ise (Number of Children) sınıfa ait çocuk düğümü sayısıdır [4].

Şekil 11'de WMC metriği ile DIT ile NSC metrikleri birlikte görselleştirilmiştir.

Görselliği elde edebilmek için DIT metriği 0-300 aralığına normalize edilerek H bileşenine, NSC metriği 10-100 aralığına normalize edilerek S bileşenine, WMC metriği 30-100 aralığına normalize edilerek V bileşenine atanmıştır. Aralıklar görselliğin görünürlüğü ve anlaşılabilirliğini arttırmak amacıyla sınırlandırılmıştır.

Görüntü, analiz ünitesinin sıralama özelliği kullanılarak, DIT değerlerine göre küçükten büyüğe doğru sıralanmıştır. Böylelikle kalıtım ağacının tepesindeki sınıflar görüntünün yukarısında, kalıtım ağacının sonundaki sınıflar görüntünün en altında toplanmıştır. V bileşeni beneklerin parlaklığını kontrol ettiğinden parlak olarak gözüken sınıflar WMC değeri yüksek olan sınıflardır. S bileşeni renge doygunluk miktarını kontrol ettiği için benekler ne kadar renkli gözüküyorlarsa NSC metrik değerleri o kadar yüksektir. O halde yazılımın bakımı esnasında dikkat etmemiz gereken sınıfları bulmak istiyorsak parlak, renkli ve de resmin yukarılarında olan beneklere karşı düşen sınıfları analiz etmemiz gerekir. Bu sınıflar kalıtım ağacının tepesinde, çok sayıda çocuk düğüme sahip, karmaşık metotları içeren sınıflar olacaktır.



Şekil 11 Metriklerin birleştirilmesi - Bakım Maliyeti

Şekil 11 de halka içine alınan ve kırmızı rengiyle dikkat çeken benek, analiz ünitesinin bilgi ve odak özellikleri kullanılarak incelendiğinde gerçekten de kalıtım ağacının tepesine yakın çok fazla çocuk sınıfa sahip ve karmaşıklığı yüksek bir sınıf olan UMLModelElementListModel2'ye ulaşılır. Bu sınıfa ait metrikler Tablo 2'de, karşı düşürülen beneğin renk değerleri Tablo 3'de görülmektedir. Görsellik kullanılarak potansiyel olarak bakım maliyeti yüksek olan bir sınıf kolaylıkla bulunmuştur.

Tablo 2 UMLModelElementListModel2 metrikleri

DIT	NSC	WMC
3	108	79

Tablo 3 UMLModelElementListModel2 renk değerleri

H	S	V
55	99	48

4.3.2. Büyük Sınıfların Bulunması

Kalitesiz kod göstergeleri kavramı ilk olarak, Martin Fowler tarafından var olan kodların iyileştirilmesi için ortaya koyulmuştur [15]. Düşük kaliteli kod göstergelerinden birisi de *Büyük Sınıf*'tır. Çok fazla sayıda metoda sahip, çok sayıda üye nitelik değişkeni bulunduran ve genellikle kalıtım ağacının tepelerinde yer alan sınıflar büyük sınıflardır.

Büyük sınıfların tespiti için yapılan görselleştirme Şekil 12'de gösterilmiştir.

Şekil 12 Metriklerin Birleştirilmesi - Büyük Sınıfların Tespiti

NOF (Number of Attributes) bir sınıfın toplam nitelik sayısına karşı düşen metriktir [4]. Görselliği elde edebilmek için DIT metriği 0-300 aralığına normalize edilerek H bileşenine, NOM metriği 0-100 aralığına normalize edilerek S bileşenine, NOF metriği 0-100 aralığına normalize edilerek V bileşenine atanmıştır. Görüntü DIT değerlerine göre küçükten büyüğe doğru sıralanmıştır.

Ortaya çıkan görsellikte görüntünün yukarılarında yer alan parlak ve renkli benekler büyük sınıf olmaya aday sınıflara işaret etmektedir. Gerçekten de bu beneklere karşı düşen sınıfların NOM ve NOF metrik değerleri yüksek, DIT metrik değeri düşüktür. Şekil 12'de halka içine alınan, rengi ve parlaklığıyla dikkati çeken beneğin metrik değerleri Tablo 4'de, taşıdığı renk değerleri Tablo 5'de görülmektedir. Analiz ünitesinin odak ve bilgi özellikleri kullanılarak beneğe karşı düşten sınıfın ProjectImpl olduğu tespit edilmiştir. Bu sınıf sahip olduğu çok sayıdaki metot ve nitelikleri ile potansiyel bir büyük sınıftır.

Tablo 4 ProjectImpl metrikleri

DIT	NOM	NOF
1	79	24

Tablo 5 ProjectImpl renk değerleri

H	S	V
55	99	48

5. Tartışma

Görselleştirirken kullandığımız bütün değerler yazılımdan elde edilmiş metriklerden oluştuğuna göre görselliği analiz ederek ulaştığımız tüm sonuçlara metrikleri analiz ederek de ulaşabilirdik. Ancak, metrikler de sonuçta yazılımı anlamaya çalışan yazılımcılar için üretilen değerler olduğuna göre, görselleştirmenin anlamayı kolaylaştıran, hızlandıran ve yazılımı bir fotoğraf olarak özetleyen niteliklerini hafife alamayız.

Bu çalışmada anlatılan yöntem çok büyük ölçekli yazılım metriklerini hızlı bir şekilde görselleştirme imkânı sunmakta, birden fazla metriğin etkileşimini aynı anda analiz etme fırsatı vermektedir. Sunulan görselleştirme metodunu geliştirmek için; üç metrikten daha fazlasını aynı anda görselleştirmek, metrik değerlerini normalize ederken doğrusal olmayan yöntemleri de denemek ve bir arada ele alınan metriklerin birbirini örtmeden görselleştirilmesi üzerine çalışmalar yapmamız gerekmektedir.

Nispeten yeni bir kavram olan yazılımın görselleştirilmesi üzerinde umut verici çalışmalar yapılabilir. Bunlardan bazıları aşağıda sıralanmıştır.

Yazılım Evrimi: Yazılımın farklı sürümleri arasında geçirdiği evrim, her bir sürüme ait görselleştirme çıktıları ile takip edilip, yazılım kalitesindeki değişim hakkında fikir elde edilebilir.

Görüntü İşleme Teknikleri: Görselleştirme teknikleri görüntü işleme teknikleri ile birleştirilerek görselleştirmenin sonuçta bir görüntü olan çıktısı üzerinde gözle kolayca fark edilemeyen yeni bilgilere ulaşılabilir.

Hizmet Odaklı Yaklaşım: Hizmet odaklı yaklaşım [16] popülerliği gittikçe artan bir yazılım tasarım yöntemidir. Ancak, bu yönteme göre tasarlanmamış mevcut yazılımlarda hizmetleri tespit etmek çözülmesi gerekli problemler arasındadır. Görselleştirme ile yazılımlardaki hizmet odaklı yapıların bulunması üzerinde çalışmalar yapılabilir.

6. Kaynakça

[1] B. W. Boehm, *Software Engineering Economics,* PrenticeHall, 1981.

[2] B. Lientz, E. Swanson, and G. E. Tompkins., "Characteristics of application software maintenance", *Communications of the ACM*, 21(6), 1978.

[3] A. Endres and D. Rombach., "A Handbook of Software and Systems Engineering", *Addison*-Wesley, 2003.

[4] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Trans. Software Eng.,*20(6), 476- 493, 1994.

[5] Panas, T., Berrigan, R. & J.C.Grundy (2003), "A 3d metaphor for software production visualization", in E. Banissi, K. Borner, C. Chen, G. Clapworthy, C. Maple, A. Lobben, C. J. Moore, J. C. Roberts, A. Ursyn & J. Zhang, eds, *'Seventh International Conference on Information Visualization,* IV 2003, 16-18 2003, London, UK', IEEE Computer Society, 314–319.

[6] Stasko, J. T., Domingue, J., BROWN, M. H., ve Price, B. A., Eds, 1998, "Software Visualization Programming as a Multimedia Experience", *The MIT Press.*

[7] Richard Wettel, Michele Lanza, "Visualizing Software Systems as Cities," vissoft, 92-99 *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2007.

[8] Graham, H., Yang, H.Y. and Berrigan, R., "A Solar System Metaphor for 3D Visualization of Object Oriented Software Metrics", *In Proc. Australasian Symposium on Information Visualization, (invis.au'04),* Christchurch, New Zealand*, 2004.*

[9] C. Lewerentz ve F. Simon., "Metrics-based 3d visualization of large object-oriented programs", *In 1st Int. Workshop on Visualizing Software for Understanding and Analysis*, 2002.

[10] Holten, D.H.R., Vliegen, R., Wijk, J.J., "Visualization of software metrics using computer graphics techniques", *Proceedings of the 12th Annual Conference of the Advanced School for Computing and Imaging*", 135-140, Belgium, 2006.

[11] Guillaume Langelier, Houari Sahraoui, Pierre Poulin, "Visualization-based Analysis of Quality for Large-scale Software Systems", *In International Conference on Automated Software Engineering (ASE 2005)* ,7-11, 2005, Long Beach, California, USA.

[12] Termeer, M., Lange, C.F.J., Telea, A.C. & Chaudron, M.R.V., "Visual exploration of combined architectural and metric information", *Proceedings 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005),* 21-26, Budapest, Hungary, 2005.

[13] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing Multiple Evolution Metrics", *InSoftVis '05: Proceedings of the 2005 ACM Symposium on Software Visualization*, 67–75, New York, NY, USA, 2005. ACM Press.

[14] U. Erdemir, U. Tekin, F. Buzluca, "Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi", 10/2008, s. 249-258, *Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS08)*, İstanbul, 2008.

[15] M. Fowler and K. Beck, "Bad Smells in Code", *in Refactoring: Improving the Design of Existing Code*, *Addison-Wesley*, 2000, pp. 75-88.

[16] Bell, Michael (2008). "Introduction to Service-Oriented Modeling", Service-Oriented Modeling: Service Analysis, Design, and Architecture. *Wiley & Sons*, 3. ISBN 978-0-470-14111-3.