Nesneye Dayalı Yazılımlarda Yapısal Değişimlerin Analizi ve Maliyetlerinin Hesaplanması

Sinan Eski[1] Feza Buzluca[2]

[1]TÜBİTAK-BİLGEM, Kocaeli
[2]Bilgisayar Mühendisliği Bölümü, İstanbul Teknik Üniversitesi, İstanbul

[1]e-posta: eski@bilgem.tubitak.gov.tr [2] e-posta: buzluca@itu.edu.tr

Özetçe

Yazılımlar yeni özellik ekleme, kalite iyileştirme ve hata düzeltme işlemleri sebebiyle sürekli değişime uğramaktadır. Bu çalışmada nesneye dayalı yazılımların sürümleri arasındaki değişimlerin ölçülmesi amaçlanmaktadır ve "değişim maliyeti" kavramı değişimlerin miktarını göstermek için kullanılmıştır. Değişim maliyetlerinin hesaplanması için nesneye dayalı yazılımlardaki temel değişimler dikkate alınarak bir sınıflandırma yapılmıştır. Bu değişimlere etkilerine göre farklı ağırlıklar verilerek sınıfların değişim maliyetleri hesaplanmıştır. Bu sayede yazılımın sürümleri arasında en çok değişimin hangi sınıflar üzerinde yapıldığı, hangi sınıfların kararlı ve tekrar kullanılabilir, hangi kısımların kararsız ve çok değişen olduğu bilgilerine ulaşılabilir. Değişim maliyeti çok yüksek olan ve yazılımın son sürümlerine yakın kısımlarda da değişime uğrayan sınıflar tehlikeli ve kritik sınıflar olarak ifade edilebilir. Bu kısımlar daha çok ve öncelikli olarak sınanmalı; bakım ve yazılımın gözden geçirilmesi sırasında öncelikli olarak değerlendirilmelidir. Bu sayede proje yöneticileri sınama için ayıracakları kaynakları daha verimli bir biçimde ayarlayabilir; yazılım geliştiricileri yazılımın düşük kaliteli bu kısımlarından başlayarak yazılımın iyileştirilmesini sağlayabilirler.

1. Giriş

Bir yazılımın tarihsel gelişim bilgisi incelemek o yazılım hakkında daha çok bilgi edinmemizi ve daha doğru kararlar vermemizi sağlar. Yazılımlar hayat döngüleri içersinde sürekli değişime uğrarlar. Hata düzeltme, yeni özellik ekleme ve kalite iyileştirme başlıca değişim nedenleridir. Ancak, yazılımların kaliteli kısımlarında, hata çıkma olasılığı diğer kısımlarına göre daha azdır; ayrıca iyi bir tasarımda bu kısımlar eklenecek yeni özelliklerden az etkilenir. Diğer taraftan kalite özellikleri daha düşük olan sınıflar hataya açıktır ve gelen değişiklik isteklerinden çok etkilenir. Dolayısı ile yazılımlardaki düşük kaliteli kısımlar yazılımların kritik kısımlarıdır ve yazılımın diğer kısımlara göre daha çok değişir.

Önceki çalışmalarımızda [1,2] yazılımların kaynak kodundaki "yapısal değişimleri" analiz eden ve yazılımın gerçekten değişen kısımlarını bulan değişim analiz yöntemi, çalışmanın sonuçlarının doğrulanması için geliştirilmiştir. Bu çalışmada ise bu yöntem yazılımın gelişiminin incelenmesi ve yorumlanması için kullanılmıştır.

Çalışma kapsamında yazılımların sürümleri arası değişimleri ölçmek için "değişim maliyeti" kavramı tanımlanmıştır ve bu kavram ardışıl iki sürüm arasındaki farkların ölçülmesi için kullanılmıştır. Değişim maliyeti hesaplanırken, yazılımlardaki değişimlerin yazılım mimarisini ve yapısını etkileme miktarını hesaba katmak için değişim türleri tanımlanmış ve bunlara ağırlıklar verilmiştir. Bu sayede, yorumlarda yapılan düzenlemeler gibi yazılım mimarisini etkilemeyen "yapısal olmayan değişimler" hesaba katılmaz iken, yazılımın birçok yerini etkileyebilecek bir değişim olan metot imzasındaki değişimin etkisi daha ön plana çıkartılmıştır.

Değişim analizi yöntemi ile yazılımın ardışıl sürümleri incelenerek sınıflardaki değişimler çıkarılmakta ve sınıfların değişim maliyetleri hesaplanmaktadır. Bu yöntem ile yazılımın sürümleri arasındaki değişim miktarı nicel olarak ifade edilmektedir. Dolayısı ile bir sürümden diğerine geçişte yapılan değişimlerin ölçülmesi sağlanmaktadır. Hangi sürümlerde en çok değişimin yapıldığı, bu değişimlerin hangi sınıflar üzerinde ne kadarlık bir etkiye sahip olduğu ve bunlarla birlikte sınıfların değiştiği sürümler önerilen yöntem sayesinde görülmektedir.

Değişim maliyetleri dikkate alınarak yazılımlardaki kritik sınıflar belirlenebilir. Bu sınıflar olgunlaşmamış ve kararsız olabileceğinden öncelikli olarak ve daha çok sınanmalıdır. Ayrıca gözden geçirme ve yeniden yapılma işlemlerinde bu sınıflar öncelikli olarak dikkate alınmalıdır.

Yazının planı şu şekildedir. Bölüm 2'de önceki yapılan çalışmalardan bahsedilmiştir. Değişimlerin sınıflandırılması ve hesaplama yöntemleri Bölüm 3'te, yapılan deneysel çalışma Bölüm 4'de açıklanmıştır. Bölüm 5 sonuçlar ve ileriki çalışmaları içermektedir.

2. Literatür Özeti

Bazı araştırmacılar, sistem tasarımının değişebilirliğe ve değişimin yayılmasına olan etkisini araştırmışlardır. M.Chaumun ve diğerleri yaptıkları araştırmada [3] sistem tasarımının değişebilirliğe etkisini incelemek için bir değişim etkisi modeli oluşturmuşlardır. Yazılımın bir yerinde yapılan değişimin, yazılımdaki diğer hangi kısımları etkilediğini bulmanın; yazılımın değişimden sonrada kararlı ve doğru çalışmasını sağlamak için önemli olduğunu vurgulamışlardır. Ana odakları, sistemin bir değişimi nasıl karşılayacağını ve sistemin değişebilirliğinin değişimleri ne kadar soğurabildiğidir.

Bazı araştırmacılar, yeni eklenen bir özelliğin, var olan bir özelliğin değiştirilmesinin sınıfları etkileme olasılıklarını dikkate alarak değişim eğilimli sınıfları olasılıksal yaklaşımlarla belirleme üzerine çalışmışlardır. Tsantalis ve diğerleri bu konuda yaptıkları çalışmada [4], bu yaklaşımla, nesneye dayalı sistemlerde verilen bir değişime karşılık diğer sınıfların değişim olasılıklarını belirlemeye yönelik bir yöntem önermişlerdir. Sharafat ve Tahvildari'nin makalesinde [5] yazılımların, Tsantalis ve diğerlerin önerdiği yönteme

yakın olarak, kaynak kodunun geçmiş sürümleri ve UML diyagramlarındaki bağımlılıkları inceleyerek, ilerleyen sürümlerde hangi sınıfların değişeceğini olasıksal olarak hesaplamaktadır. Bir diğer çalışmada [6], [4] de kullanılan yöntemin çok sınırlı bir versiyonunu kullanarak bir teknik sunmuşlardır; ancak gerçekleme ya da benzeri diğer yöntemlerle karşılaştırma vermemişlerdir.

Bir çalışmada [7], yazılımın önceki sürümleri, sürüm denetim sistemleri üzerinden incelenerek kaynak kodun bir yerinde yapılan değişimin ilgili olabileceği diğer kısımları geliştiricilere önermek için bir yöntem geliştirilmiştir.

L.Li ve A. Offut kullanıcı tarafından tanımlanarak girilen bir değişimin etkisini belirlemek için değişimin bağımlılık graflarına dayalı bir sistem önermiştir [8].

Yazılımın gelişim sürecindeki metriklerin değişimlerini analiz eden, bunların projelerin ilerleyen aşamalarındaki kalite değişimi hakkında bilgi çıkarmak üzerine yapılmış çalışmalar vardır.

Young Lee ve diğerleri tarafından yapılan çalışmada [9] fan-in/out bağımlılık ve uyum metrikleri ile yazılımın gelişim davranışlarını anlamak için açık kaynak kodlu JFreeChart yazılımı analiz edilmiştir. Yaptıkları deneysel çalışmada incelenen yazılımdaki sınıf sayısı artışı ile bağımlılık ve uyum metrikleri arasındaki ilişkiyi incelemişlerdir. Elde edilen sonuçlar bağımlılık metriklerinin sınıf sayısı ile ilişkili olduğunu göstermiştir. Uyum metriği için ise negatif korelasyon bulumuştur. Yeni eklenen sınıfların, silinen sınıflara göre fan-in metrik değerleri yüksek, fan-out metrik değerleri ise daha düşüktür. Bu da tekrar kullanılabilirlik için istenen bir durumdur.

Yedi çevik (agile) geliştirme yöntemi ile geliştirilen proje, nesneye dayalı metriklerinin bu projelerdeki değişimlerini ve farklı proje içeriklerinin kaynak koddan nasıl etkilendiğini belirlemek için incelenmiştir [10] Metriklerin kullanılmasının, çevik geliştirme yöntemi ile geliştirme yapan grupların yazılımı anlamalarını kolaylaştıracağını, iyileştirmelerin ve gelişimin sağlanması için günlük işlemlerini belirlemede yardımcı olacağını ileri sürmektedirler.

Bu çalışmada, sınıfların değişim maliyetleri çıkarılmıştır. Yapılan deneyler sırasında; yazılımların değişim bilgileri, yazılımların ardışıl sürümlerinden elde edilmiştir.

3. Değişim Analizi

Bu çalışmada, yazılımlardaki değişimleri tasarımsal düzeyde incelemek için, kaynak kod analizine dayalı bir yöntem izlenmiştir ve değişim sınıfları oluşturulmuştur.

3.1. Değişimlerin Sınıflandırılması

Değişimlerin sınıflarındırılmasında; nitelik, metot, sınıf gibi nesneye dayalı yazılım bileşenleri, bunlar arasındaki ilişkiler ve kaynak kod üzerindeki olası değişim türleri göz önünde bulundurulmuştur. Ayrıca, bu değişim sınıflarına ağırlıklar verilmesi ile hesaplanan değişim maliyeti kavramı tanımlanmıştır.

Yazılımın mimari değişimleri, yazılımın gelişimsel süresince kaynak kodu incelenerek çıkarılmıştır. Yazılımın kaynak kodundaki değişimler: yapısal değişimler ve yapısal olmayan değişimler olarak iki kısıma ayrılmıştır. Yapısal değişikler yazılımın derlenen çalıştırılabilir kodunu değiştirirken, yapısal olmayan değişimler çalıştırılabilir kodunu değiştirmemektedir. Yorum ve boşluk ekleme, silme, değiştirme; kod hizalama veya bir metodun sınıfın içerisinde yerinin değiştirilmesi yapısal olmayan değişimlere örnek olarak verilebilir ve bu tür değişimler değişim maliyeti hesaplanırken ihmal edilmiştir. Nesneye dayalı yazılımların temel bileşeni sınıflardır. Sınıflar üzerinde ekleme, silme, taşıma ve sınıfın içinde yapılabilecek değişimleri hesaplayabilmek için; sınıfların nitelik, metot gibi sahip oldukları alt bileşenler belirlenmiştir. Dolayısı ile yapısal değişimler belirlenirken, nesneye dayalı yazılımlardaki alt bileşenler ve bunlar üzerinde yapılabilecek işlemler ile bunların aralarındaki ilişkiler dikkate alınmıştır.

İsim değiştirme ve sınıfların taşınması işlemleri yapısal değişim olarak değerlendirilmiştir. Değişim analizinde bu operasyonların belirlenmesi hesaplamaların sağlıklı yapılabilmesi için önem taşımaktadır. Bu işlemlerin belirlenemediği durumda; bu işlemlerin yapıldığı yazılım birimlerinin operasyondan önceki hali silinmiş, yeni hali eklenmiş gibi bir etkiye sahip olmaktadır.

Farklı türdeki yapısal değişimler yazılım tasarımı üzerinde farklı etkiye sahiptir. Örneğin bir sınıfın metodu içerisindeki yerel değişimler programın diğer kısımlarını etkilemezken, metodun parametre listesindeki bir değişim farklı kısımları etkileyebilir. Farklı değişim türlerinin ayırt edilebilmesi için değişimlere, yazılım tasarımına etkilerine göre farklı ağırlıklar verebiliriz.

Yapısal değişimlerin kısaltmaları, açıklamaları, ağırlıkları ve çalışmada genel olarak kullanılan ağırlıklar (KA) Tablo 1'de verilmiştir.

Tablo 1: Yapısal Değişimler

Değişim Türü	Açıklama	Ağırlık	KA
MA	Bir sınıfa bir metot eklenmesi sabiti	W_{MA}	0
MD	Bir sınıftan bir metodun silinmesi sabiti	W_{MD}	0
FA	Bir sınıfa bir nitelik eklenmesi	W_{FA}	1
FD	Bir sınıftan bir niteliğin silinmesi	W_{FD}	1
CA	Bir pakete bir sınıfın eklenmesi sabiti	W_{CA}	0
CD	Bir paketten bir sınıfın silinmesi sabiti	W_{CD}	0
R	Sınıf veya metodun isminin değiştirilmesi	W_{R}	1
M	Bir sınıfın başka bir pakete taşınması	W_{M}	1
MC	Görünürlük değişimi (public/private/protected)	W_{MC}	1
FTC	Nitelik tipinin değiştirilmesi	W_{FTC}	1
SCC	Sınıfın türediği sınıfların değişimi	W_{SCC}	1
IIC	Gerçeklenen arayüzün değiştirilmesi	W_{IIC}	1
MLC	Metodun yerel değişiklikleri sabiti	W_{MLC}	0
MRTC	Metodun dönüş tipinin değişmesi	W_{MRTC}	1
MPC	Metodun parametre değişimi	W_{MPC}	1
MLOCC	Metodun yerel operasyon ve kontrol değişiklikleri	W_{MLOCC}	1
MAAF	Metoda nitelik erişimi eklenmesi	W_{MAAF}	1
MDAF	Metoddan nitelik erişimi silinmesi	W_{MDAF}	1
MCAF	Metodun eriştiği niteliğin değiştirilmesi	W_{MCAF}	1
MACM	Metoda metot çağrısı eklenmesi	W_{MACM}	1
MDCM	Metoddan metot çağrısı silinmesi	W_{MDCM}	1
MCCM	Metodun çağırdığı metodun değiştirilmesi	W_{MCCM}	1

3.2. Değişimlerin Hesaplanması

Burada tanımlanan değişim işlemleri ve onlara verilen ağırlıklar kullanılarak, nesneye dayalı yazılımların alt bileşenlerinin değişim maliyeti hesaplanır. Bu maliyetler: Nitelik ekleme maliyeti (FAC), nitelik silme maliyeti (FDC), nitelik değişim maliyeti (FCC); metot ekleme maliyeti

(MAC), metot silme maliyeti (MDC), metot değiştirme maliyeti (MCC), sınıf ekleme maliyeti (CAC), sınıf silme maliyeti (CDC), sınıf değiştime maliyeti (CCC) olarak tanımlanmıştır. Değişim maliyetlerinin hesaplanması ile ilgili geliştirilen denklemler aşağıda verilmiştir.

$$\begin{aligned}
\mathrm{FDC} &= W_{\mathrm{FD}} \\
\mathrm{FAC} &= W_{\mathrm{FA}} \\
\mathrm{FCC} &= W_{\mathrm{MC}} + W_{\mathrm{FTC}} \\
\mathrm{MDC} &= W_{\mathrm{MD}} + \sum W_{\mathrm{MDAF}} + \sum W_{\mathrm{MDCM}} \\
\mathrm{MAC} &= W_{\mathrm{MA}} + \sum W_{\mathrm{MAAF}} + \sum W_{\mathrm{MACM}} \\
\mathrm{MCC} &= W_{\mathrm{MLC}} + W_{\mathrm{MLOCC}} + W_{\mathrm{R}} + W_{\mathrm{MC}} + W_{\mathrm{MRTC}} + W_{\mathrm{MPC}} \\
&\quad + \sum W_{\mathrm{MAAF}} + \sum W_{\mathrm{MDAF}} + \sum W_{\mathrm{MCAF}} \\
&\quad + \sum W_{\mathrm{MACM}} + \sum W_{\mathrm{MDCM}} + \sum W_{\mathrm{MCCM}} \\
\mathrm{CDC} &= W_{\mathrm{CD}} + \sum \mathrm{FDC} + \sum \mathrm{MDC} \\
\mathrm{CAC} &= W_{\mathrm{CA}} + \sum \mathrm{FAC} + \sum \mathrm{MAC} \\
\mathrm{CCC} &= \sum \mathrm{FAC} + \sum \mathrm{FDC} + \sum \mathrm{FCC} + \sum \mathrm{MAC} \\
&\quad + \sum \mathrm{MDC} + \sum \mathrm{MCC} + W_{\mathrm{R}} + W_{\mathrm{M}} + W_{\mathrm{MC}} + W_{\mathrm{SCC}} + W_{\mathrm{IIC}}
\end{aligned}$$

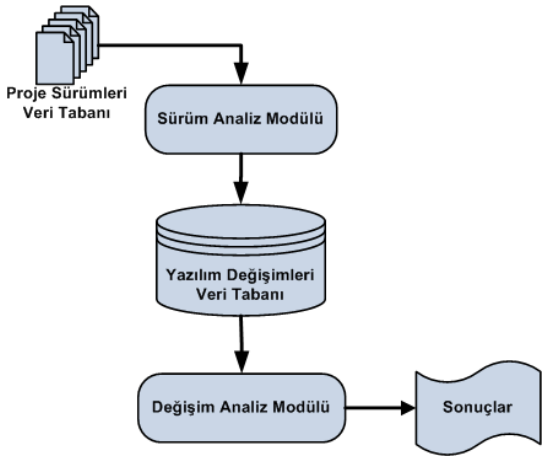
Hesaplamalar sırasında, değişim ağırlıkları sadece ilgili değişim varsa ve hesaplamaya katılması seçilmişse hesaplamaya katılmıştır. Bu çalışmada değişim maliyeti hesaplanırken yapısal değişim ağırlıkları MLC, MA, MD, CA ve CD sabitleri hariç olmak üzere diğerlerine "1" verilmiştir. Metot ve sınıfların hesaplanmasında sabitlere değer atayarak sadece onlar kullanılabildiği gibi, metot ve sınıfların sahip oldukları veya eriştikleri nitelik ve metotların ağırlıkları toplamı da dikkate alınabilir. İstenirse metot ya da sınıf ekleme, silme işlemlerine ekstra ağırlık vermek için bu sabitlere sıfırdan farklı ağırlık vererek de kullanılabilir. Değişim maliyeti hesaplama, bu şekilde esnek yapıda tutulmuştur. Metot ekleme, silme ve yerel değişim için sabitler kullanılması yerine metodun eriştiği nitelik ve çağırdığı metotların ağırlıkları ile yerel değişimler için kontrol, döngü ve metot içi diğer değişikliklere verilen diğer ağırlıkların toplamı kullanılmıştır. Benzeri şekilde sınıflar için de ekleme, silme ve değişim maliyetleri hesaplanırken sahip oldukları nitelik ve metotların değişim maliyetleri toplamı hesaba katılmıştır. O nedenle burada kullanılan sabitlerin değerleri metot ve sınıflar için "0" olarak seçilmiştir. Değişim zorluğunu hesaba katmak için metot değişim maliyetleri hesaplanırken metot ekleme, silme veya yerel değişimine sabit bir katsayı vermek yerine, erişilen nitelikler ve çağrılan metotların durumu dikkate alınmıştır. Benzer şekilde sınıf silme maliyeti hesaplanırken, sabit bir katsayı yerine sınıf içerisindeki niteliklerin ve metotların değişim maliyetlerinin toplamı kullanılmıştır. Sınıfların ilk büyüklüklerine, nitelik ve metot maliyetleri toplamı, olan bağımlılığı ortadan kaldırmak için sınıf ekleme maliyeti ihmal edilmiştir. Aksi halde değişim maliyetlerinin hesaplanması sınıfların ilk sahip oldukları nitelik ve metot sayısına bağlı olacaktır.

Bu denklemler iki sürüm arasındaki değişim maliyetlerinin hesaplanması içindir. Yazılımın ardışıl sürüm serisi analiz edilirken, değişim maliyetleri birbirini takip eden ardışıl sürüm çiftleri arasındaki değişim maliyetlerinin kümülatif toplamı olarak hesaplanmaktadır. Değişim maliyet analizi sonrası sınıfların değişim maliyetlerine göre sıralı bir liste oluşturulmuştur.

3.3. Sistem Mimarisi

Sistem genel olarak, yazılımın ardışıl sürümlerinden kaynak kod analiz eden ve tersine mühendislik ile yapısal değişimleri çıkaran; verilen hesaplama yöntemlerine ve değişimlere verilen ağırlıkları dikkate alarak değişim maliyetlerini çıkaran alt modüllerden oluşmaktadır. Sistem mimarisi Şekil 1'de gösterilmiştir.

Sürüm analiz modülü ardışıl sürümlerindeki farkları çıkararak değişim veritabanına kayıt etmektedir. Burada yazılımdaki tüm yapısal değişimler çıkarılmakta ve kaydedilmektedir. Değişim analizi modülü, yazılımın yaşam döngüsündeki tüm farksal değişimleri, kullanıcı tarafından verilen ağırlıklar ile istenen hesaplama bilgilerini kullanarak istenen ölçüm sonuçların elde edilmesini sağlamaktadır.



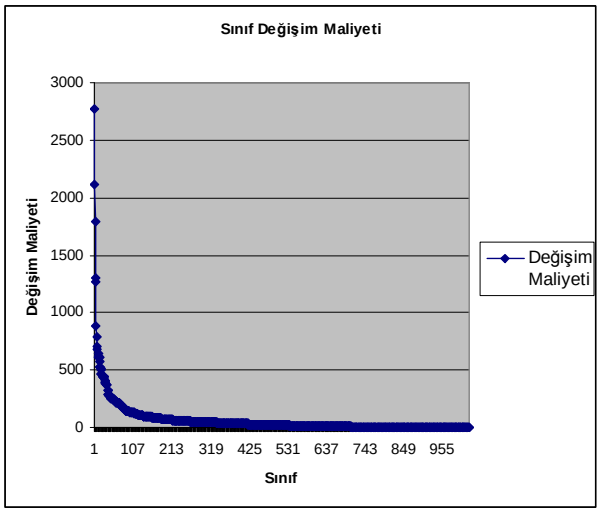
Şekil 1: Sistem mimarisi.

Değişimlerin analizi kaynak kod üzerinden, soyut sentaks ağaçları (AST) kullanılarak tersine mühendislik yöntemi ile elde edilen yazılım modelleri üzerinden yapılmıştır. Öncelikli olarak, kod analizi için kendi laboratuarımızda geliştirilen E-Quality API'si [11] kullanılarak, yazılımların her bir sürümü için kaynak kod soyut sentaks ağacı ve yazılım modeli çıkarılmıştır; daha sonra ardışıl yazılım modelleri incelenerek aralarındaki yapısal farklar elde edilmiştir. Değişimlere verilen ağırlıklar ve kullanılacak denklemler göz önünde bulundurularak, sınıfların değişim maliyetleri hesaplanmıştır.

4. Deneysel Çalışma

Deneysel çalışmada açık kaynak kodlu JFreeChart yazılımının ardışıl sürümleri incelenmiştir. JFreeChart açık kaynak kodlu güçlü bir çizim kütüphanesidir [12]. Uzun dönemli bir projedir ve incelemek için yeteri kadar sürümü vardır. Bu çalışmada 07.06.2002 ile 10.09.2004 tarihleri arasında yayınlanmış 0.9.0 ile 0.9.21 arasındaki ardışıl yirmi iki sürümü incelenmiştir.

Projenin ardışıl yirmi iki sürümü analiz edilerek sınıfların kümülatif değişim maliyetleri çıkarılmıştır. Sınıfların projenin gelişim sürecindeki toplam değişim maliyetleri büyükten küçüğe sıralı olarak aşağıdaki Şekil 2'de verilmiştir.



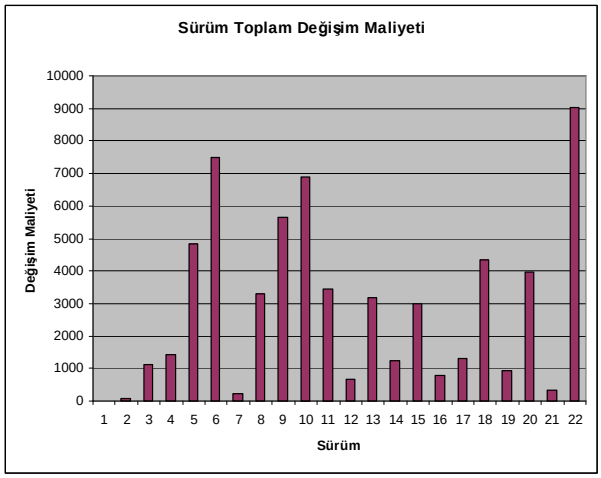
Şekil 2: Sınıfların değişim maliyeti.

Sınıfların değişim maliyetleri incelendiğinde en üstteski 19 sınıfın maliyetinin 500'den yüksek olduğu görülmektedir. *org.jfree* paketi altında yer alan bu sınıfların değişim maliyetleri (DM) ve kaç sürümde değiştiğini gösteren değişim sayısı (DS) Tablo 2'de verilmiştir.
Bu sınıflar bu projenin birçok sürümünde değişime uğramış ve üzerlerinde çok işlem yapılmış sınıflarıdır. Bu nedenle yeniden yapılandırma sürecinde bu sınıflara öncelik verilmelidir. Bu sınıflardan en üstteki 5 sınıfın değişim miktarlarının diğerlerine göre oldukça yüksek olduğu görülmektedir.

Tablo 2: JFreeChart Sınıfların Değişim Maliyeti ve Sayısı

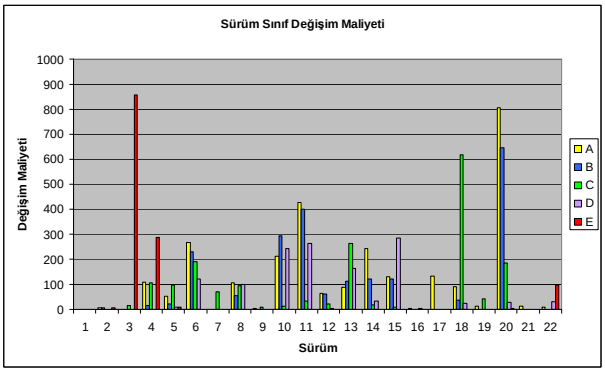
Sınıf	DM	DS
chart.plot.XYPlot	2778	20
chart.plot.CategoryPlot	2119	16
chart.plot.PiePlot	1787	17
chart.renderer.AbstractRenderer	1303	16
chart.demo.JFreeChartDemo	1266	10
chart.axis.DateAxis	886	19
chart.axis.CategoryAxis	788	15
chart.demo.JFreeChartDemoBase	704	16
chart.axis.NumberAxis	684	16
chart.axis.ValueAxis	640	18
chart.plot.MeterPlot	622	14
chart.renderer.category.AbstractCategoryItemRenderer	614	18
data.time.TimeSeriesCollection	613	13
chart.StandardLegend	608	17
chart.renderer.category.BarRenderer	579	17
chart.renderer.xy.AbstractXYItemRenderer;	520	18
chart.JFreeChart;	520	18
chart.plot.ThermometerPlot;	508	15
chart.renderer.HorizontalBarRenderer3D;	501	9

Projenin gelişimi boyunca projedeki sınıfların toplam değişimini ifade etmek için ardışıl iki sürüm arasındaki toplam değişim maliyetini gösteren grafik Şekil 3'de verilmiştir. [0.9.0-0.9.21] arasındaki ardışıl sürümler [1-22] aralığında gösterilmiştir. Bir sürümden diğerine geçişte ne kadarlık bir değişim yapıldığı bu grafikten görülmektedir. Ayrıca projede hangi sürümlerde en çok değişim yapıldığı bilgisine de ulaşılabilmektedir. Bu sayede projenin kullanılacak sürümleri arasında tercih yapılırken, sürümler arasında ne kadarlık değişimin yapıldığı nicel olarak ifade edilebilir ve bu dikkate alınarak karar verilebilir.



Şekil 3: Sürüm - sınıfların toplam değişim maliyeti.

İncelenen projenin sürümleri arasında en çok değişim miktarına sahip olan 5 sınıf için her bir sürümde bir öncekine göre değişim maliyetini gösteren grafik Şekil 4'de verilmiştir.



Şekil 4: Sürüm - sınıfların değişim maliyeti.

Tablo 2'de en üstte yer alan ve değişim maliyetleri 1200'den yüksek olan bu 5 sınıfın gösterimi için harf kullanılmıştır. A *XYPlot*, B *CategoryPlot*, C *PiePlot*, D *AbstractRenderer* ve E *JFreeChartDemo* sınıflarını göstermektedir. Buradan *JFreeChartDemo* sınıfının 0.9.2 ve 0.9.3 sürümüne geçişte çok değiştiği ancak ilerleye sürümlerde kararlı hale geldiğini görmekteyiz. Bu sürümler arası geçişte bu sınıf yeniden yapılandırılmıştır. *XYPlot*, *CategoryPlot* ve *PiePlot* sınıfları son sürümlerde de çok değişime uğramıştır; ayrıca birçok sürümünde bu sınıflar değiştirilmiştir. Bu sınıfların kararsız olduğunu ve gelen değişim isteklerinden çok etkilendirildiklerini dolayısı ile sınama, gözden geçirme ve yeniden yapılandırma işlemlerinde bu sınıflara öncelik verilmesi gerektiği söylenebilir. Bu yöntem sayesinde projedeki sınıfların tarihsel olarak değişim miktarları görülebilmekte ve sınıfların anlaşılırlığının artırılması

sağlanmaktadır. Bir sınıfın çok değiştiği sürüm geçisi incelenerek neden bu şekilde bir değişikliğe gidildiği incelenebilir. Bu sayede yazılımlardaki önemli sınıflar ve önemli değişikliklerin yapıldığı zamanlar daha kolay takip edilebilir.

5. Sonuçlar

Bu çalışmada, nesneye dayalı yazılımlarda sınıfların değişimleri yapısal olarak incelenmiş ve değişim miktarını ifade etmek için değişim maliyeti tanımlanmıştır. Nesneye dayalı yazılımlardaki olası yapısal değişimler sınıflandırılmış ve değişimlere etkilerine göre farklı ağırlık verilmesini ve önerilen denklemleri kullanarak bu maliyetlerin hesaplanmasına olanak sağlayan bir program geliştirilmiştir.

Yazılım sürümleri arasında ne kadar değişim yapıldığı, en çok hangi sınıflar değişime uğradığı, hangi sınıfların kararlı ve tekrar kullanılabilir, hangi kısımların kararsız ve çok değişen olduğu bilgilerine ulaşılabilir. Bununla birlikte değişim maliyeti çok yüksek olan ve yazılımın son sürümlerine yakın kısımlarda da değişime uğrayan sınıflar tehlikeli ve kritik sınıflar olarak ifade edilebilir. Bu sınıflara sınama ve gözden geçirme işlemlerinde öncelik verilmelidir. Yazılımın gelişiminin anlaşılması, hangi sınıfların hangi sürümlerde değişime uğradığı bilgisine ulaşılabilmektedir. Bu sayede geliştiriciler yazılımdaki temel sınıfları görme ve onları gelişimini inceleme sahibi olurlar. Sürümler arası geçişte maliyetlerin ölçülmesi, bir önceki sürüme göre ne kadar iş çıkarıldığının bir ölçüsü olarak kullanılabilir.

İleriki çalışmalarda yazılımların ilk sürümlerindeki değişim miktarının etkisini azaltmak, son sürümlere doğru yapılan değişimlerin etkisini artırmak için bir yaşlandırma yöntemi geliştirilmesi planlanmaktadır. Ayrıca yazılımlardaki değişimlerin niteliklerinin kullanıcıya sunulması için bir çatı geliştirilmesi düşünülmektedir.

6. Kaynakça

[1] Eski, S., Buzluca, F., "An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes," *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Berlin, Almanya, 2011, sayfa 566-571.

[2] Eski, S., Buzluca, F., Nesneye Dayalı Yazılımlarda Sınama ve Bakım Öncelikli Sınıfların Belirlenmesi, Yüksek Lisans Tezi, 2011, İTÜ.

[3] Chaumun, M., Kabaili, H., Keller, R., ve Lustman, F., Change Impact Model for Changeability Assessment in Object-Oriented Software Systems. *Science of Computer Programming*, 2002, 45(2–3):155–174.

[4] Tsantalis, N., Chatzigeorgiou, A., ve Stephanides, G., Predicting the Probability of Change in Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 2005, 31(7):601–614.

[5] Sharafat, A. R., ve Tahvildari, L., A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems. *IEEE CSMR'07*, 2007.

[6] Xia, F., A Change Impact Dependency Measure for Predicting the Maintainability of Source Code. *In Proceedings of the Annual International Computer Software and Applications Conference (COMPSAC)*, 2004, sayı 2, sayfa 23–24.

[7] Hassan, A. E., Holt, R. C., Predicting Change Propagation in Software Systems. *In Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 2004, sayfa 284–293.

[8] Lee, M., Offutt, J., Alexander, R. T., Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software. *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, 2000, s.61, Temmuz 30-Ağustos 03.

[9] Lee, Y., Yang, J., Chang, K. H., Metrics and Evolution in Open Source Software, *Seventh International Conference on Quality Software, QSIC 2007*, 2007.

[10] Sato, D., Goldman, A., ve Kon, F., Tracking the Evolution of Object-Oriented Quality Metrics on Agile Projects. *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and Extreme Programming*. Haziran 18-22, 2007 - Como, İtalya.

[11] Erdemir, U., Tekin, U., Buzluca, F., EQuality: A Graph Based Object Oriented Software Quality Visualization Tool, *6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2011)*, Eylül 29-30, 2011 - Williamsburg, Virginia, ABD.

[12] JFreeChart: http://www.jfree.org/jfreechart