

**T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ**

**UYGULAMA PROGRAMLAMA
ARAYÜZLERİNİN KULLANILABİLİRLİĞİ**

**Burcu ARDIÇ
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI**

**GEBZE
2010**

**T.C.
GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ
MÜHENDİSLİK VE FEN BİLİMLERİ
ENSTİTÜSÜ**

**UYGULAMA PROGRAMLAMA
ARAYÜZLERİNİN KULLANILABİLİRLİĞİ**

**Burcu ARDIÇ
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI**

**TEZ DANIŞMANI
Yrd.Doç.Dr. Mehmet GÖKTÜRK**

**GEBZE
2010**



YÜKSEK LİSANS TEZİ JÜRİ ONAY SAYFASI

G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 07/06/2010 tarih ve 2010/25 sayılı kararıyla oluşturulan jüri tarafından 28.06.2010 tarihinde tez savunma sınavı yapılan Burcu ARDIÇ'ın tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) :

Yrd. Doç. Dr. Mehmet GÖKTÜRK

ÜYE

:

Yrd. Doç. Dr. İlyas KANDEMİR

ÜYE

:

Dr. Hidayet TAKÇI

ONAY

G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../20... tarih ve/..... sayılı kararı.

İMZA/MÜHÜR

ÖZET

TEZ BAŞLIĞI : UYGULAMA PROGRAMLAMA ARAYÜZLERİNİN KULLANILABİLİRLİĞİ

TEZ YAZARI : BURCU ARDIÇ

Günümüzde yazılım geliştirme, uygulama programlama arayüzlerinin (UPA) kullanımından bağımsız düşünülemez. Yazılım geliştiricilerin UPA'ları kullanırken yaşadıkları problemler, geliştirdikleri yazılım ürünlerinin kalitesini doğrudan ve olumsuz etkilemektedir. Bu sebeple, kullanılabilirlik göz önünde tutularak geliştirilmiş uygulama programla arayüzlerine ihtiyaç vardır. Ne var ki, kullanılabilirlik odaklı kalite çalışmalarının sayısı oldukça azdır. Mevcut insan-bilgisayar etkileşimi tekniklerinin UPA'lara uyarlanarak, bu arayüzlerin kullanıcı merkezli bir şekilde tasarlanmasını sağlayacak hedeflerin belirlenmesi gerekmektedir. Bununla birlikte, “kullanıcı” olarak “yazılım geliştiricinin” merkeze koyularak ihtiyaçlarının anlaşılması gerekmektedir. Bu çalışmada, geniş bir literatür taraması yapılarak UPA'ların kullanılabilirliğini etkileyen faktörler ve UPA tasarımcılarına yönelik pratikler bir araya getirilmiş ve açıklanmıştır. Ayrıca kullanıcı grafik arayüzlerinin kullanılabilirlik değerlendirilmesinde faydalanan geleneksel yöntemler, uygulama programlama arayüzlerine uyarlanarak bir değerlendirme çerçevesi ortaya konmuştur. Bu değerlendirme çerçevesine göre, UPA kullanılabilirliğini etkileyen faktörlerden üçü ile değerlendirme yapılmış ve sonuçlar raporlanmıştır.

SUMMARY

THESIS TITLE : USABILITY OF APPLICATION PROGRAMMING INTERFACES

THESIS AUTHOR : BURCU ARDIÇ

Recently, software engineering came to a point that, development cannot even be thought of, without use of application programming interfaces (API). Developers often face with difficulties using these APIs, and these problems are directly and negatively affecting the quality of software products and the success of software projects. Therefore, developers require application programming interfaces, which are designed by taking usability into account. Nevertheless, the studies focusing on usability aspect are far from sufficient. Existing human-computer interaction techniques should be adapted to APIs, and the goals for a user-centered design should be determined, for “the developer” being “the end user” in the API case. During this study, a thorough literature search was made, in order to gather factors that affect API usability and some practical tips to guide API designers. Moreover, the primary evaluation techniques to review the usability of graphical user interfaces are adapted to API usability, and an evaluation framework for APIs is presented. In addition, an evaluation practice is done to test both the framework, and a subset of the factors affecting the API usability.

TEŞEKKÜR

Yüksek lisans eğitimim ile tez çalışmam boyunca bana destek veren ve teşvik eden tez danışmanım Gebze Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü Öğretim Üyesi Sayın Yrd. Doç. Dr. Mehmet GÖKTÜRK’e,

Deneyletime katılarak araştırmalarımın doğru yöne akmasını sağlayan yazılım geliştiricilere,

Bilime olan tutkularıyla bana yol gösteren Ankara Fen Lisesi’nin değerli öğretmenlerine,

Hayatta ufkumu hep açan sevgili babam Gazi’ye, her isteğimi olanca gayretiyle destekleyen sevgili annem Hatice’ye ve üretkenlikleriyle bana örnek olan ablalarım Ayça ve Başak’a,

Tez çalışmam boyunca parlak fikirlerini, sabrını ve desteğini benden asla esirgemeyen sevgili arkadaşım Serdar SERPEN’e en içten dileklerle teşekkür ederim.

İÇİNDEKİLER DİZİNİ

ÖZET	v
SUMMARY	vi
TEŞEKKÜR	vii
İÇİNDEKİLER DİZİNİ	viii
ŞEKİLLER DİZİNİ	xi
ÇİZELGELER DİZİNİ	xii
SİMGELER VE KISALTMALAR DİZİNİ	xiii
1. GİRİŞ	1
2. KULLANILABİLİRLİK	4
2.1. İnsan-Bilgisayar Etkileşimi	4
2.2. Kullanılabilirlik Mühendisliği Yaşam Döngüsü	6
2.3. Kullanılabilirlik Değerlendirme Yöntemleri	6
2.3.1. Sezgisel Değerlendirme	7
2.3.2. Üzerinden Geçme	11
2.3.3. Kullanıcı Testleri	12
3. KULLANILABİLİR UYGULAMA PROGRAMLAMA ARAYÜZLERİ	14
3.1. UPA Kullanılabilirlik Hedefleri	14
3.1.1. Kullanıcının İhtiyaç Duyduğu Fonksiyonları Sağlaması	14
3.1.2. Kolay Öğrenilmesi	14
3.1.3. Kolay Hatırlanması	14
3.1.4. Kodun Kolay Yazılabilmesi	15
3.1.5. Kullanıcı Kodunun Anlaşılabilirliği	15
3.1.6. Yanlış Kullanımın Zorluğu	15
3.2. UPA Kullanılabilirliğine Etki Eden Faktörler	15
3.2.1. Karmaşıklık	15
3.2.2. İsimlendirme	16
3.2.3. Dokümantasyon	16
3.2.4. Nesne Yaratılması	17
3.2.5. Metot Parametreleri ve Dönüş Tipi	18

3.2.6. Metot Yerleşimi	18
3.2.7. Hata ve Kural Dışı Durum İşleme	19
3.2.8. Kademeli Sınama	20
3.2.9. Çağrı Sonrası Anlaşılabilirlik	21
3.2.10. Kaynak Kodunun Anlaşılabilirliği	22
3.2.11. UPA’larda Değişiklik	22
3.3. UPA’larda Kullanılabilirlik Değerlendirmesi	23
3.3.1. Sezgisel Değerlendirme	23
3.3.2. Kullanıcı Deneyleri	24
4. HİPOTEZ	27
4.1. İsimlendirmenin Kullanılabilirliğe Etkisi	27
4.2. Parametrelili Yaratıcı Metodun Kullanılabilirliğe Etkisi	27
4.3. Metot Yerleşiminin Kullanılabilirliğe Etkisi	27
5. YÖNTEM	29
5.1. UPA Seçimi	29
5.1.1. Görevlerin Seçimi	29
5.1.2. Java 2D API’nin Sezgisel Değerlendirmesi	31
5.2. Karşılaştırılacak UPA’ların Üretilmesi	33
5.2.1. Kötü İsimlendirme	33
5.2.2. Parametresiz Yaratıcı Metot	34
5.2.3. Ana Sınıf Üzerinde Metot Yerleşimi	35
5.3. Sıralama	35
6. UPA DEĞERLENDİRMESİ	38
6.1. Örneklem	38
6.2. Deney Ortamı	39
6.3. Deney Aşamaları	40
6.3.1. Giriş Brifingi	40
6.4. Ölçüm	41
7. SONUÇLAR	42
7.1. Yalancı Kod Aşaması Sonuçları	42
7.2. Görev Gerçekleme Aşaması Sonuçları	42
7.3. Anket Sonuçları	46

8. SINIRLAMALAR	47
8.1. Deney Kapsamı	47
8.2. Sıralama Etkisi	47
8.3. Dil	48
8.4. Deney Ortamı	48
9. TARTIŞMA	49
10. GELECEK ÇALIŞMALAR	52
KAYNAKLAR	54
ÖZGEÇMİŞ	56
EKLER	
10.1. Ek-1: Deney Kılavuzu	
10.2. Ek-2: Anket	

ŞEKİLLER DİZİNİ

<u>Sekil</u>	<u>Sayfa</u>
2.1. İnsan-Bilgisayar Etkileşimi - ACM SIGCHI Curricula for Human-Computer Interaction'dan uyarlanmıştır [Hewett, 1992]	5
2.2. Kullanıcı Odaklı Tasarım	6
2.3. Sistem Durumu	8
2.4. Tutarlı ve tutarsız program kontrolleri	9
2.5. Kullanıcıya seçeneklerin sunulduğu bir ekran görüntüsü	10
3.1. Örnek. Parametrelili yaratıcı metot kullanımı	17
3.2. Örnek. Parametrelili yaratıcı metot kullanımı	18
3.3. Kademeli Sınama Örneği - I	20
3.4. Kademeli Sınama Örneği - II	21
5.1. Kullanıcı Deneylerinde Gerçeklenen Görevler	30
5.2. Parametrelili yaratıcı metot ile nesne yaratımı	31
5.3. Parametrelili yaratıcı metot ile nesne yaratımı	31
5.4 Ana sınıf üzerinden metot çağırısı	32
5.5 Dolaylı sınıf üzerinden metot çağırısı	32
5.6. BN API – Basitleştirilmiş Sınıf Diyagramı	33
5.7. PC API – Basitleştirilmiş Sınıf Diyagramı	34
5.8. MP API – Basitleştirilmiş Sınıf Diyagramı	35
5.9. Karşılaştırılacak UPA'lar	36
7.1. Tamamlanan görev sayısı ortalamaları	43
7.2. Tüm görevleri tamamlayan kullanıcıların tamamlama süresi ortalamaları	43
7.3. Kullanıcıların sordukları soruların ortalamaları	44
7.4. Kullanıcıların denemelerindeki hata ortalamaları	44
7.5. Kullanıcıların UPA'ları sıralamasından elde edilen sıralama ortalaması	45
7.6. Deney sonuçları özeti	46

ÇİZELGELER DİZİNİ

<u>Çizelge</u>	<u>Sayfa</u>
5.1. Her kullanıcı için UPA sıralaması	36
7.1. Ortalama Deney Sonuçları	42

SİMGELER VE KISALTMALAR DİZİNİ

UPA	: Uygulama Programlama Arayüzü
API	: Application Programming Interface
IBE	: İnsan Bilgisayar Etkileşimi
PPIG	: Programming Psychology Interest Group
CD	: Cognitive Dimensions
Java SE	: Java Standard Edition
JDK	: Java Development Kit
AWT	: Abstract Windowing Toolkit
BN	: Bad Naming
MP	: Method Placement
PC	: Parameterless Constructor
TÜBİTAK	: Türkiye Bilim ve Teknoloji Araştırma Kurumu
UEKAE	: Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü

1. GİRİŞ

Gerekli işlevsellik ve güvenilirliğin sağlanması, zaman ve bütçe planlaması dışında, yazılım tasarımcılarının dikkat etmeleri gereken bir diğer nokta da insan-bilgisayar etkileşimi (İBE) hedeflerinin yerine getirilmesidir. Kullanılabilirlik, kalitenin vazgeçilmez bir parçasıdır. Gündelik yaşamda kullanılan her üründe olduğu gibi, insanın bilgisayarla etkileşimi sırasında faydalandığı tüm programların, sistemlerin ve arayüzlerin de kullanılabilirliği tercih nedenlerinin başında gelmektedir. İşlevsellik, kullanılabilir olmadığı sürece amacına hizmet etmemektedir.

Son yıllarda, yazılım mühendisliği, insan bilgisayar etkileşimi çalışmalarından faydalananak kullanılabilirlik açısından kayda değer bir yol kat etmiştir. Bu çalışmalar özellikle kullanıcı grafik arayüzleri üzerinde yoğunlaşmıştır. Ancak, insan-bilgisayar etkileşiminin sorumluluk alanı bu kadarla sınırlı değildir. Yazılım projelerinde maliyet kaygısı ile yeniden kullanım talebinin artması, yazılım geliştiricilerini üçüncü parti bileşenleri daha fazla kullanmaya zorlamaktadır. Uygulamalarına entegre ettikleri kütüphanelerin kendilerine sunduğu arayüzler, yazılım geliştiricilerin bilgisayarla etkileşim noktaları arasına girmiştir.

Uygulama Programlama Arayüzü (UPA) [*İngilizce: Application Programming Interface – API*], işletim sisteminin, bir kütüphanenin veya bir servisin diğer programlara sağladığı fonksiyon ve sınıf kümesidir. UPA’ların tasarımı ve geliştirilmesi konusunda literatürde çok sayıda yazılım mühendisliği çalışması olmasına rağmen, bunların çok azı bu arayüzlerin kullanılabilirliğine işaret etmektedir. Diğer taraftan, kullanılabilirlik üzerine çok sayıda insan-bilgisayar etkileşimi çalışması olmasına rağmen, bunların yalnızca küçük bir kısmı UPA’ları kullanarak yazılım geliştirmeye çalışan geliştiricilerin de aynı zamanda birer “kullanıcı” olduğunu hesaba katmıştır. UPA’larda bulunan kullanılabilirlik problemleri, çok sayıda yazılım projesini olumsuz olarak etkilemeye devam etmektedir.

Programlama psikolojisi, 1970 yılından beri üzerinde çalışılan bir alandır. İnsanın yazılım geliştirme becerileri bilişsel psikoloji tarafından araştırılmaktadır.

1987 yılında The Psychology of Programming Interest Group (PPIG) [ppig.org], bu araştırmaları organize etmek için kurulmuştur. Buna rağmen, ancak 1996 yılında Steven Pemberton'ın "Programmers are Humans, Too" başlıklı çarpıcı konuşmasıyla yazılım geliştiriciye yönelik kullanılabilirlik fikrine dikkat çekilebilmiştir [Pemberton, 1996]. Makalede de belirtildiği gibi, programcılar da bilgisayarla etkileşim içinde olan insanlardır. Bu etkileşimin araçları ise, başta programlama dilleri ve kütüphaneler olmak üzere, yazılım geliştirme ortamları, test araçları ve editörlerdir. Yazılım geliştirme faaliyetleri de İBE çalışma alanına dâhil olmaktadır. Bu nedenle programlama dilleri ve kütüphaneler üzerinde de görev analizi, gereksinim analizi, kullanıcı testleri ve kullanıcı merkezli yinelemeli tasarım yapılmalıdır. Aslında programlama dilleri ve kütüphaneler programcılar için var olmasına rağmen, çoğumuzun aklında bunların bilgisayarlar için olduğu algısı bulunmaktadır. Belki de bu sebeple, insanlar için tasarlanmamakta, bilgisayarlar için tasarlanmaktadır. İnsan faktörü yeteri kadar göz önünde bulundurulmamaktadır [Pemberton, 1996].

Temel kullanılabilirlik prensiplerinin UPA'lara uygulanması, yazılım geliştiricilerin bilgisayarla etkileşimini daha etkili, verimli ve tatmin edici hale getirecektir. Bu amaç doğrultusunda, çalışma kapsamında yazılım geliştiricinin ihtiyaçları anlaşılmaya çalışılmıştır. Geniş bir literatür taraması yapılarak, çok çeşitli kaynaklardan UPA'ların kullanılabilirliğine etki eden faktörler toplanmıştır. Ayrıca literatürde kullanılabilirlik değerlendirilmelerinde kullanılmış olan yöntemler UPA'lara uyarlanarak bir değerlendirme çatısı oluşturulmuştur. Hem toplanan faktörlerin bir alt kümesini geçerlemek, hem de ortaya konan UPA kullanılabilirlik değerlendirmesi çatısının bir uygulamasını yapmak amacıyla, faktörlerin üçü seçilmiş ve uygun UPA'lar elde edildikten sonra önerilen deneyler yapılmıştır.

Tez çalışması on bölümden oluşmaktadır.

İkinci bölümde insan bilgisayar etkileşimi disiplininin faaliyet alanlarından kısaca bahsedilmekte ve kullanılabilirlik kavramı tarif edilmektedir. Genel kullanılabilirlik değerlendirmesi için kabul görmüş yöntemler bu bölümde sunulmaktadır.

Üçüncü bölümde, geniş bir literatür taraması sonucu elde edilen, UPA kullanılabilirliğine etki eden faktörler açıklanmaktadır. Ayrıca bu bölümde UPA kullanılabilirliğine dikkat etmek isteyen tasarımcılara ilgili başlıklar altında tavsiyeler verilmektedir. UPA kullanılabilirliğini değerlendirmek için bir çatı sunulmaktadır.

Tezin devamında, kullanılabilirliği etkileyen etmenlerden üçünün etkilerini incelemek, diğer taraftan da ortaya konan çatının bir uygulamasını yapmak amacıyla yapılan deneylerden bahsedilmektedir. Dördüncü bölümde deneyler için hipotezler, beşinci bölümde yöntem ve altıncı bölümde deneyin ayrıntıları anlatılmaktadır.

Deneylerin tamamlanmasıyla elde edilen sonuçlar sekizinci bölümde paylaşılmaktadır. Deneyler sırasında ve öncesinde fark edilen, sonuçları etkileyebilecek sınırlamalar dokuzuncu bölümde verilmektedir. Nihayet onuncu bölümde, elde edilen sonuçlar tartışılmakta ve alınan dersler paylaşılmaktadır. Onuncu bölümde ise bu konuya ilgi duyan araştırmacılar tarafından yapılabilecek çalışmalar sunulmaktadır.

2. KULLANILABİLİRLİK

Kullanılabilirlik, kullanıcıların bir sistemle etkileşimlerinin kolaylığını ifade eder. ISO 9241’de kullanılabilirlik, “*Hedef kullanıcıların gerekli görevleri etkili, verimli ve tatmin edici bir şekilde yerine getirebilmeleri*” olarak tanımlanmaktadır [ISO 9241, 1998].

Kullanılabilirlik tanımında da belirtilen insan-bilgisayar etkileşimi hedefleri şunlardır:

a) Etkili Olma (‘Effectiveness’): Kullanıcının yerine getirmesi gereken görevleri eksiksiz ve doğru bir şekilde tamamlayabilmesi için gerekli işlevselliğin sağlanmasıdır.

b) Verimlilik (‘Efficiency’): Kullanıcının eksiksiz ve doğru bir şekilde yerine getirebildiği görevlerin, harcadığı fiziksel ve zihinsel kaynaklara (zaman veya fiziksel uğraş gibi) oranıdır.

c) Kullanıcı Tatmini (‘User Satisfaction’): Kullanımın rahat ve kabul edilebilir oluşudur. Kullanıcının bilgisayarla etkileşimi süresince yaşadığı sinir bozukluğu ile ters orantılıdır.

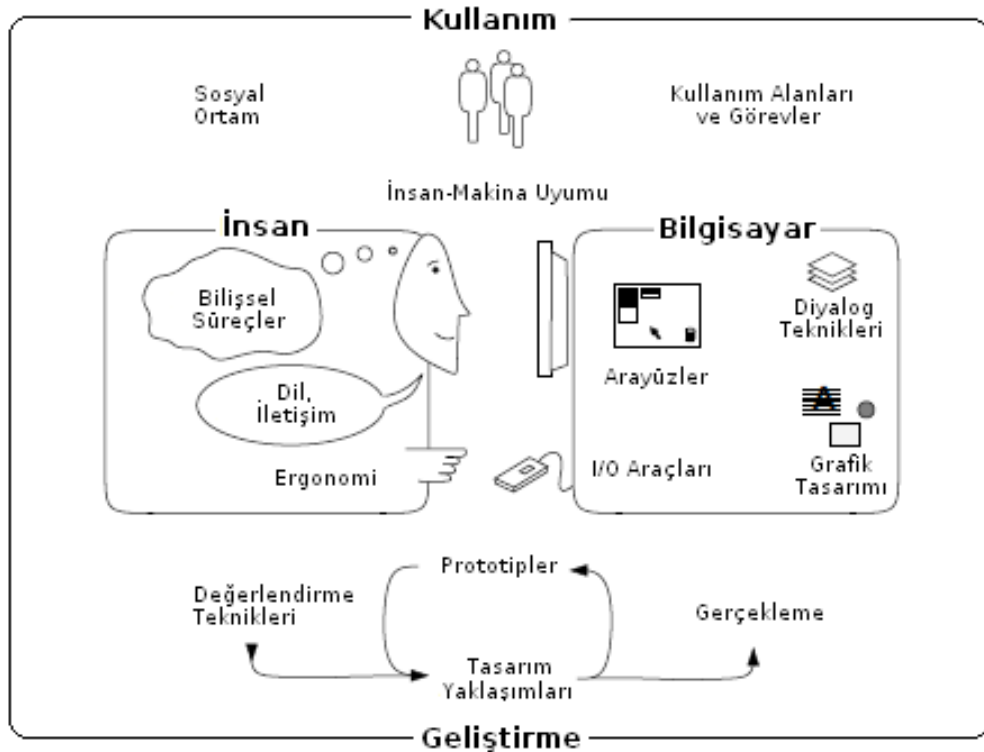
2.1. İnsan-Bilgisayar Etkileşimi

İnsan-bilgisayar etkileşimi, ACM SIGCHI Curricula for Human-Computer Interaction’da “Etkileşimli bilgisayar sistemlerinin insan kullanımı için tasarlanmaları, değerlendirmeleri ve gerçeklenmeleriyle ilgilenen bir disiplin” olarak tanımlanmaktadır [Hewett, 1992].

İnsan-bilgisayar etkileşimi, bilgisayara dayalı teknolojinin insanlar tarafından daha kolay kullanılmasını amaçlamaktadır [Preece, 1994]. İnsanları ve onların sosyal ortamlarını, teknolojiyi, insanların teknolojiye ihtiyaç duydukları alanları ve neyin kullanılabilir olup neyin olmadığını anlamaya çalışır. Bilgisayar bilimi altında

çalışılmasının yanında, bilişsel bilimler, psikoloji ve sosyolojiden de faydalanan disiplinler arası bir konudur.

Şekil 2.1’de insan bilgisayar etkileşiminin çalışma alanı özetlenmektedir. İnsanların bilgisayarla etkileşimi sosyal bir ortam içinde gerçekleşmektedir. İnsanların üzerinde olan bazı görevlerin bilgisayar ortamına taşınabilmesi, bilgisayarların kullanım alanlarının neler olabileceği ve görevlerin bilgisayarlar ile insanlar arasında nasıl paylaştırılacağı sorularını beraberinde getirmiştir. Bu soruların cevabı, ancak insanların ve bilgisayarların yeteneklerini ve sınırlamalarını bilerek verilebilir. İnsanların bilişsel yetenek ve sınırlamaları, ergonomi gereksinimleri; diğer taraftan bilgisayarların etkileşime sunabilecekleri teknik özelliklerden I.O araçları, bilgisayar grafikleri, her türlü arayüz ve diyalog teknikleri İBE araştırma alanlarına girmektedir. Araştırmalardan çıkarılan sonuçlar, ürün geliştirmeye gerek tasarım gerekse değerlendirme önerileri olarak dahil edilmektedir.

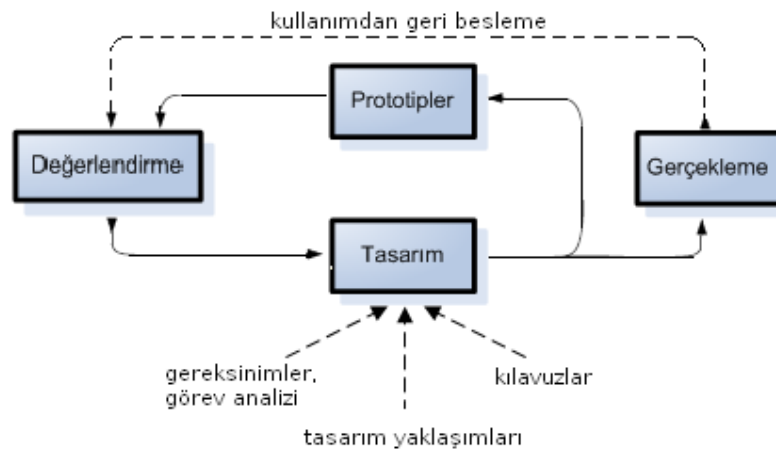


Şekil 2.1. İnsan-Bilgisayar Etkileşimi - ACM SIGCHI Curricula for Human-Computer Interaction’den uyarlanmıştır [Hewett, 1992]

2.2. Kullanılabilirlik Mühendisliği Yaşam Döngüsü

Kullanılabilirlik gereksinimleri belirlenen bir projede, bu gereksinimlerin gerçekleşip gerçekleşmediği tıpkı diğer gereksinimler gibi kontrol edilmelidir. Kullanıcı odaklı tasarım yaklaşımının proje geliştirme metodolojisine eklenmesiyle kullanılabilirlik mühendisliği yaşam döngüsü ortaya çıkmıştır.

Kullanıcı odaklı tasarım yaklaşımına göre (Bkz. Şekil 2.2), ürünün tasarımı, kullanıcının görevleri analiz edildikten ve kullanılabilirlik gereksinimleri ortaya konduktan sonra, literatürdeki yaklaşımlar ve kılavuzlardan faydalanılarak yapılır. Ortaya çıkan tasarımın değerlendirilebilmesi için belirli aralıklarla prototipler üretilir. Bu prototipler üzerinde bölüm 2.3'te anlatılan değerlendirme teknikleri ile uygun değerlendirmeler yapılarak tasarımdaki kullanılabilirlik hataları ortaya çıkarılır. Giderilen kullanılabilirlik hatalarının ardından üretilen prototipler yeniden değerlendirilerek evrimsel bir çalışma yapılmış olur. Tüm değerlendirmelere ve hata gidermelere rağmen, kullanılabilirlik hatalarının bazıları fark edilemeden kalacaktır. Nihaî düzeltmeler, ürün kullanılmaya başladıktan sonra elde edilen geri beslemelerle yapılmalıdır.



Şekil 2.2. Kullanıcı Odaklı Tasarım

2.3. Kullanılabilirlik Değerlendirme Yöntemleri

Uzmanlar ve kullanıcılar tarafından yapılan kullanılabilirlik değerlendirmeleri, kullanıcı odaklı arayüz tasarımının önemli bir bileşenidir. Sistemlerin kullanılabilirlik gereksinimlerini karşılayıp karşılamadıklarını ortaya çıkarmak için

değerlendirmeler yapılmalıdır. Farklı avantajlara ve maliyetlere sahip başlıca yöntemler aşağıda anlatılmıştır. Bir sistemi değerlendirmek için birden fazla yöntem bir arada kullanılabilir. Değerlendirmeler sonunda ortaya çıkan problemler giderilerek daha kullanılabilir bir sistem elde edilebilir.

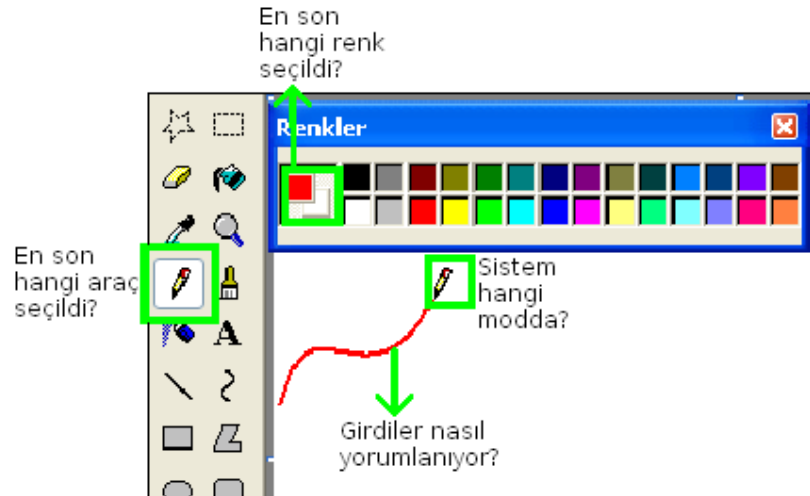
2.3.1. Sezgisel Değerlendirme

Sezgisel değerlendirme (*“heuristic evaluation”*), kullanıcı arayüzünün uzman bir değerlendirici tarafından, belirli bir kriter kümesine göre değerlendirilerek o kriterlere uyum ölçüldüğü bir yöntemdir. Değerlendirme uzman tarafından yapıldığı için “Uzman Değerlendirmesi” de denir. Sezgisel değerlendirme, ciddi kullanılabilirlik hatalarının tasarım aşamasında yakalanarak düzeltilmesine olanak tanır. Kullanıcı davranışlarının yorumlanmasını gerektirmediği ve belirli bir kriter kümesine dayandığı için kolay ve hızlıdır. Tasarım aşamasında kullanılabilirlik hatalarını teşhis edebildiği için maliyet açısından avantajlıdır. Diğer taraftan değerlendiricilerin taraflı veya yetersiz olmaları durumunda hata oluşabileceğinden kullanıcı deneyleriyle birlikte kullanılmalıdır. Bu yöntem, kullanıcı grafik arayüzleri için 10 kriter ile birlikte, kullanılabilirlik uzmanı Jacob Nielsen tarafından ortaya konmuştur [Nielsen and Molich, 1990]. Aşağıda bu kriterler açıklanmaktadır:

1. Sistem Durumu ile İlgili Geri Besleme (“Feedback”)

Sistem kullanıcıları mevcut durum ile ilgili sürekli bilgilendirmelidir. Kullanıcı, verdiği girdinin nasıl yorumlandığı ve o anda ne yapıldığı hakkında daima bilgi sahibi olmalıdır. En yaygın kullanılan örnek, sistem kullanıcıyı bekleteceği durumlarda kullanılan ilerleme çubukları ve kum saatidir.

Şekil 2.3’te sistem durumu hakkında kullanıcıya bilgi verilmesinin başarılı bir örneği görülmektedir.



Şekil 2.3. Sistem Durumu

2. Kullanıcının Dilini Konuşma (“Speak the Users’ Language”)

Bilgisayar sistemleri genellikle el ile yürütülen süreçlerin bilgisayar ortamına taşınması ile oluşturulmuşlardır. Bu nedenle, kullanılan terimler, kelimeler, ya da kavramlar, kullanıcıya tanıdık olanlardan seçilmeli, sisteme yönelik tanımlamalardan kaçınılmalıdır. Kullanılan kelimeler ya da ikonlar, gerçek hayat ile analogi kurabilmelidir. Örneğin Şekil 2.3’teki araç çubuğunda görülen kalem, fırça, silgi gibi semboller, öğrenmeyi kolaylaştırmaktadır.

3. Kullanıcı Kontrolü (“Clearly Marked Exits”)

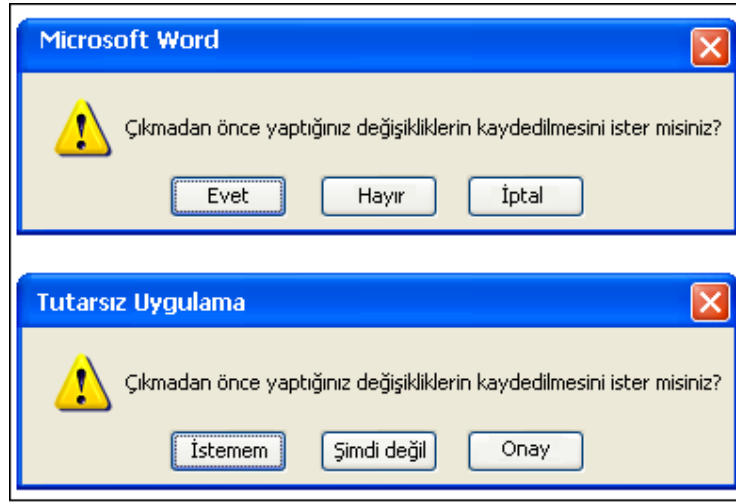
Kullanıcılar bir sistemin bazı kısımlarına yanlışlıkla girerler, ya da bazı değişiklikler yanlışlıkla yapılabilir. Bu yanlışlıklar kullanıcıların kendilerini tuzağa düşmüş hissetmesine neden olur. Sistemler kullanıcıya, mümkün olan durumlar için, kolaylıkla bulabileceği çıkışlar sunmalıdır. Örneğin kullanıcıdan karar beklenen pencerelere “İptal” tuşu koymak, “Geri Al”/“Yinele” işlevi sağlamak, uzun süren işlemlerin kesilebilmesine imkan tanımak, istenen her hangi bir anda “Kaydetmeden Çıkış” ya da yanlış yapılan ayarların üstesinden gelmek için “Varsayılan Ayarlara Dönüş” opsiyonu sunmak tavsiye edilen stratejiler arasında yer alır.

4. Tutarlılık (“Consistency”)

Bir üründe, gerek ürünün muadilleri ile, gerekse ürünün kendi içinde tutarlılık sağlanmalıdır. Farklı isimler ya da durumlar daima farklı anlamlara gelmelidir,

benzer şeyler farklı biçimlerde tarif edilmemelidir. Kullanıcılar farklı isimlerin ya da durumların aynı şey anlamına gelip gelmediğini düşünmek zorunda bırakılmamalıdır.

Dikkat edilmesi gereken bir başka husus, sistemin çalışacağı platformun standartlarına uymaktır. Örneğin işletim sisteminde, kontrollerin sırası nasılsa, uygulama da aynı sırayı kullanmalıdır.



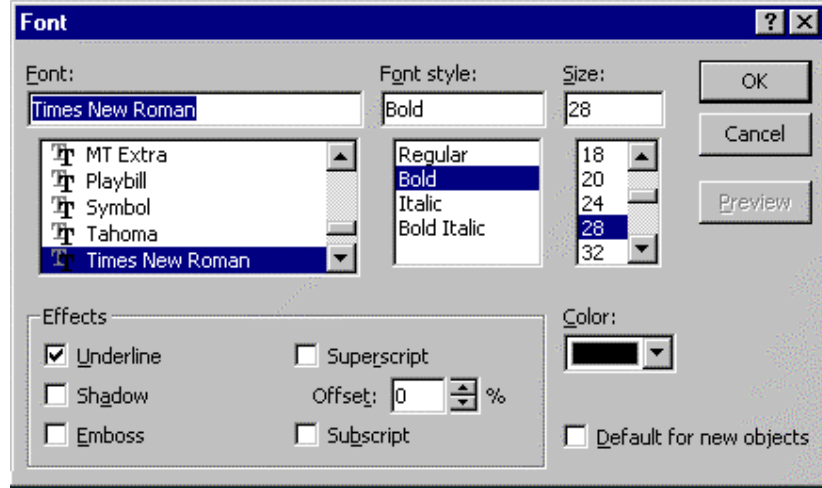
Şekil 2.4. Tutarlı ve tutarsız program kontrolleri

5. Hata Önleme (“Error Prevention”)

Hata mesajlarının anlaşılır ve yönlendirici olması gerekliliği ile birlikte, hatalar henüz oluşmadan engellenmesi daha faydalıdır. Sistemin mevcut durumda uygun olmayan işlevlerini engellemesi, dosyanın ismini yazdırmak yerine dosya gezgininden seçtirilmesi gibi pratikler örnekler arasında sayılabilir.

6. Hatırlama Yerine Tanıma (“Recognition rather than Recall”)

Kullanıcıların kısa süreli hafızaları sınırlı olduğu için, kısa süreli hafıza üzerindeki yükü azaltmak gerekmektedir. Bu amaçla, kullanıcıların girdileri ya da komutları hatırlamaları yerine, sınırlı bir kümeden seçimleri kullanılabilirliği artırır.



Şekil 2.5. Kullanıcıya seçeneklerin sunulduğu bir ekran görüntüsü

Şekil 2.5'te uygun seçenekler listelerle kullanıcıya sunulmaktadır. Böylece kullanıcılar, istedikleri yazı tipinin adını hatırlayamasa bile, listede gördüklerinde anımsayabileceklerdir.

7. Hızlandırıcıların Kullanımı (“Accelerators”)

Bir uygulama bir süre kullanıldıktan sonra, kullanıcılar tecrübe kazanırlar. Tecrübeli kullanıcıların sık yaptıkları işleri kolaylaştıracak kısayol tuşları, komut tamamlayıcılar, en son kullanılan dosyaların listesi gibi hızlandırıcılar, ürünün uzun vadeli kullanımı için vazgeçilmezdir.

8. Estetik ve Minimalist Tasarım (“Minimalist Design”)

Kullanıcıya sunulan arayüzler, ilgisiz ya da nadiren kullanılacak bilgi içermemelidir. Çünkü sunulan ilgisiz bilgiler, arayüzdeki gerekli bilgileri karmaşıktırmakta ve fark edilmelerini güçleştirmektedir. Bu nedenle, estetiğin yanı sıra minimalist tasarıma da dikkat edilmelidir.

9. Açık Hata Mesajları (“Good Error Messages”)

Uygulamalar kullanıcıları mümkün olduğunca hatadan uzak tutmalıdır, fakat yine de hataların oluşması kaçınılmazdır. Kullanıcı hatalı girdiler sağladığında, kimi sistemler hatalı olan isteği yerine getirmez, fakat kullanıcıya da herhangi bir hata mesajı sunmaz. Yanlış kayıtların oluşması engellenmiş olsa da, bu defa kullanıcı ne olup bittiğini anlamayacağı için kontrolü yitirdiğini hissedecektir. Kimi sistemler ise,

kullanıcıyı uyarmadan hatalı girdiyi düzeltirler. Fakat bu durum da yanlış düzeltmeler yapılabileceğinden risklidir. Doğru olan, kullanıcının bir hata oluştuğuna dair açıkça ve kullanıcının dilini kullanan bir mesaj verilmesi ve varsa olası düzeltmelerin kullanıcıya birer seçenek olarak sunulmasıdır.

Hata mesajları açıklayıcı olmalı ve gerçeklemeye dair detay içermemelidir. Yapıcı öneriler sunmalı ve kullanıcıyı suçlayıcı ifadelerden kaçınılmalıdır.

10. Yardım ve Dokümantasyon (“Help and Documentation”)

Sistemin dokümantasyonsuz kullanılabilecek şekilde tasarlanması gerekliliği ile birlikte, yardım ve destekleyici dokümantasyon yine de gerekli olabilir. Sunulan yardım ve dokümantasyon kaynağı kolayca erişilebilir olmalıdır. Üzerinde uğraşılan görevle ilgili arama yapılabilmesi, sade ve basit olması kullanımı kolaylaştıracaktır. Uygun durumlarda kullanım örnekleri de sağlanmalıdır.

2.3.2. Üzerinden Geçme

Üzerinden geçme, bir arayüzün gerçek kullanıcılar olmaksızın değerlendirilmesidir. Bu yöntemde geliştiriciler, analistler ve İBE uzmanları, kendilerini tecrübesiz kullanıcının yerine koyarak, belirlenmiş görevlerin üzerinden geçerler. Kullanıcıların çoğu yeni bir sistemi öğrenmek için eğitim almak ya da kullanım kılavuzları okumak yerine, deneme yanılma yöntemini tercih etmektedir [Carroll and Rosson, 1987]. Üzerinden geçme ile değerlendirilmek olan bir sistemde, bir görevi yerine getirmek için geçilen aşamaların hepsi için başarılı ve başarısız senaryolar için, kullanıcının başarılı senaryoları seçip seçmeyeceği ve nedenleri raporlanır. Bu yöntem ilk olarak [Lewis et al., 1990] tarafından önerilmiştir.

Üzerinden geçme yöntemi, görevlere özel kullanılabilirlik problemlerini bulmada çok etkilidir. Kullanıcıların hedeflerini ve varsayımları anlamayı kolaylaştırır. Ayrıca geliştirmenin erken evrelerinde de kullanılabilir. Diğer taraftan, zaman alıcı olması ve sadece öğrenilebilirlik problemlerini ortaya çıkardığı için kullanım alanı dardır.

2.3.3. Kullanıcı Testleri

Sezgisel değerlendirme geliştirmenin erken aşamalarında kullanılabilir, fakat kullanıcı testleriyle desteklenmelidir, çünkü gerçek kullanıcıların ne beklediklerini tahmin etmek her zaman mümkün olmamaktadır. Mevcut kullanılabilirlik seviyesine karar verebilecek olanlar son kullanıcılar veya kullanıcı adaylarıdır. Ürün son kullanıcıların hizmetine girmeden önce, kritik görevler kümesi bir grup aday kullanıcı tarafından laboratuarda ya da kullanıcının çalışma ortamında test edilir [Preece, 1994].

Bu yöntemde ilk yapılması gereken test planının hazırlanmasıdır [Shneiderman, 1997]. Kullanılabilirlik mühendisleri, tasarımcılarla bir araya gelerek testlerin başlamasından 2-6 hafta önce test planını hazırlamaya başlarlar. Planda, testlerde gerçekleştirilecek görevler, başarı kriterleri ve toplanacak metrikler seçilir. Her deney için verilecek süre ve toplam katılımcı sayısı da planda belirtilir. Ayrıca katılımcıların hangi kaynaklardan bulunacağı da planlanır. Planın hazırlanmasının ardından, deneye en geç bir hafta kala, 1-3 pilot deney gerçekleştirilir, böylece deneyde var olan eksiklikler gerçek deneyler başlamadan düzeltilir.

Deneyler tasarlandıktan sonra, gerçek kullanıcıları temsil eden kullanıcılar seçilmelidir. Deneyin konusuna göre değişmekle birlikte, kullanıcının ilgili alandaki tecrübesi, eğitimi, dil becerisi seçimi etkileyen unsurlar arasındadır. Seçilen kullanıcılar, kendilerinin becerilerinin değil programın test edildiği konusunda ikna edilmelidir. Katılım tamamen gönüllü olmalıdır ve kullanıcı istediği zaman deneyi bırakabilmelidir. Deney başlamadan önce, katılımın karşılığında simgesel de olsa bir hediye verilmelidir.

Deneyler sırasında hata oranı, gerçekleştirme süresi, yardım dokümanlarına başvurma sayısı gibi metrikler -planda belirtildiği yöntemlerle- toplanır. Deneyler sırasında, bir gözletmen kullanıcıyla ilgilenir ve bu sırada gözlemsel bazı verileri toplar. Ayrıca deneyler video-kamera ile ya da masaüstü kaydedici programlar ile kaydedilir. Ayrıca, kullanıcıdan deney boyunca aklından geçenleri sesli olarak ifade etmesi istenir.

Deneyler tamamlandıktan sonra ortaya çıkan kullanılabilirlik problemleri, kullanılabilirlik mühendisleri tarafından önceliklendirilerek raporlanır. Hatalar giderildikten sonra, evrimsel geliştirme metodolojisine göre gerek görülürse deneyler tekrarlanır ve ürün olgunlaşmıca kadar bu süreç devam eder. Fakat kullanılabilirlik problemlerinin bir kısmı fark edilmeden kalabilir. Bu durumda, gerçek kullanım sırasında elde edilen geri beslemeler değerdendirilerek gerekli iyileştirmeler yapılır.

3. KULLANILABİLİR UYGULAMA PROGRAMLAMA ARAYÜZLERİ

3.1. UPA Kullanılabilirlik Hedefleri

Uygulama programlama arayüzlerinin kullanılabilirliğini sağlamak için ulaşılmaması gereken hedefler, Bölüm 2.1’de sıralanan İBE hedeflerinin UPA’lara uyarlanmasıyla elde edilir. Kullanıcının yazılım geliştirici olduğu, ve kullanılan arayüzün bir UPA olduğu durumda İBE hedefleri özelleştirilmiş ve UPA’ların kullanılabilirliği için aşağıdaki hedefler elde edilmiştir [Ardıç ve Göktürk, 2009]:

3.1.1. Kullanıcının İhtiyaç Duyduğu Fonksiyonları Sağlaması

Her UPA’nın bir kapsamı vardır. Yazılım geliştiricinin ihtiyaç duyacağı fonksiyonlar, bu kapsama girmelidir. Bu nedenle UPA, konusu dâhilinde kullanıcının ihtiyaç duyacağı tüm fonksiyonları sağlamalıdır. Kullanıcının kendisinin gerçeklemek zorunda kalacağı eksiklikler bırakılmamalıdır.

Diğer taraftan, kullanıcının ihtiyaç duyacağı tüm fonksiyonların sağlanması tek başına yeterli değildir. Kullanıcının da sağlanan fonksiyonları algılayabilmesi gereklidir. Farkına varılamayan işlevsellik, pek çok üründe rastlanan bir kullanılabilirlik hatasıdır [Clarke, 2004].

3.1.2. Kolay Öğrenilmesi

Bir UPA ile yazılım geliştirmenin verimli olabilmesi için, o UPA’nın kolay öğrenilebilmesi gerekmektedir. UPA’ların kullanıcılar tarafından keşfedilmesinin kolaylığı ve yardımcı kaynakların varlığı, kolay öğrenilme özelliğini destekler.

3.1.3. Kolay Hatırlanması

Kullanıcının yazılım geliştirirken ihtiyaç duyacağı sınıflar ya da metotların akılda kalıcı olması gerekmektedir. Uzun süreli hafızadan (“long term memory”) getirilen bilgiler, sınıfların isimleri, metotların ait oldukları sınıflar gibi önceden öğrenilmiş, detaylı bilgilerdir. Kısa süreli hafızadan (“short term memory”) getirilen bilgiler ise dokümantasyona bakıldıktan hemen sonra hatırlanması gereken,

parametrelerin türleri ve sırası gibi, bilgilerdir. Kolay hatırlanma, uzun ve kısa süreli hafızada depolanacak bilgilerin kalıcı olması özelliğidir.

3.1.4. Kodun Kolay Yazılabilmesi

Kullanıcının kod yazarken karşılaşıacağı zorlukların engellenmesi özelliğidir. Kullanıcının ek kontroller yapmak zorunda bırakılmaması örnek olarak verilebilir. Bu özellik diğer tüm maddelerle ilişkilidir.

3.1.5. Kullanıcı Kodunun Anlaşılabilirliği

Kullanıcının yazdığı kodun okunabilir ve anlaşılabilir olması özelliğidir. Bu özellik sayesinde, bir UPA kullanılarak yazılmış bir kod parçasının değiştirilebilirliği ve yeniden kullanılabilirliği de artacaktır.

3.1.6. Yanlış Kullanımın Zorluğu

Kullanıcının hata yapmasını engelleyici önlemlerin alınması özelliğidir. Hedeflerin en önemlisi belki de bu maddedir. Bir UPA'nın kullanımı kolay, yanlış kullanımı zor olması gereklidir. Aksi halde, kullanıcının yapacağı hatalar UPA'yı kullanmasına karşı direnç göstermesine sebep olacaktır.

3.2. UPA Kullanılabilirliğine Etki Eden Faktörler

Bölüm 2.2'de verilen kullanılabilirlik hedefleri göz önünde bulundurularak yapılan literatür çalışması neticesinde aşağıdaki 11 faktör elde edilmiştir. Bu faktörler, bir ya da birden fazla hedefle ilişkili olduğu için herhangi bir üst gruplandırma ya da sıralama yapılmamıştır.

3.2.1. Karmaşıklık

Bir UPA'nın çok fazla fonksiyon içermesi, kullanıcıya esneklik sağlaması açısından iyidir, fakat aralarından seçim yapmak zor olacağından kullanılabilirliği olumsuz etkileyebilir. UPA'nın sunduğu fonksiyon listesi, kullanıcının ihtiyaç duyacağı tüm fonksiyonları içermeli, fakat hatırlama ve öğrenme zorluğu yaratacak kadar da karmaşık olmamalıdır. UPA'nın çok karmaşık olması hata yapma ihtimalini de artırır. Soyutlama-karmaşıklık dengesi iyi ayarlanmalıdır.

3.2.2. İsimlendirme

UPA geliştiricileri, sınıfların, metotların ve parametrelerin isimlendirmesini özenli yapmalıdır. Tutarsız ve ilgisiz isimlendirmeler öğrenim ve kullanım zorluğu yaratır. Başarılı isimlendirme, UPA'nın kolay öğrenilmesi ve hatırlanması için anahtar roldedir [Henning, 2009].

Kullanılan isimler kendini açıklayıcı nitelikte olmalı ve mümkünse gerçek hayatla analogi kurabilmelidir. Çünkü bir UPA tıpkı bir dil gibidir. Kullanıcıların o UPA ile okumayı ve yazmayı öğrenmesi gereklidir. Doğru isimlendirme sağlandığı takdirde, yazılan kod düzyazı kadar rahat okunabilecektir [Bloch, 2006]. Kullanıcıyı mümkün olduğunca dokümantasyona sevk etmeyecek şekilde isimlendirme yapılmalıdır.

İsimlendirme sırasında yaygın alışkanlıklara uyulmalı ve UPA kendi içinde tutarlı olmalıdır. Her noktada aynı şeylerin her zaman aynı isimlendirilmesi ve aynı olmayan şeylerin de asla aynı isimlendirilmemesi önem taşımaktadır.

3.2.3. Dokümantasyon

Dokümantasyon, kolay ve hızlı öğrenmeyi destekleyen her türlü kaynağı içerir. UPA'lar, kullanıcıların mümkün olduğunca dokümantasyona başvurmayacakları şekilde tasarlanmalıdır [Henning, 2009]. Yine de ayrıntılı bir dokümantasyon sağlanmalı, sık gerçekleştirilen görevler için kod örnekleri hazırlanmalıdır. Açık kaynak kodlu UPA'larda, kullanıcı kodun içine girebileceği için, kaynak kodun yorumlarla ('comment') desteklenmesine dikkat edilmelidir.

UPA'lar yayınlanmalarının ardından birçok yazılım geliştirici tarafından kullanılır ve bu sayede UPA'ya dair belirli bir tecrübe birikir. Dokümantasyonun, yeni UPA kullanıcılarına yol gösterecek şekilde bu tecrübeden beslenmesi, UPA'nın kullanılabilirliğini olumlu yönde etkileyecektir. "Jadeite" adlı çalışmada [Stylos et al, 2009], UPA'ların mevcut dokümantasyonlarının, internette yayınlanan kullanım bilgilerinden elde edilen çeşitli tavsiyelerle zenginleştirilebileceği gösterilmiştir.

Dokümantasyon konusunda sıkça yapılan bir yanlışlık, bir hatayı ya da eksikliği dokümantasyona ekleyerek sorumluluktan kurtulmaya çalışmaktır [Clarke, 2004]. Dokümantasyona güvenerek aksaklıkların düzeltilmemesi kullanılabilirliği olumsuz etkilemektedir.

3.2.4. Nesne Yaratılması

Nesne yönelimli programlama dillerinde, nesnelerin nasıl yaratılacağı UPA tasarımındaki en önemli noktalardan birisidir. Fabrika veya inşacı gibi tasarım kalıplarının kullanımı ya da yaratıcı metot kullanılarak nesne yaratılması, tasarım sırasında karar verilmesi gereken konulardandır. Yapılan seçimin kullanılabilirliğe etkisi üzerinde de çalışmalar yapılmaktadır.

Fabrika tasarım kalıbının kullanılabilirliğe etkisi üzerinde yapılan bir deneysel kullanılabilirlik çalışmasında [Ellis et al, 2007], yazılım geliştiricilerin nesneleri fabrikalardan elde ederken zorlandıkları gözlenmiştir. Aynı çalışmada, fabrika tasarım kalıbının çoğu durumda kullanılmaktan kaçınılmasının mümkün ve gerekli olduğu belirtilmektedir.

Nesne yaratılmasında bir diğer önemli karar, sınıfların yaratıcı metotlarının ne kadar iş yapacağıdır. Yaratıcı metotlar için iki temel seçim söz konusudur. İlki, metotların bazı parametreler alarak yarattığı nesne üzerinde ilklendirmeler yapmasıdır. Bu ilklendirmeler genellikle sınıfın “gerekli” özelliklerinin atanmasını içerir. Bu gerekli özellikler, ilgili sınıfın bir örneğinde muhakkak var olması gereken özelliklerdir. Örneğin bir çemberi iki boyutlu düzlemde ifade etmek için bir merkez noktası ve bir de yarıçap uzunluğuna ihtiyaç vardır:

```
Circle circle = new Circle(center, radius);
circle.use();
```

Şekil 3.1. Örnek. Parametrelili yaratıcı metot kullanımı

Diğer seçim ise, yaratıcı metodun herhangi bir parametre almaması ve herhangi bir ilklendirme yapmamasıdır. Gerekli ilklendirmeler, nesne yaratıldıktan sonra kullanıcı tarafından yapılır.

```
Circle circle = new Circle();
circle.setCenter(center);
circle.setRadius(radius);
circle.use();
```

Şekil 3.2. Örnek. Parametrelili yaratıcı metot kullanımı

Yaratıcı metodun aldığı parametrelerin kullanılabilirliğe etkisi yapılan deneylerle karşılaştırılmıştır [Stylos and Clarke, 2007]. Parametrelili yaratıcı metodun daha kullanılabilir olduğu hipoteziyle yola çıkılmıştır, çünkü parametrelili yaratıcı metodun arkasından atama yapmaya gerek yoktur. Ayrıca, yaratıcı metot aldığı parametreler ile nesnenin nasıl kullanılması gerektiği konusunda kılavuzluk etmektedir. Fakat deneylerde parametresiz yaratıcı sınıflarla hem geliştirme yapmanın, hem de yazılmış kodu okumanın daha kolay olduğu ortaya çıkmıştır. Bu nedenle parametrelili metot kullanımından kaçınılmalı, kaçınılamıyorsa da en azından parametresiz bir yaratıcı metot mutlaka kullanıcıya sunulmalıdır.

3.2.5. Metot Parametreleri ve Dönüş Tipi

Özellikle arka arkaya aynı türde çok sayıda parametre kullanımı kullanılabilirliği olumsuz etkiler. Kısa süreli hafızada çok miktarda veri tutulamayacağından 3 ve daha az parametre kullanılmaya gayret gösterilmelidir. Ayrıca parametrelerin sırası ve türü tutarlı olmalıdır [Bloch, 2006].

Dönüş değerinin başarıyı belirtmesi kullanımı kolaylaştırır. Metodun işini başarıyla yerine getirip getirmediği bazı durumlarda dönüş değerinde kullanıcıya sunulabilir. Bu noktada dönüş değerinin ek kontrollere yol açmaması gerektiği belirtilebilir. (Örneğin, kullanıcının hata almaması için, null yerine boş bir dizi dönmek.)

3.2.6. Metot Yerleşimi

Nesne yönelimli programlama dillerinde, bir metodun hangi sınıf üzerinde yer aldığı kullanılabilirliği büyük ölçüde etkilemektedir [Stylos and Myers, 2009]. Jeffrey Stylos ve Brad Myers tarafından yapılan çalışmada, yazılım geliştiricilerinin hiç bilmedikleri bir UPA'yı keşfetmeye büyük çoğunlukla aynı sınıf üzerinden başladıkları gözlenmiştir. Bu durum, insanların UPA'ları öğrenirken doğal bir strateji

izlediklerini göstermektedir. Yazılım geliştiriciler başlangıç sınıfında işaret edilmeyen sınıfları daha geç keşfetmektedirler.

Aşağıda iki farklı UPA kullanılarak e-posta göndermek için yazılmış kod parçaları bulunmaktadır. Benzer görünümlere sahip bu iki satır kod öğrenilebilirlik açısından farklı sonuçlar doğurmaktadır.

```
1. mailServer.send(mailMessage);
2. mailMessage.send(mailServer);
```

İlk satırda verilen kod açık ve anlaşılır görünmektedir. Fakat yazılım geliştiricilerin bir e-posta UPA'sını keşfetmeye MailMessage sınıfından başladıkları kabul edildiğinde, MailServer sınıfına ulaşmaları daha çok zaman alacaktır. Bu sebeple ikinci satırdaki alternatif kodun üretilmesi, daha kolay ve yazılım geliştiricinin zihinsel modeline daha uygundur.

UPA tasarımcıları, gerektiğinde kullanıcı deneyleri yaparak UPA'nın başlangıç sınıfına doğru karar vermeli ve ilgili olabilecek metotları bu sınıf üzerine yerleştirmelidir. Sık gerçekleşen görevlerde kullanılacak diğer sınıfların başlangıç sınıfı tarafından işaret edilmesine özen gösterilmelidir.

3.2.7. Hata ve Kural Dışı Durum İşleme

Program akışı içinde hatalar ve kural dışı durumlar ('exception') kaçınılmaz olarak oluşmaktadır. Dikkat edilmesi gereken nokta, hata mesajlarının yeterli bilgi içermesi ve 'exception'ların sadece kural dışı durumlarda fırlatılmasıdır [Bloch, 2006]. Diğer taraftan, kural dışı durumlar olduğu halde 'exception' fırlatmamak ve bunun yerine hata kodu dönmek metotların kullanılabilirliğini düşürmektedir.

Her aşamada kullanıcının hata yapması zorlaştırılmalıdır. Bu sebeple metot ve alan erişimlerinin sınırlandırılması için gerekli erişim niteliklerinin ('private', 'public' gibi) kullanımlarına dikkat etmek doğru bir pratiktir. Kullanıcı erişiminin şart olmadığı metotlara dışarıdan erişim engellendiğinde hem UPA sadeleşmiş olur, hem de sınıflar arası bağlaşım ('coupling') azalır.

3.2.8. Kademeli Sınama

Yazılım geliştiriciler, program henüz tamamlanmasa da, yazdıkları kodu belirli noktalarda deneyerek geri besleme elde etme ihtiyacı duyarlar. Özellikle bir UPA'yı ilk kez kullanan kullanıcı, geliştirme süreci sırasında sıklıkla geriye dönerek kendi geliştirme yöntemlerini sınamaya ve aşama aşama elde ettiği sonuçları değerlendirmeye eğilimlidir. Kademeli olarak gerçekleştirilen bu geriye dönüşler ve değerlendirmeler, UPA'nın öğrenilmesine büyük katkıda bulunur. UPA'lar, henüz tamamlanmamış kod parçalarının denenebilmesine olanak verecek şekilde tasarlanmalıdır [11]. Bu pratik bir örnekle daha rahat anlaşılabilir [Clarke, 2004]:

```
IPAddress hostIPAddress = IPAddress.Parse("130.23.43.234");
IPAddress theAddress = new IPAddress(hostIPAddress);
CEmailServer theServer = new CEmailServer(theAddress);

if (theServer.Connect())
{
    CMailbox theMailbox = theServer.GetMailbox("stevencl");
    theMailbox.Open("stevencl", true);
    CEmailMessage theMsg = theMailbox.NewMessage();
    theMsg.Build("message", "subject", "to@you.com");
    theMailbox.Send(theMsg, true);
    theMailbox.Close();
}
else
    Console.WriteLine("Could not connect to the mail server");
```

Şekil 3.3. Kademeli Sınama Örneği - I

Yukarıda, bir UPA kullanılarak yeni bir e-posta oluşturulması ve gönderilmesini sağlayan bir kod parçası verilmiştir. Bu UPA'nın kullanıcıları için, senaryo tamamlanmadan ilerlemelerini kontrol etmeleri zordur. Çünkü sunucuya bağlanıp posta kutusunu açmadan bir CEmailMessage nesnesi yaratmak mümkün değildir. Bu sebeple bu UPA'nın kademeli sınamayı desteklediği söylenemez. Oysa UPA aşağıdaki kullanıma uygun olarak tasarlanmış olsa, kademeli olarak sınanabilecektir:

```

CEmailAddress theMsg = new CEmailAddress();
theMsg.Build("message", " subject", "to@you.com");

/* CEmailAddress nesnesi bu aşamadan itibaren sinanabilir */
IPAddress hostIPAddress =
IPAddress.Parse("130.23.43.234");
IPAddress theAddress = new IPAddress(hostIPAddress);
CEmailServer theServer = new CEmailServer (theAddress);

if (theServer.Connect())
{
    CMailbox theMailbox = theServer.GetMailbox ("stevencl");
    theMailbox.Open("stevencl", true);
    theMailbox.Send(theMsg, true);
    theMailbox.Close();
}
else
    Console.WriteLine("Could not connect to the mail server");

```

Şekil 3.4. Kademeli Sınama Örneği - II

Kademeli sınama sayesinde, nihai ürüne varana kadar, gerçekleşen tüm program bileşenlerinin gözden geçirilmesi ve sınanması sağlanmış olur.

3.2.9. Çağrı Sonrası Anlaşılabilirlik

UPA metotları çağrıldıktan sonra anlaşılabilirliklerini koruyacak şekilde tasarlanmalıdır. Çözülme istenen probleme UPA kullanıcısının perspektifi ile yaklaşılmalıdır. Bu özellik basit bir örnekle açıklanabilir. Aşağıda, özenle isimlendirilmiş bir metot tanımlaması yer almaktadır [Henning, 2009]:

```

void makeTV(bool isBlackAndWhite, bool isFlatScreen) {
/* ... */
}

```

Fakat, bu metot çağrı sonrası anlaşılabilirliğini yitirmektedir. Renkli ve düz ekran bir TV yaratmak isteyen bir kullanıcı aşağıdaki çağrıyı yapacaktır:

```
makeTV(false, true);
```

Bu satırda bir TV yaratıldığı isimlendirmeden anlaşılmaktadır. Fakat parametrelerin ne anlama geldiğini kestirmek olanaksızdır. Aşağıda çok daha anlaşılabilir bir çağrı bulunmaktadır:

```
makeTV(Color, FlatScreen);
```

İkinci alternatifte renkli ve düz ekran bir TV yaratıldığı anlaşılabilmektedir. Hem çağrı sonrası anlaşılabilirlik artmış, hem de hata yapma ihtimali azaltılmıştır. Farkı yaratan aşağıdaki `enum` tanımlamalarını kullanan yeni metod imzasıdır:

```
enum ColorType {
    Color,
    BlackAndWhite };

enum ScreenType {
    CRT,
    FlatScreen };

void makeTV(ColorType col, ScreenType st);
```

UPA tasarımcısı açısından, kullanıcı perspektifini göz önünde bulundurmak bir miktar daha fazla çaba gerektirebilir, fakat sonuçta daha kullanılabilir bir UPA ortaya çıkacaktır.

3.2.10. Kaynak Kodunun Anlaşılabilirliği

Özellikle açık kaynak kodlu UPA'lar için, kaynak kodunun anlaşılabilirliği önemlidir. Kod okunabilir yazılmış olmalı ve kullanılan programlama dilinin geleneksel yazım standartlarına uyulmalıdır. Girintilemeye ("*indentation*") ve kod bloklarını dilin ayraçlarını kullanarak ayırmaya özen gösterilmelidir. Tekrar eden kod parçaları, çok büyük sınıflar ve uzun metotlar, uzun parametre listeleri kodun okunabilirliğini azaltan ve yazılım kalitesini düşüren unsurlar sayılır [Fowler and Beck, 2000]. Bu özelliğin sağlandığını kontrol eden, hatta bazı ufak düzeltmeler yapan yazılımlar mevcuttur, bunlardan faydalanılabilir (Bkz. [Checkstyle]).

3.2.11. UPA'larda Değişiklik

UPA'lar kullanılmaya başlandıktan sonra değişiklik yapılması istenen bir durum olmasa da kaçınılmazdır. Bu değişikliklerin mümkün olduğunca seyrek yapılması gerekliliği bir tarafa, en önemli ayrıntı, bir UPA'da yapılan değişikliğin, o UPA'nın eski versiyonunu kullanan programlarda bir uyumsuzluğa sebep olmamasıdır. Bir değişiklik söz konusu ise, bu zorunluluk zamana yayılabilmeli, önceden yazılmış kodlar "geçersiz" duruma düşmemelidir. Java programlama dili,

geriye dönük desteğin mümkün olduğunun en güzel ispatıdır. Java'nın tüm eski versiyonlarında yazılmış tüm kodlar, şu anda Java'nın son sürümü olan 1.6'da halen çalışabilmektedir.

Ayrıca, bir UPA'dan tamamen desteğin kesilerek yepyeni bir UPA ile kullanıcı karşısına çıkmak da büyük zorluklara yol açmaktadır. Bu nedenle UPA'lar öldürülmemeli, yavaş yavaş yaşlandırılmalıdır [Zibran, 2008].

3.3. UPA'larda Kullanılabilirlik Değerlendirmesi

UPA'ların teknik olarak değerlendirilmesi, daha sık yapılan ve daha somut sonuçları olan bir çalışmadır. Diğer taraftan, kullanılabilirlik açısından değerlendirilmesi, çok gerekli olmakla birlikte, henüz oturmuş yöntemler bulunmadığı için teknik değerlendirmeye kıyasla daha zordur.

UPA'ların kullanıcı merkezli bir şekilde tasarımı için ilk defa kullanıma sunulmadan önce mutlaka ayrıntılarıyla değerlendirilmesi ve elde edilen bulgulara göre adımlar atılması gerekmektedir. Ayrıca UPA'lar üzerinde yapılacak değişikliklerin yayınlanmasından önce de, değişikliklerin kullanılabilirliğe zarar vermediğinden emin olmak için değerlendirme yapılmalıdır. Tez çalışması kapsamında UPA kullanılabilirliğini etkileyen unsurları sunmanın yanında, UPA kullanılabilirliğini değerlendirmek için bir çatı sunulması da hedeflenmektedir.

3.3.1. Sezgisel Değerlendirme

Sezgisel değerlendirme, kullanıcı arayüzünün tasarımındaki problemleri ortaya çıkarmak için kullanılan geleneksel yöntemlerden biridir. Kullanıcı arayüzü, uzman bir değerlendirici tarafından, belirli bir kıstas kümesine göre değerlendirilerek o kriterlere uyum ölçülür. Sezgisel değerlendirme yöntemi, uygulama programlama arayüzlerinin değerlendirilmesinde de kullanılabilir. Bu çalışmada UPA'ların sezgisel değerlendirilmesinde kullanılmak üzere Bölüm 3.2'te verilen faktörlerin kriter kümesi olarak kullanılması önerilmektedir.

3.3.2. Kullanıcı Deneyleri

İnsan-bilgisayar etkileşimi hedeflerini yerine getirmek için en önemli nokta, kullanıcının zihinsel modelinin doğru anlaşılmasıdır. Bu amaçla, kullanıcı ile insan-bilgisayar etkileşimi laboratuvarında ya da kullanıcının kendi çalışma ortamında bazı deneyler yapılır. Deneye katılan kullanıcılara sembolik de olsa bir karşılık verilir (bir kitap ya da lisanslı bir yazılım olabilir.). Denekler, test edilen UPA'nın alanında daha önce çalışmamış, fakat programlama diline hâkim programcılar arasından seçilmelidir. Bu çalışmalar 3 aşamalı olarak tasarlanmıştır:

3.3.2.1. Sesli Düşünme

Sesli düşünme protokolü (“think aloud protocol”), kullanıcı grafik arayüzlerinin kullanılabilirlik açısından değerlendirilmeleri için yapılan deneyler sırasında, kullanıcıdan doğru veriyi toplayabilmek için sıkça başvurulmuş bir yöntemdir. Kullanıcı kendisine verilen görevleri gerçeklerken aklından geçenleri sesli ifade ederek, zihinsel modelindeki ürün ile değerlendirilen ürün arasındaki farkı ortaya çıkarır.

Sesli düşünme yöntemi, UPA değerlendirmesine uyarlanırken bir miktar genişletilir. Görev gerçekleştirme aşamasından önce, deneklere bazı programlama görevleri verilir. Uygulama programlama arayüzü ya da herhangi bir dokümantasyona erişim sağlanmaz. Deneklerden verilen görevleri yerine getirmeleri için yalancı kod (“pseudo code”) yazmaları istenir [Beaton and Myers, 2008]. Böylece, kullanıcı beklentileri ilk olarak *yazılı* olarak ifade edilmiş olur. İsimlendirme, karmaşıklık, parametre türleri ve sırası, nesne yaratımı, metot yerleşimi gibi başlıklarda beklentiler elde edilebilir. Bu bilgilerle UPA'nın kullanıcının zihinsel modeline ne kadar uyduğunun değerlendirmesi yapılır. Yazılı ifadenin ardından, görev gerçekleştirme aşamasına geçilir (Bkz. Bölüm 3.3.4). Görevler gerçekleştirilirken sesli düşünme protokolü uygulanır. Kullanıcı, karşılaştığı problemleri ya da düşünmekte olduğu şeyleri kendi cümleleri ile ifade eder.

3.3.2.2. Görev Gerçekleme

Bu aşamada deneklerden Sesli Düşünme bölümünde bahsi geçen görevleri, UPA'yı kullanarak gerçeklemleri istenir. Görev gerçekleştirme aşaması boyunca

kamera ile kayıt yapılır. Bu yöntemle aşağıdaki veriler toplanabilir [Shneiderman, 1997]:

a) Öğrenme zamanı: Kullanıcıların verilen görevleri yerine getirmek için gerekli sınıflar ve metotlar ile bunlar arasındaki ilişkiyi öğrenmeleri için geçen zaman.

b) İşlem hızı: Kullanıcıların verilen görevleri yerine getirmek için harcadıkları zaman.

c) Hata sayısı: Kullanıcıların verilen görevleri yerine getirirken yaptıkları hataların türleri ve sayısı (Kullanıcılar hata yaparken işlem hızları da olumsuz etkilenecektir. Fakat kullanıcıların yaptıkları hatalar, kullanılabilirlik problemlerine doğrudan etki ettiği için hataların da ayrıca değerlendirilmesi gerekmektedir.).

d) Hatırlama oranı: Kullanımın üzerinden geçen belirli bir süre sonunda, öğrenilenlerin ne kadarının akılda kaldığı.

Deneye başlamadan önce her görev için ölçülebilir hedefler koyulur. Deneylerin tamamlanmasıyla elde edilen veriler, hedeflenenlerle karşılaştırılarak iyileştirmeye gerek olup olmadığına karar verilir. Ayrıca, kaydedilen görüntüler daha sonra incelenerek kullanıcıların yaşadıkları zorluklar ve dokümantasyona başvurdıkları noktalar belirlenir.

3.3.2.3. Anket

Kullanıcı tatminini ölçmek için doğrudan kullanıcıdan görüş almak etkili bir yöntemdir. Verilen bir dizi görev, değerlendirilen UPA ile gerçekleştirildikten sonra, kullanıcıya bir anket sunularak memnuniyet derecesi anlaşılmaya çalışılır. Bir kullanıcı için düşünüldüğünde verilen yanıtın öznel olacağı ve sağlıklı sonuç vermeyeceği doğru olsa da, tüm deneklerden alınan sonuçların ortalaması alındığında kullanıcı tatmini ile ilgili nesnel veri elde edilir [Nielsen, 1993].

Cognitive Dimensions (CD), sistemler üzerinde hızlı ve etkili kullanılabilirlik değerlendirmesi yapabilmek için Thomas Green tarafından ortaya konmuş bir çatıdır [11]. UPA'ların kullanılabilirlik problemlerini ortaya çıkarmak için kabul görmüş ve uygulanmıştır [Clarke, 2005]. CD, her biri kullanılabilirliği etkileyen 13 boyuttan oluşur. Bu 13 boyut, sezgisel değerlendirme sırasında bir uzman tarafından kullanılabilirliği gibi, kullanılabilirlik anketlerinin oluşturulmasında da çerçeve görevi görmüştür [Blackwell and Green, 2000]. Deneyin bu aşamasında kullanıcıdan, CD'deki her bir boyutla ilgili düşüncelerini ortaya çıkarıcı nitelikte bir anketi yanıtlaması istenir. Böylelikle 13 boyut üzerinden görev gerçekleştirme aşamasında gözden kaçırılan, fakat kullanıcının zorlandığı veya eksik bulduğu bir durum varsa yakalanmış olur.

4. HİPOTEZ

Tez çalışması kapsamında UPA kullanılabilirliğini etkileyen faktörler incelenerek, üç faktör üzerinden hipotezler kurulmuştur. Bu hipotezler, literatürde işaret edildiği doğrultuda oluşturulmuştur.

4.1. İsimlendirmenin Kullanılabilirliğe Etkisi

Sınıfların, metotların ve parametrelerin özenli isimlendirilmesi, isimlendirme yapılırken yaygın alışkanlıklara uyulması ve isimlendirmede tutarlılık kullanılabilirliği doğrudan etkilemektedir. İsimlendirme kötü yapıldığında yazılım geliştiricilerin verimlilikleri ve memnuniyetleri düşecektir.

Java’da metotlar bir fiil ile isimlendirilir. O metot içinde hangi iş yapılıyorsa, yazılım geliştirici metot çağrısını kodunda emir kipinde bir cümle ile yapmaktadır. Tüm metot isimlerinin fiilden isme dönüştürülmesi, kullanıcıları yavaşlatacaktır.

4.2. Parametrelili Yarattıcı Metotun Kullanılabilirliğe Etkisi

Nesnelerin yaratıcı metot içerisinde yaratıldıktan sonra, “gerekli” parametreler ile ilklendirilmesi, varsayılan nesne yaratıcısı (hiç bir parametre almayarak sadece nesneyi yaratan yaratıcı metotlar) ile yaratılmasına kıyaslandığında kullanılabilirliği düşürdüğü gözlenecektir. Güncel kütüphanelerdeki yaygın kullanımın ve çok sayıda tecrübeli UPA tasarımcısının sezgisinin aksine, parametresiz yaratıcı metot ile nesneleri yaratarak, gerekli özelliklerin sonradan atanması kullanıcıların verimliliğini arttıracaktır.

4.3. Metot Yerleşiminin Kullanılabilirliğe Etkisi

Kullanıcılar bir görevi gerçeklemek için, hiç bilmedikleri bir UPA’yı keşfetmeye çoğunlukla aynı sınıf ile başlamaktadır. Örneğin, düz bir çizgi çizmesi istenen bir kullanıcı, “Line” ya da benzer isimli bir sınıf ile denemelerine başlayacaktır. Bu başlangıç sınıfına “ana sınıf” adı verilir. Yaratılan bir “Line” nesnesini çizdirmek isteyen bir kullanıcı, ilgili metodu aynı Line sınıfında arayacaktır. Line sınıfı tarafından işaret edilmeyen bir sınıfta bulunması kullanılabilirliği azaltacaktır.

`Graphics2D.draw(Line)` çağrısından, `Line.draw()` çağrısı yapmak kullanıcının zihinsel modeline daha uygundur.

5. YÖNTEM

Hipotezlerin sınanması için kullanıcı deneyleri yöntemi benimsenmiştir. Deneylerde kullanıcılara her bir hipotez için 2 UPA verilerek, hangi UPA üzerinde daha rahat uygulama geliştirdiklerinin gözlemlenmesine karar verilmiştir. İlk adım olarak bu hipotezlerin ayrı ayrı sınanması için, uygun özelliklerde UPA'lar aranmıştır. Fakat, karşılaştırmak için bulunan UPA'ların birbirinden sadece hipotez konusunda değil, farklı bir çok kriter üzerinden ayrıldığı fark edilmiştir. Bu nedenle, en sağlıklı yöntemin UPA'ları kontrollü bir şekilde üretmek olduğuna karar verilmiştir.

5.1. UPA Seçimi

Seçilen yöntemde, mevcut bir UPA seçilmiş ve bu UPA'nın isimlendirme, metot yerleşimi ve parametrelili yaratıcı metot özelliklerinin her biri, seçilen UPA'dakinin tersi olacak şekilde yeni UPA'lar üretilmiştir.

Grafik kütüphaneleri incelenerek, Java SE JDK içerisinde bulunan Java 2D API seçilmiştir. Java 2D™ API, Abstract Windowing Toolkit (AWT) paketi içinde yer alır ve iki boyutlu grafik, yazı ve resim işlemeden sorumludur. Zengin kullanıcı arayüzleri geliştirmek, çizim programları veya resim editörlerinde kullanılabilecek tam kapsamlı bir çatı sunar. Bu kütüphanenin başta gelen işlevleri iki boyutlu temel şekiller (dikdörtgen, elips, düz çizgi, eğri gibi) ile rastgele şekillerin çizdirilmesi, boyanması, çeşitli kenar ve dolgu biçimleri ile zenginleştirilmesi; grafik dönüşümleri (uzayda yer değiştirme, boyut değiştirme, vs)'dir. Java 2D API, çok büyük ölçekli olduğu için, kullanıcı deneylerinde tamamının kapsanması mümkün değildir. UPA'nın kullanılacak kısmı, daha basit ve görsel olduğu için, şekil çizdirme ve boyama alanına daraltılmıştır.

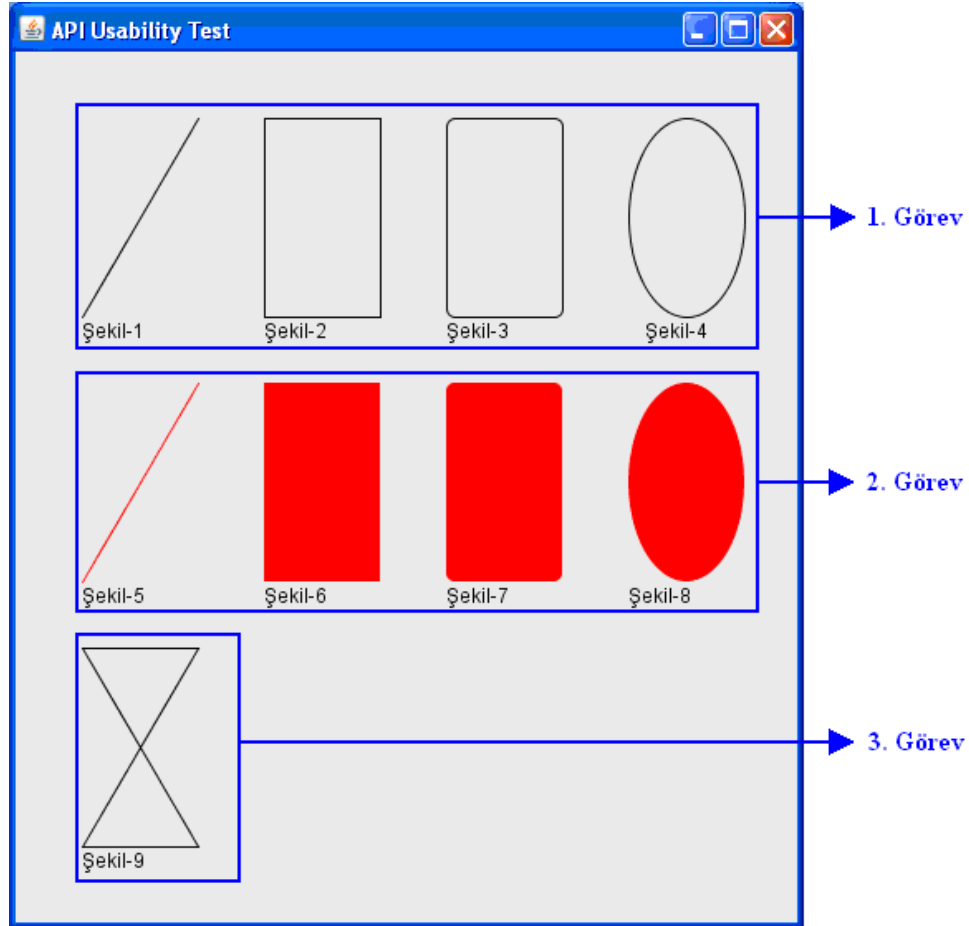
5.1.1. Görevlerin Seçimi

UPA seçiminin ardından verilen en önemli karar kullanıcıların gerçekleyeceği görevlerdir. Çünkü kullanılabilirlik deneylerinin kapsayacağı kısımlar, bu görevleri gerçeklemek için gerekli olan sınıflar ve sınıfların ilgili metotlarıdır. Deneylerde gerçekleştirilmek üzere 3 görev tasarlanmıştır (Bkz. Şekil 5.1):

1. Görev: Düz çizgi, dikdörtgen, yuvarlatılmış dikdörtgen ve elipsten oluşan temel şekiller kümesinin çizilmesini içerir. Bu görevde kullanıcıların temel şekil sınıflarını ve çizdirme fonksiyonunu keşfetmeleri beklenmektedir.

2. Görev: 1. Görevde verilen şekillerin kırmızı ile çizilerek içlerinin doldurulmasını içerir. Bu görevde, kullanıcıların renk özelliğini ve bir şeklin içini doldurma fonksiyonunu keşfetmeleri beklenmektedir.

3. Görev: Bir rastgele şekil çizdirilmesini içerir. Bu görev, diğer iki göreve göre daha kısa olsa da, kütüphane üzerinde ilgili sınıfın keşfi daha zor olduğu için diğer iki görevle eşit puanlandırılmaktadır.



Şekil 5.1. Kullanıcı Deneylerinde Gerçeklenen Görevler

5.1.2. Java 2D API'nin Sezgisel Değerlendirmesi

Tez çalışması öncesinde gerçekleştirilen Seminer II çalışmasında, Java 2D API'nin kullanılabilirlik açısından bir değerlendirmesi yapılmıştı. Bu değerlendirmeye göre, seçilen 3 başlıkta Java 2D API'nin durumu aşağıdaki gibidir.

a) İsimlendirme

Java SE içinde yer aldığı için tüm Java konvansiyonlara uygundur. Sınıfların, metotların ve parametrelerin isimlendirmesi, kendi kendini ifade eder niteliktedir. Açıktır ve tutarlıdır.

b) Parametrelili Yaratıcı Metot

Tüm sınıflarda parametrelili yaratıcı metot bulunmaktadır. Bu metotlar ilgili sınıftan bir nesne yaratarak, verilen 'gerekli' parametreleri nesneye atamaktadır.

Örneğin, bir Rectangle2D nesnesi yaratmak için, bir dikdörtgeni kullanıcı koordinat uzayında tanımlayan sol üst köşesinin koordinatları ile dikdörtgenin eni ve boyu parametre olarak sağlanmalıdır (Bkz. Şekil 5.2).

```
new Rectangle2D.Double(x, y, rectWidth, rectHeight);
```

Şekil 5.2. Parametrelili yaratıcı metot ile nesne yaratımı

Java 2D API'de, şekil sınıfları üzerinde alanlara atama yapmak için kullanılan atayıcılar ('setter') bulunmamaktadır. Bu nedenle, Java 2D'nin nesne yaratımının alternatifli, parametresiz yaratıcı metot ve yanında atayıcılar sunulmasıdır (Bkz. Şekil 5.3).

```
Rectangle rectangle = new Rectangle();
rectangle.setX(x);
rectangle.setY(y);
rectangle.setWidth(rectWidth);
rectangle.setHeight(rectHeight);
```

Şekil 5.3. Parametrelili yaratıcı metot ile nesne yaratımı

c) Metot yerleşimi

Java 2D API'nin tümü ele alındığında, metot yerleşimi tercihi değişiklik göstermektedir. Kimi zaman metotlar, doğrudan ana sınıf üzerinde ya da ana sınıfın işaret ettiği sınıfın üzerine iken, kimi zamansa ana sınıftan herhangi bir referans barındırmayan bir sınıfın üzerinde yer almaktadır. Temel şekillerin çizdirilmesi ve boyanması işlevi, literatürde [Stylos and Myers, 2008] tavsiye edilenin aksi şekilde yapılmaktadır. Bir başka deyişle, kullanıcının ihtiyaç duyacağı çizdirme ve boyama metotları, şekil sınıfları üzerinde değil, bu şekil sınıfları tarafında hiçbir şekilde işaret edilmeyen başka bir sınıf üzerinde bulunmaktadır.

Örneğin, bir düz çizgi çizilmesi için bir `Line2D` nesnesi yaratıldıktan sonra, bu çizgi çizdirilmek istendiğinde, Şekil 5.4'te gösterildiği gibi yaratılan çizgi üzerinden `draw()` metodunun çağrılması yerine; çizdirmenin yapılacağı panele ait olan `Graphics2D` nesnesi üzerinden bu metot çağrılmak zorundadır (Bkz. Şekil 5.5). `Graphics2D` sınıfı, `Line2D` sınıfı tarafından işaret edilmediği için, keşfedilmesi zaman alacaktır.

```
public void paint(Graphics g) {
    // draw a Line on this panel
    Line2D line = new Line2D.Double(x1, y1, x2, y2);
    line.addToPanel(thisPanel);
    line.draw();
}
```

Şekil 5.4 Ana sınıf üzerinden metot çağrısı

```
public void paint(Graphics g) {
    // draw a Line on this panel
    Line2D line = new Line2D.Double(x1, y1, x2, y2);

    Graphics2D g2 = (Graphics2D) g;
    g2.draw(line);
}
```

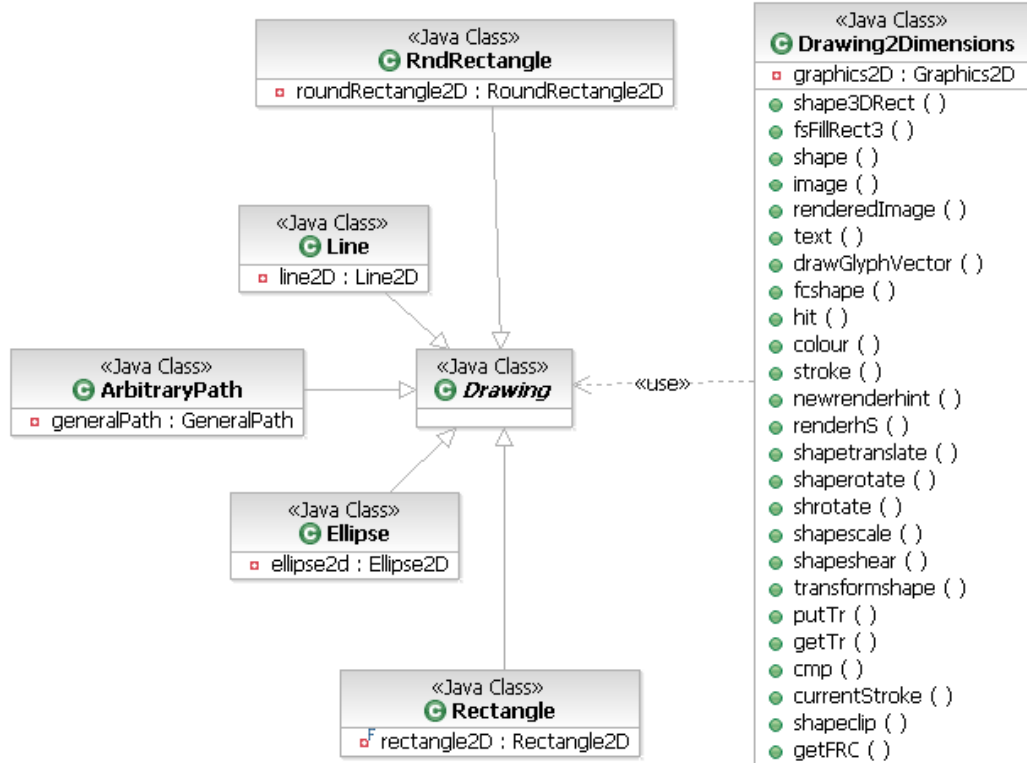
Şekil 5.5 Dolaylı sınıf üzerinden metot çağrısı

5.2. Karşılaştırılacak UPA'ların Üretilmesi

Önceki bölümde belirtildiği gibi, Java 2D API ile isimlendirme, parametrelili yaratıcı metot ve metot yerleşimi başlıkları altında karşılaştırılacak olan UPA'lar, tez çalışması kapsamında üretilmiştir. Bölüm 5.1.2'de verilen sezgisel değerlendirme sonuçlarına dayanarak üretilmesi gereken UPA'ların özellikleri belirlenmiştir. Bu bölümde üretilen UPA'lar tanıtılmaktadır.

5.2.1. Kötü İsimlendirme

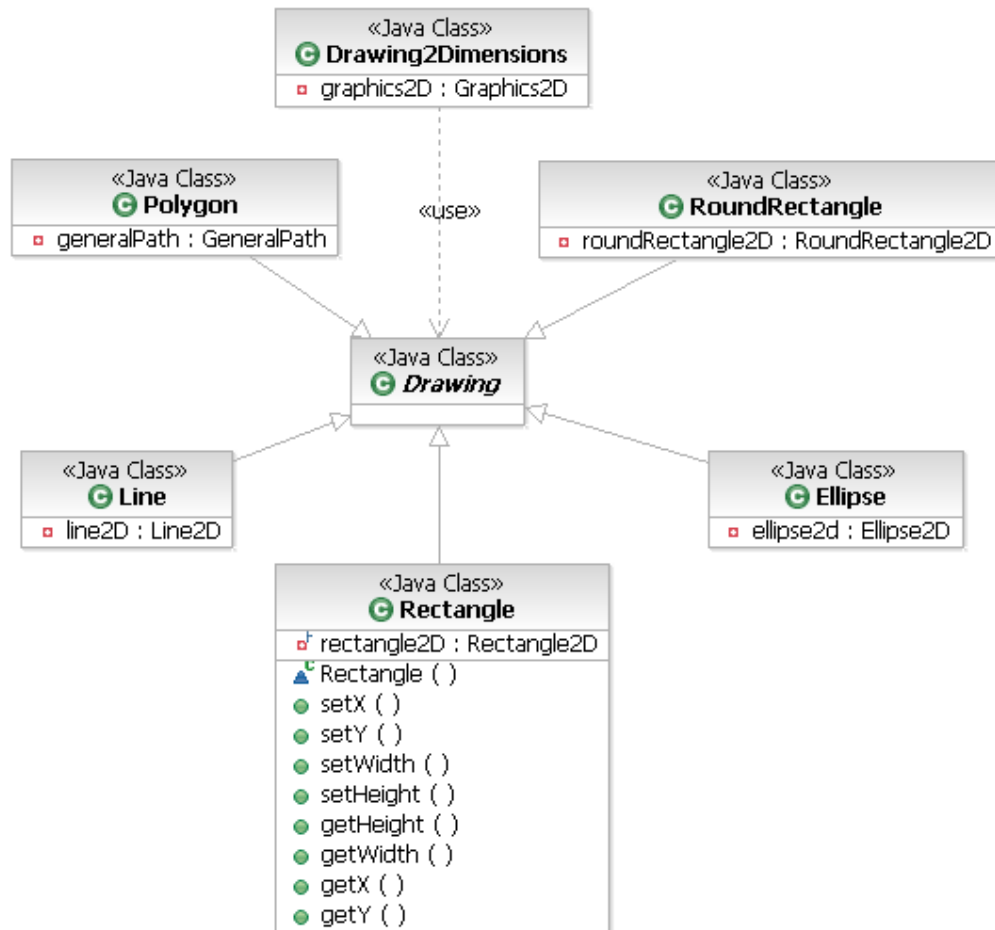
Java 2D API'de isimlendirme başarı ile yapıldığından, karşılaştırılacak UPA'da kötü isimlendirme yapılması gerekmiştir. Sınıf adlarını kötü isimlendirmek kullanıcıları tamamen çözümsüz bırakacağı için, metot isimlerinin parametre isimlerinin kötü isimlendirilmesiyle yetinilmiştir. Şekil 5.6'da bu UPA'daki temel sınıflar ve metotları içeren bir sınıf diyagramı verilmiştir. Üretilen UPA'ya, kötü isimlendirme UPA'sı anlamına gelen “*Bad Naming API*” adı verilmiştir. İsmi kullanıcıları yönlendirmesi endişesiyle “BN API” olarak kısaltılmıştır.



Şekil 5.6. BN API – Basitleştirilmiş Sınıf Diyagramı

5.2.2. Parametresiz Yaratıcı Metot

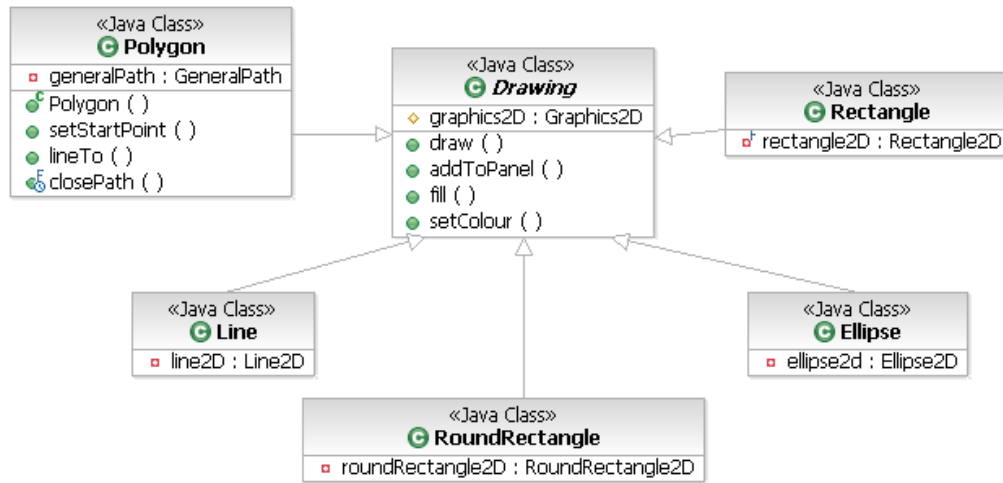
Java 2D API’de, deney kapsamına alınan her sınıfta parametrelili yaratıcı metot bulunduğundan, yaratıcı metotta parametre kullanımının etkilerini gözleyebilmek amacıyla yeni UPA’nın tüm sınıflarında yalnızca parametresiz yaratıcı metot kullanılmıştır. Şekil 5.7’da bu UPA’daki en temel sınıfları ve metotları içeren bir sınıf diyagramı verilmiştir. Diyagramı sadeleştirmek amacıyla, yalnızca Rectangle sınıfında tüm alanların ilklendirilmesini sağlayan metotlar gösterilmiştir. Benzer atayıcılar diğer tüm sınıflarda da yer almaktadır. Bu UPA, parametresiz yaratıcı metotlarına istinaden “*Parameterless Constructor API*” olarak adlandırılmıştır. Kullanıcılara ipucu vermemek amacıyla, “PC API” olarak kısaltılmıştır.



Şekil 5.7. PC API – Basitleştirilmiş Sınıf Diyagramı

5.2.3. Ana Sınıf Üzerinde Metot Yerleşimi

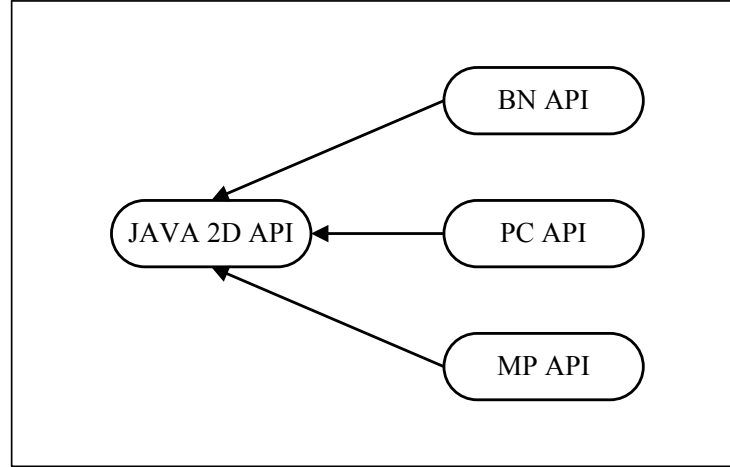
Java 2D API'nin sezgisel değerlendirmesinde de belirtildiği gibi, deney kapsamına alınan en temel işlevler (şekil çizdirme, şekil boyama gibi), ana sınıf üzerinde değil, keşfedilmesi zor olan bir sınıf üzerinde bulunmaktadır. Metot yerleşiminin etkisini görebilmek ve bu konuda oluşturulan hipotezi sınamak amacıyla, karşılaştırılacak UPA'da, en temel metotlar sınıfın üzerine konmuştur. Şekil 5.8'de görüldüğü gibi draw(), fill(), setColour() gibi metotlar tüm şekil sınıflarının üst sınıfı olan Drawing sınıfı üzerinde bulunmaktadır. Bu yeni UPA, metot yerleşimine işaret ederek “*Method Placement API*” olarak isimlendirilmiş ve “MP API” olarak kısaltılmıştır.



Şekil 5.8. MP API – Basitleştirilmiş Sınıf Diyagramı

5.3. Sıralama

Bölüm 4.2’de belirtilen her üç UPA, Java 2D API ile karşılaştırılmak üzere üretilmiştir (Bkz. Şekil 6). Her kullanıcı ile yapılan deney sonucunda üç adet karşılaştırma verisi elde edilmesi tasarlanmıştır (Java – BN; Java – MP; Java - PC). Öğrenmenin sonuç üzerindeki etkisini en aza indirmek için, kullanıcıların yarısı, Java 2D ile gerçeklemeyi ilk sırada, kalan yarısı ise son sırada yapmalıdır. Böylece 6 kullanıcıdan Java – BN; Java – MP; Java – PC, kalan 6 kullanıcıdan ise BN - Java; MP - Java; PC – Java karşılaştırma verileri gelecektir.



Şekil 5.9. Karşılaştırılacak UPA'lar

Java 2D ile gerçeklemeyi ilk ya da son sırada yapan altışar kişilik gruplar içinde de, sıralama etkisini en aza indirmek amacıyla, üretilen Üç UPA'nın sırası değiştirilmiştir. Üç UPA'nın $3! = 6$ adet diziliminin her biri, 6 kişilik gruptan bir kişiye verilecek şekilde UPA sıralamaları ayarlanmıştır. Çizelge 5.1'de, deneylere katılan 12 kullanıcıdan her birinin hangi sıra ile UPA'ları kullanacağı gösterilmektedir.

Çizelge 5.1. Her kullanıcı için UPA sıralaması

	1	2	3	4
1. kullanıcı	JAVA	BN	MP	PC
2. kullanıcı	JAVA	BN	PC	MP
3. kullanıcı	JAVA	MP	BN	PC
4. kullanıcı	JAVA	MP	PC	BN
5. kullanıcı	JAVA	PC	BN	MP
6. kullanıcı	JAVA	PC	MP	BN
7. kullanıcı	BN	MP	PC	JAVA
8. kullanıcı	BN	PC	MP	JAVA

9. kullanıcı	MP	BN	PC	JAVA
10. kullanıcı	MP	PC	BN	JAVA
11. kullanıcı	PC	BN	MP	JAVA
12. kullanıcı	PC	MP	BN	JAVA

6. UPA DEĞERLENDİRMESİ

Bu bölümünde, yöntem kısmında tanımlandığı şekilde oluşturulan uygulama programlama arayüzlerinin, değerlendirilmesine ait detaylar anlatılmaktadır.

6.1. Örneklem

Her 3 hipotez için 12’şer deney yapılmıştır. Tüm deneyler için, deney örnekleme 12 kişiden oluşmaktadır. Her katılımcı toplam 4 UPA üzerinde verilen görevleri gerçeklemiştir.

Deneylerin tamamı Türkiye’de gerçekleştirilmiştir. Tüm kullanıcıların anadili Türkçedir. Bununla beraber, tüm UPA’lar ve dokümantasyon İngilizce olduğu için, deney kullanıcıları İngilizceyi orta ve üzeri bilen insanlardan seçilmiştir. Kullanıcıları bulmak için bir elektronik ilan hazırlanmış ve birkaç yazılım e-posta grubuna postalanmıştır.

2’si bayan 10’u erkek olan kullanıcıların yaşları 25 ile 32 arasında değişmektedir. Yaş ortalaması 28,7’dir (standart sapma: 2,14).

Kullanıcıların tamamı Java bilgisine sahiptir. Deney için kullanıcı seçerken adayların temel Java bilgisini sınavan ufak görevleri gerçeklemeleri istenmiştir. Java tecrübeleri 3 ile 8 yıl arasında değişmektedir. Java tecrübesi ortalaması 4.6 yıldır (standart sapma: 1,7).

Katılımcıların tamamı, hali hazırda yazılım geliştirici olarak çalışmaktadır. 2 tanesi yüksek lisans, 1 tanesi doktora çalışmalarına işin yanı sıra devam etmektedir. Katılımcıların programlama tecrübeleri lisans eğitiminin üçüncü yılından itibaren hesaplanmıştır ve ise 5 yıl ile 10 yıl arasında değişmektedir. Programlama tecrübesi ortalaması 8,7 yıldır (standart sapma: 2,14).

Katılımcıların son 5 yıldır hiçbir grafik kütüphanesi kullanmamış olması koşulu aranmıştır. Tüm katılımcılar bu koşulu sağlamaktadır.

Katılımcılara, deney başlamadan önce, tüm kullanılabilirlik testlerinde yapıldığı gibi, bir karşılık verilmiştir. Bu karşılık, teşekkür ifadesi olmakla birlikte, katılımcıların yapılan çalışmaya daha istekle katılmasını ve daha dikkatli olmalarını sağlamaktadır. Her katılımcıya, tez yazarının seçtiği bir kitap armağan edilmiştir.

6.2. Deney Ortamı

Deneyler, kullanıcılara uygun olan çeşitli mekânlarda gerçekleştirilmiştir. Kimi deneyler Gebze Yüksek Teknoloji Enstitüsü, Bilgisayar Mühendisliği Bölümü binasındaki insan-bilgisayar etkileşimi laboratuvarında gerçekleştirilmiş, kimi ise deney süresi uzun olduğu için katılımcının kendi çalışma alanında yapılmıştır.

Deneyleri birden fazla mekânda gerçekleştirebilmek için deney bilgisayarı olarak ASUS F3FM, 2GB, Intel Core2 Duo, 15” bir dizüstü bilgisayar kullanılmıştır. Dokunmatik yönlendiriciye (“touch-pad”) alışık olmayan kullanıcılar için fare de sağlanmıştır.

Deneyler sırasında, masaüstü video kaydedici bir program ile katılımcının bilgisayarda yaptığı tüm işlemler kaydedilmiştir. Ayrıca üç-ayak ile sabitlenmiş bir video-kamera ile tüm deney kayıt altına alınmıştır. Yapılan kayıtlar, ölçümlerin kontrol edilmesini ve eksik olanların kayıtlardan toplanmasını sağlamıştır. Kullanıcıların kamera önünde rahat etmelerini sağlamak amacı ile kayıtların 3. kişiler ile paylaşılmayacağı ifade edilmiştir.

Kullanıcılar görevlerini yerine getirirken, tez yazarı gözetmen ilgili ölçümleri yapmış ve bazı notlar almıştır. Kullanıcılar istedikleri zaman soru sorabilmişlerdir. Bu soruların küçük bir bölümü yanıtlanmıştır, fakat sorulan sorulardan kullanıcıların tereddütleri anlaşıldığı için, soru sormak sözlü olarak teşvik edilmiştir.

Kullanıcıların verilen görevleri gerçeklemek için her UPA başına yarım saattir vardır. Her yarım saatlik oturumun ardından istendiği kadar ara verilir. Gerçekleme Java programlama dili ve Eclipse aracı kullanılarak yapılır. Kullanıcının proje oluşturma, ilgili kütüphaneyi sınıf yoluna (“*classpath*”) ekleme gibi işlerle zaman harcamaması için dört proje oluşturulmuştur. Her proje içinde *APIUsabilityTest.java*

isimli bir sınıf bulunmaktadır. Bu sınıf, çizimleri içerecek bir paneldir, ve `javax.swing.JPanel` sınıfından türemektedir. Sınıfta kullanıcının her görev için kod eklemesi gereken yerler yorumlarla belirtilmiştir. Koordinat hesapları ile ilgili ipuçları brifingde verilmekle beraber sınıf içinde yorumlarla da desteklenmiştir.

Kullanıcıların internete erişimine izin verilmemiştir. Dokümantasyon olarak sadece Javadoc'tan faydalanmaları sağlanmıştır.

6.3. Deney Aşamaları

Katılımcılar seçildikten ve ortam ayarlandıktan sonra deneyler yapılmıştır. Katılımcılara deneye başlarken bir kılavuz verilmiştir (Bkz. Ek-1 Deney Kılavuzu). Kılavuzda da belirtildiği gibi deney üç aşamadan oluşmaktadır:

1. Yalancı kod ('pseudo code') ile görevleri gerçekleştirme
2. Verilen UPA ile görev gerçekleştirme -Her dört UPA için tekrarlanır-
3. Memnuniyet anketi -Her dört UPA için tekrarlanır- (Bkz. Ek-2 Anket)

Kılavuz kullanıcı tarafından okunduktan sonra, deney hakkında bir brifing verilmiştir.

6.3.1. Giriş Brifingi

“Öncelikle deneyimize katılarak çalışmalarımıza yön verdiğiniz için teşekkür ederiz.

Çalışmamızda, farklı özelliklere sahip uygulama programlama arayüzlerini karşılaştırarak, kullanımı sizlere en doğal ve kolay gelen UPA'ların ortak özellikleri toplanmaktadır. Deneyler sırasında kullanacağınız her dört UPA, sizi memnun etmeleri ve rahat programlama imkânı vermeleri konusunda değerlendirilecektir. Bir kez daha, teste tabi tutulanan kullanacağınız kütüphaneler olduğunu belirtmek isterim.

Kılavuzda da belirtildiği gibi, deney üç aşamadan oluşmaktadır. İlk aşamada kılavuzda okuduğunuz görevleri, ihtiyaç duyduğunuz kütüphaneler mevcutmuş gibi

size verilen boş kağıda, ya da masaüstünde bulunan *pseudocode.txt* dosyasına yalancı kod ile gerçeklemeniz istenmektedir. İkinci aşamada, 4 farklı kütüphane ile aynı görevler gerçekleştirilecektir. Her birisi için Eclipse’te birer proje oluşturulmuştur ve görevlerin gerçekleştirilmesi için yarımşar saat süre sınırlaması getirilmiştir. Projelerin hangi sırayla tamamlanacağı size söylenecektir. Her bir proje tamamlandığında, ilgili kütüphanenin anketi size sunulacaktır. Anketin doldurulması deneyin üçüncü aşamasıdır.

Lütfen görevleri gerçeklerken aklınızdan geçenleri sesli olarak ifade ediniz. Olumlu ve olumsuz eleştirilerinizi dile getirmek için anketi beklemeyiniz. Deneyler boyunca dilediğiniz kadar soru sorabilirsiniz.”

6.4. Ölçüm

Deney sırasında gözetmen tarafından bazı ölçümler toplanmıştır. Toplanan ölçümlerin kontrolü için kaydedilen görüntüler bir kez daha izlenmiş ve gerekli yerlerde düzeltmeler yapılmıştır. Deneylerde her UPA için toplanan sayısal metrikler şunlardır:

- 1.Tamamlanan görev sayısı: Kullanıcıya verilen üç görevden kaçının başarı ile tamamlandığı
- 2.Tamamlama süresi (dakika): Tüm görevlerin tamamlanma süresi (görevlerin tümünü 30 dakikadan önce tamamlayan kullanıcılar için hesaplanır)
- 3.Hatalı çalıştırma sayısı: Verilen UPA ile görev gerçekleştirme boyunca yapılan hatalı çalıştırma sayısı
- 4.Sorulan soru sayısı: Kullanıcının UPA ile ilgili sorduğu soru sayısı (sadece yanıtlananlar hesaplanır)
- 5.Ankette yer alan UPA sıralaması: Kullanıcının UPA’ları 1. ile 4. Arasında sıralaması sonucu UPA’yı uygun gördüğü sıra (Bkz. Ek-2, Soru:12)

7. SONUÇLAR

7.1. Yalancı Kod Aşaması Sonuçları

Deneylerin ilk aşamasında katılımcılara yazdırılan yalancı kodlardan aşağıdaki sonuçlar çıkmıştır:

1. Katılımcıların tümü nesne yaratımlarını parametrelili yaratıcı metot ile yapmışlardır. Nesnelere atanmasını bekledikleri özellikleri, nesne yaratma sırasında yaratıcı metoda vermişlerdir.
2. 12 kullanıcıdan 9'u çizdirme ve renk değiştirme gibi görevleri doğrudan nesnenin üzerinde bekleyerek, ilgili metotları nesneler üzerinden çağırmışlardır.
3. 12 kullanıcıdan 1'i çizdirme ve içeri doldurma için “çiz” ve “dolu çiz” kelimelerini kullanmış, bir tanesi ise bu kelimelerin İngilizce karşılıkları olan “draw” ve “fill” kelimelerini kullanmıştır.

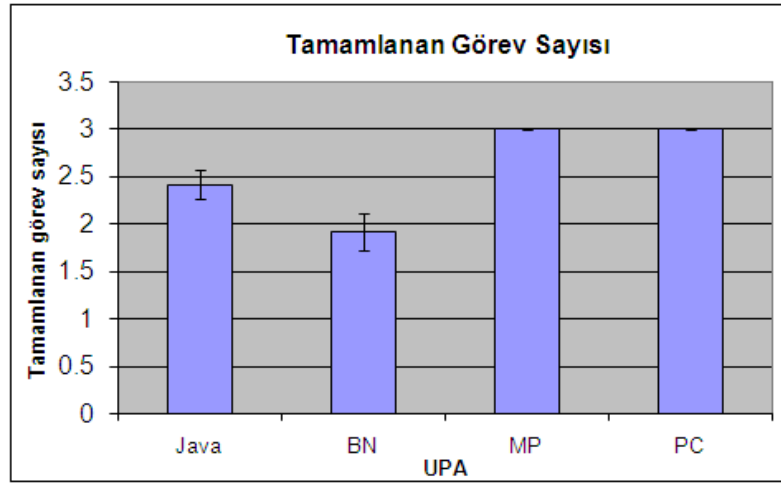
7.2. Görev Gerçekleme Aşaması Sonuçları

Kullanıcı deneylerinin ikinci aşaması olan görev gerçekleştirme sırasında toplanan ölçümlerin ortalaması Çizelge 7.1’de verilmiştir. Çizelgenin her satırında, dört UPA’ya ait sonuçlar verilmiştir. Çizelgede verilen sonuçlar, standart hata değerleri ile birlikte aşağıdaki 4 grafikte sunulmaktadır (Bkz. Şekil 7.1 – Şekil 7.5).

Çizelge 7.1. Ortalama Deney Sonuçları

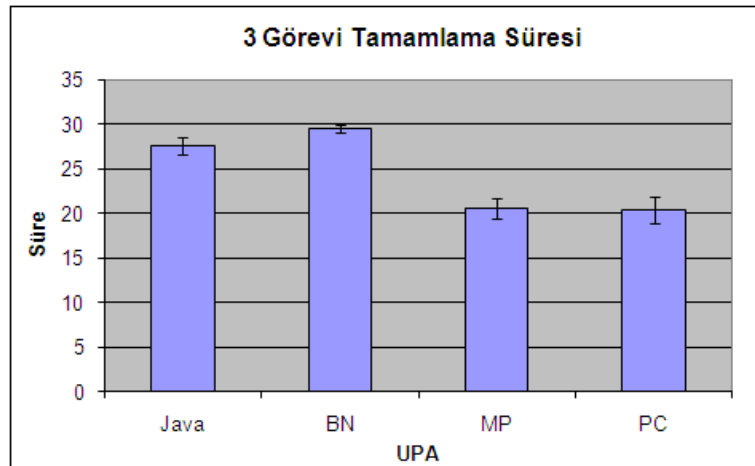
	JAVA	BN	MP	PC
Tamamlanan görev sayısı	2.416667	1.916667	3	3
Tamamlama süresi	27.6	29.5	20.5	20.41667
Soru sayısı	1.916667	2.333333	1.083333	0.916667
Hatalı çalıştırma sayısı	1.5	3.583333	0.75	0.833333
Anket sırlaması	2.916667	3.916667	1.25	1.916667

Kullanıcıların tamamı Java 2D API ile gerçekledikleri 3 görevin 2 ya da 3'ünü bitirebilmişlerdir (ortalama 2.4). BN API ile ortalama 1,9 görev tamamlayabilirken, 12 kullanıcının yalnızca ikisi tüm görevleri tamamlayabilmiş, kalan kullanıcılar 1 ya da 2 görevi bitirebilmişlerdir. MP ve PC'de, tüm kullanıcılar tüm görevleri tamamlayabilmişlerdir (Bkz. Şekil 7.1).



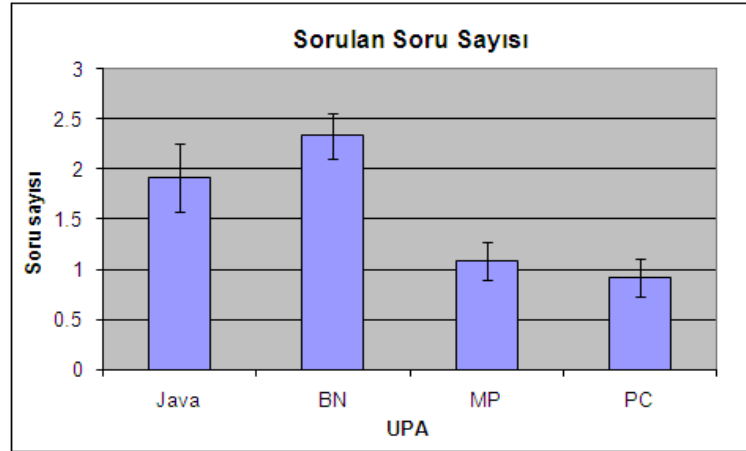
Şekil 7.1. Tamamlanan görev sayısı ortalamaları

Java 2D API ile tüm görevleri tamamlayan 5 kullanıcı, bu görevleri ortalama 27,6 dakikada bitirebilmişlerdir. BN API ile tüm görevleri tamamlayan 2 kullanıcı da görevleri son dakikada tamamlayabilmiştir. MP ve PC'de ise tüm görevler ortalama olarak 20,5 dakika da tamamlanabilmişlerdir (Bkz. Şekil 7.2).



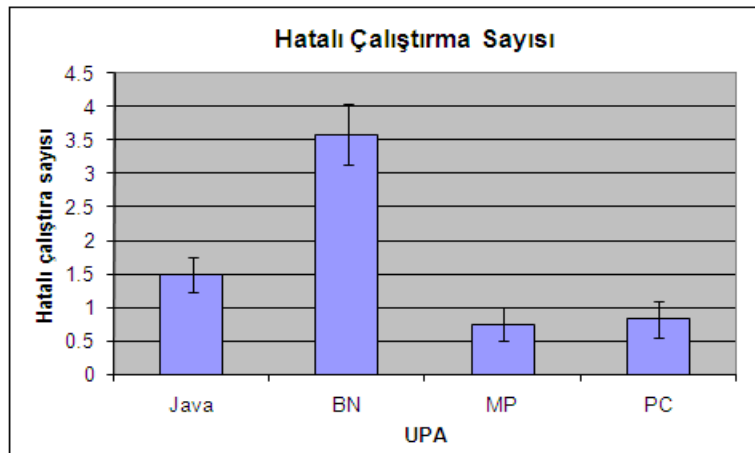
Şekil 7.2. Tüm görevleri tamamlayan kullanıcıların tamamlama süresi ortalamaları

Deneyler sırasında tüm kullanıcıların soru sorması serbest bırakılmıştır. Bu sorulardan çözümü doğrudan sormayanlar yanıtlanmıştır. Bu şekilde, kullanıcıların yanıt bulan sorularının ortalaması Şekil 7.3'te sunulmuştur.



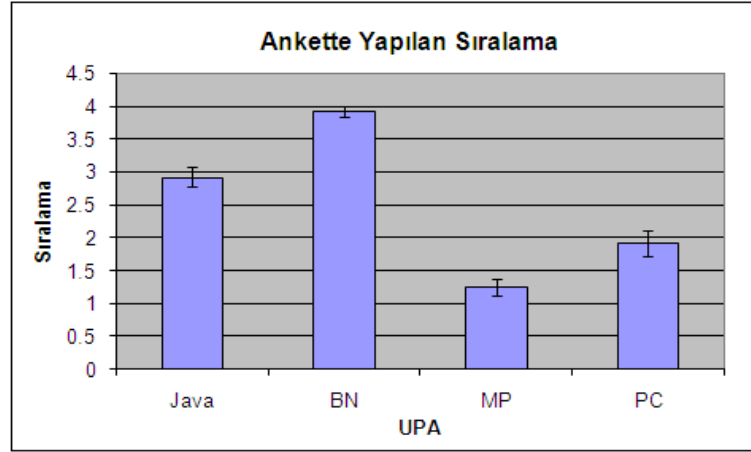
Şekil 7.3. Kullanıcıların sordukları soruların ortalamaları

Deneyler sırasında, kullanıcılar geldikleri aşamaları sınamak için istedikleri kadar çalıştırma yapabilmişlerdir. Bu denemeler sırasında yaptıkları hatalar ölçülmüş ve Şekil 7.4'te sunulmuştur. En çok hata yapılan UPA, 3.6 ortalama ile BN API olurken Java 2D API kullanımı sırasında ortalama 1.5 hata alınmıştır. MP ve PC API'lerde ise, ortalama hata sayısı 1'in altındadır.



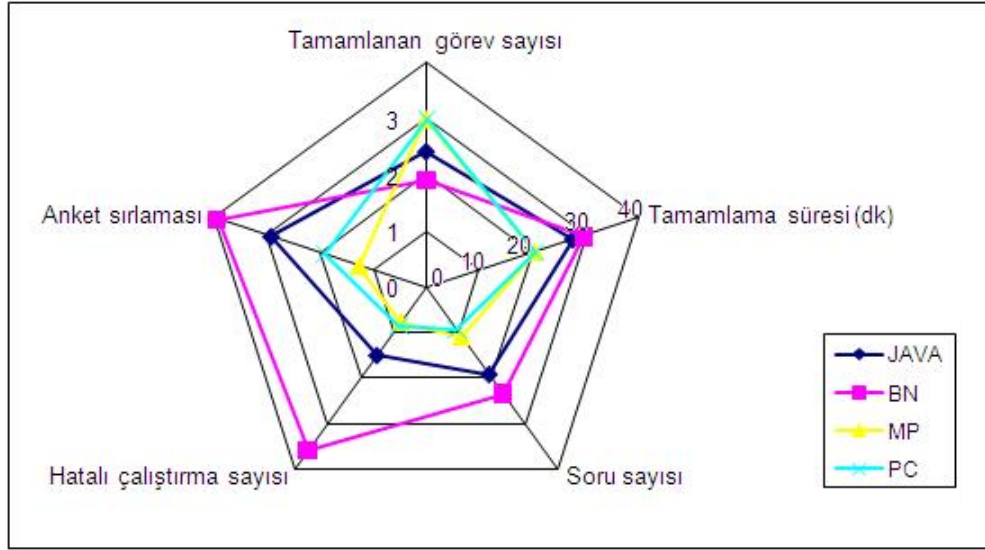
Şekil 7.4. Kullanıcıların denemelerindeki hata ortalamaları

Son olarak verilen sayısal sonuç, kullanıcıların UPA'ları sıralamalarından elde edilen ortalama sıra değeridir. 12 kullanıcıdan 11'i, BN API'yi kullanılabilirlik açısından en kötü UPA olarak 4. sıraya yerleştirmişlerdir. Kullanıcıların çoğu, Java2D API'yi üçüncü sıraya, PC API'yi ise ikinci sıraya uygun görmüşlerdir. MP API ise 12 kullanıcıdan 8'i tarafından birinci, kalan 4'ü tarafından ise ikinciliğe layık görülmüştür (Bkz. Şekil 7.5, Şekil 7.6).



Şekil 7.5. Kullanıcıların UPA'ları sıralamasından elde edilen sıralama ortalaması

Aşağıdaki grafikte (Bkz. Şekil 7.6) tüm ölçüm maddeleri üzerinden bir karşılaştırma yapmak daha kolaydır. Lacivert renkli JAVA 2D API, her ölçüm başlığında kötü isimlendirme ile elde edilmiş olan BN API'den daha iyi, parametresiz yaratıcı metot kullanımını teşvik eden PC API ile metot yerleşimini ana sınıf üzerinden gerçekleştiren MP API'den daha kötü çıkmıştır.



Şekil 7.6. Deney sonuçları özeti

7.3. Anket Sonuçları

1. Yapılan anketlerin tamamında, kullanıcılar BN API'deki isimlendirme problemlerine dikkat çekmiş ve kullanımının zor olduğunu açıkça belirtmişlerdir. Parametre isimlerinden ne olduklarının anlaşılmadığı ve doğru metotları bulurken zorlandıklarını yorumlara eklemişlerdir. UPA'nın kendilerini bazı hatalara sıklıkla düşürecek şekilde tasarlandığı sorusuna kullanıcıların tamamı "Fazla" ya da "Çok fazla" olarak yanıt vermişlerdir.

2. PC API'nin anketlerinde kullanıcıların tamamı, satır sayısı ile ilgili soruya "Uzun" ya da "Çok uzun" yanıtını vermişlerdir. Bu olumsuzluğa rağmen 12 kullanıcıdan 10'u UPA ile olumlu görüş bildirmiş, yazılan kodun anlaşılabilirliğinin çok yüksek olduğunu ve değişikliğin çok kolay yapılabildiğini ifade etmişlerdir.

3. MP API'nin anketlerinde, 12 kullanıcıdan 10'u çizdirme, boyama gibi metotların ana sınıf üzerinde bulunmasını beğendiklerini açıkça ifade etmişlerdir. Bu şekilde nesneye yönelimli programlamaya daha uygun olduğu yorumu yapılmıştır. 12 kullanıcıdan 1'i, metotların nesne üzerinde değil panel üzerinde olması gerektiğini, diğer 1 tanesi ise nesne veya panel değil, ayrı bir çizim sınıfı üzerinde olmasını daha doğru bulduğunu belirtmiştir.

8. SINIRLAMALAR

Deneyler yapılarak elde edilen sonuçlar, hipotezlerle uyumludur. Ancak, deneyler tasarlanırken bazı sınırlamalar olacağı öngörülmüştür. Sınırlamaların bazıları ise deneyler sırasında ortaya çıkmıştır. Tezde ortaya konan sonuçların doğru genellenebilmesi için kapsam ve potansiyel sınırlamaların da sunulması önem taşımaktadır.

8.1. Deney Kapsamı

Deneylerde 4 UPA ile gerçekleşmesi gereken her üç görev de, günlük hayattaki programlama görevlerinden oldukça kısadır. Bu sayede daha çok görev gerçekleştirerek daha çok test verisi elde etme fırsatı bulunmuştur. Diğer taraftan belirtilen etkenlerin uzun ve karmaşık görevlerdeki etkisi bilinmemektedir.

Günlük hayatta yazılım geliştirme yaşam döngüsü boyunca gereksinimlerin değişmesine sıklıkla rastlanmaktadır. Değişen gereksinimlerle birlikte daha önceden yazılmış olan kodun değiştirilmesi de kaçınılmaz bir durumdur. Bu nedenle UPA'ların değişiklik yapmayı zorlaştırmaması gerekmektedir. Ancak testlerde değişiklik yapma görevleri verilmemiştir. Değişiklik ve hata ayıklama görevlerinde sunulan etkenlerin etkisi araştırılmamıştır.

Deneye katılan kullanıcılarda son 5 yıldır grafik kütüphanesi kullanmamış olması koşulu aranmıştır. Kullandıkları kütüphanelerdeki yerleşik konvansiyonların zihinsel modeli etkilemesinden kaçınmak için deneyin amacına uygun bir tercih olduğu düşünülmektedir. Bununla beraber, deney konusu üzerinde çalışan kullanıcılar üzerindeki etkiler hakkında bir şey söylememektedir.

8.2. Sıralama Etkisi

Deneylerde, kullanıcılardan aynı görevi dört UPA kullanarak gerçeklemleri istenmiştir. Kullanıcıların önce kullandıkları UPA'dan öğrendiklerini, takip eden UPA'larda da arayacağı ve bu durumun sonucu etkileyeceği öngörüldüğünden, UPA sıralaması, her kullanıcı için değiştirilmiştir. Olası tüm kombinasyonları

deneyebilmek için, deney sayısı 10'dan 12'ye çıkarılmıştır. UPA sıralamasının değiştirilmesinin denekler-arası sonuçları geçerli hale getirdiği düşünülmektedir.

8.3. Dil

Deney katılımcılarının tümünün anadili Türkçedir. UPA'lar ve dokümantasyon İngilizce olduğu için, deney kullanıcıları İngilizce bilen insanlardan seçildiyse de, deneyler sırasında kullanıcılardan gelen bazı sorulardan tüm katılımcıların dil becerilerinin aynı olmadığı görülmüştür. Bu durum, bazı kullanıcılara zaman kaybettirmiştir.

8.4. Deney Ortamı

Deneyler farklı mekânlarda yapıldığından, tümünde bir dizüstü bilgisayar kullanılmıştır. 15" olan ekran boyutu, günümüzde ofislerde kullanılan monitörlerin boyutundan küçüktür. Ekran boyutunun yanında, klavye dizilimi de bazı kullanıcıları rahatsız etmiştir. Fakat karşılaştırılan tüm UPAlar aynı makinede kullanıldığı için denekler arası sonuçlarda bir etki yaratmadığı düşünülmektedir.

Deneylerde kullanılan dört UPA'dan üçü tez çalışması kapsamında geliştirildiğinden, internette bunlarla ilgili örnek kod ya da başka dokümantasyon bulunmamaktadır. Günümüzde yazılım geliştirme sırasında internet kullanımı yaygın bir alışkanlık haline geldiyse de, belirtilen nedenle deneyler boyunca internet erişimi sağlanmamıştır.

Programlama görevleri Java ile gerçekleştirileceğinden ve internet kullanımına izin verilmediğinden, kullanıcılara düz Javadoc formatında dokümantasyon sağlanmıştır. Farklı bir dokümantasyon biçimi kullanılması, doğru metot yerleşiminin yarattığı farkı değiştirebileceği eklenmelidir. Örneğin Jadeite [Stylos et al, 2009], bir kütüphanede metot yerleşimi kötü bile yapılmış olsa, yazılım geliştiricilere başka meslektaşlarının sağladığı verilerle doğru yolu göstermektedir.

9. TARTIŞMA

Yazılım sektöründe hem iş hacmi hem de yapılan işlerin karmaşıklığı gün geçtikçe artmaktadır. Her geçen gün yeni gereksinimler ortaya çıkmakta ve yeni iş süreçleri elektronik ortama taşınmaktadır. Bu koşullar altında yeniden kullanım kaçınılmaz hale gelmektedir. Günümüzde bir yazılımcı, program geliştirirken her özelliği kendisi gerçeklemeye gerek duymamaktadır. İşin önemli bir kısmı var olan fonksiyonları kullanmak ve birleştirmektir. Üretilen ve ihtiyaç duyulan yazılım miktarının artmasıyla, UPA'lar ile var olanın yeniden kullanılması daha da artacaktır. Fakat bir UPA kullanılabilir değilse, yeniden kullanılabilir de olamaz.

UPA'ları kullanan insanların sayısı, geliştiren insanların binlerce katına ulaşmaktadır. Bu nedenle, kullanıma girmesinden sonra bir UPA'yı değiştirmek, çok sayıda projeyi etkileyeceğinden, oldukça zordur. Kullanılabilirlik hesaplarını yapmak için en doğru zaman tasarım aşamasıdır. UPA geliştiricilerinin tasarım aşamasında göz önünde bulunduracakları prensiplerin bulunması ve geliştirilen UPA'ların kullanıma sunulmadan önce kullanılabilirlik değerlendirmesi yapılabilmesi büyük önem taşımaktadır.

Tez çalışması kapsamında, yazılım geliştiricilerin bilgisayarla etkileşimini etkili, verimli ve tatmin edici kılmak, bir başka deyişle insan-bilgisayar etkileşimi hedeflerini yerine getirmek için UPA tasarımında dikkat edilmesi gereken faktörler açıklanmıştır. Bunun yanı sıra, tasarlanan UPA'ların değerlendirilebilmeleri için bir çerçeve sunulmuştur. Sunulan öneriler ve değerlendirme çerçevesi, uygulama programlama arayüzleri için ortaya konulmuş olsa da, yeniden kullanılması düşünülen her programlama birimine uygulanabilir niteliktedir.

UPA kullanılabilirliğini etkileyen 11 faktör ve ilgili öneriler sıralandıktan sonra, bu faktörlerden üçü seçilmiş ve literatürde işaret edildiği yönde hipotezler kurularak tez çalışmasında sunulan değerlendirme çerçevesi ile hipotezlerin sınanması aşaması başlamıştır. İsimlendirme, metot yerleşimi ve parametrelili yaratıcı metot kullanımının kullanılabilirliğe etkilerini araştırmayı amaçlayan bu aşamada, Java'nın grafik kütüphanesi referans olarak alınmış ve seçilen üç başlıkta deneyler

yapabilmek için üç yeni UPA üretilmiştir. Deneylerin tasarlanması ve kullanıcıların seçiminin ardından 12 kullanıcı ile deneyler tamamlanmış ve sonuçlar raporlanmıştır.

Yapılan deneylerde, UPA'ların kullanılabilirliğinin karşılaştırılmasında, tamamlanan görev sayısı, görevleri gerçekleştirme süresi, hata sayısı ve kullanıcının sorduğu soru sayısının ölçülmesiyle elde edilen sayısal veriler ile yazdıkları yalancı kod ve doldurdukları anketten elde edilen zihinsel modelleri ile ilgili veriler kullanılmıştır.

Kullanıcılar kötü isimlendirme yapılarak elde edilmiş olan UPA'yı kullanırken, iyi isimlendirilmiş Java 2D API'yi kullanımlarına göre açıkça daha kötü performans göstermişlerdir. Kullanıcıların yalancı kod yazmaları sırasında kullandıkları isimlendirmenin çok benzerini Java 2D API'de bulabilmiş olmaları, bu kütüphanenin isimlendirme konusundaki başarısını ortaya koymaktadır. Deney sonucunda elde edilen sayısal verilerle de, kötü isimlendirmenin yaratabileceği olumsuz etkiler açıkça gözlenmiştir.

Kullanılabilirlik üzerindeki etkisi araştırılan ikinci etken olan metot yerleşimi konusunda, yalancı kod aşamasında kullanıcıların %75'inin, görevlerde sık kullanılan metotları ana sınıfların üzerinde beklediği görülmüştür. Görev gerçekleştirme sırasında, kullanıcıların bir UPA'yı keşfetmeye, göreve bağlı bir ana sınıftan başladıkları gözlenmiştir. Genel stratejileri, ismi nedeniyle uygun gördükleri bir sınıfın Javadoc'unu açtıktan sonra, metotlarını gözden geçirmek olmuştur. Aradıkları metodu, bu ilk sınıfta bulan kullanıcılar, hızlıca görevlerini tamamlayabilmişken, aranan metotların ana sınıf üzerinde yer almadığı kütüphaneleri kullanırken yeni sınıf aramak ve her yeni sınıfta tüm metotları kontrol etmek durumunda kalmışlardır. Bu durum da, metotların ana sınıf üzerinde yer aldığı kütüphanenin kullanımında ölçülen değerlerin daha olumlu çıkması ile sonuçlanmıştır. Böylece metot yerleşiminin doğru sınıf üzerinde yapılmasının kullanılabilirliği belirleyici bir etken olduğu da doğrulanmıştır.

Tez çalışması kapsamında kullanılabilirliğe etkisi araştırılan son etken olan parametresiz yaratıcı metot konusunda, sonuçlar ilk iki etkene göre daha şaşırtıcı çıkmıştır. Kullanıcıların tamamı, yalancı kod kullanımı aşamasında parametrelili

yaratıcı metot ile nesne yaratmayı tercih ettiđi halde, parametresiz yaratıcı metot ile nesne yaratmayı zorlayan UPA'yı kullanırken daha başarılı olmuşlardır. Yazdıkları kodun satır sayısından şikâyet etseler bile, bu şekilde yazdıkları kodun daha anlaşılır olduğunu da eklemişlerdir. Görev gerçekleştirme sırasında, yazdıkları koda daha hâkim oldukları ve daha rahat değişiklik yaptıkları da hem gözetmen tarafından gözlenmiş, hem de deney sonuçlarına açık bir şekilde yansımıştır. Kullanıcıların çoğunun, nesne yaratımı sırasında parametre verememekten şikâyet etmelerine rağmen, parametresiz nesne yaratımını da kolay ve hızlı bir şekilde yapabildikleri ve bu açıdan bir zaman kaybı yaşamadıkları fark edilmiştir. Öte yandan, parametrelili metot kullanımına izin verdiği halde, sınıf üzerinde alanların yeniden atanmasını sağlayan metotların olmayan UPA'nın kullanımı sırasında, kullanıcıların daha çok zorlandığı ve dokümantasyona başvurmak durumunda kaldığı gözlenmiştir.

Tez çalışması kapsamında kullanıcı deneyleri yapılarak, Bölüm 3.2'de sıralanan faktörlerin yalnızca üç tanesinin kullanılabilirliğe olan etkisi araştırılırken, dolaylı olarak, diğer tüm faktörlerin de etkisi gözlenebilmiştir. Bunlar ile ilgili sayısal veri sunulamamasına rağmen, deneyler sırasında yapılan gözlem, Bölüm 3.2'de sunulan faktörlerin kullanılabilirliğe olan etkisinin literatürde işaret edildiğı ve tez çalışmasında özetlendiğı yönde olduğunu doğrular niteliktedir.

Gerçek kullanıcılarla test yapmak en güvenilir sonucu vermektedir. Fakat deney için uygun kullanıcı bulmak zor ve maliyetlidir. Kullanıcı testlerinin yüksek maliyet dolayısıyla ölçeklenebilir olmadığı görülmüştür. Bu nedenle, kullanılabilirlik değerlendirmesi için, özellikle geniş kapsamlı UPA'larda destekleyici başka metotlarla birlikte kullanılması gerektiğı tecrübe edilmiştir.

Çalışmanın, UPA tasarımcılarına yardımcı olması ve yazılım kalitesi konusunda çalışan araştırmacılara ilham vermesi umulmaktadır. Gelecekteki tasarımların daha kullanılabilir hale gelmesi ve yeniden kullanılabilirliğin artması, tasarımcıların kullanılabilirliğe daha çok dikkat etmeleri ve araştırmacıların daha çok unsurun anlaşılmasını sağlamasıyla gerçekleşecektir.

10. GELECEK ÇALIŞMALAR

Çalışma kapsamında, UPA'ların kullanılabilirliğini etkileyen faktörler geniş bir literatür taraması sonucunda elde edilmiş ve listelenmiştir. Çok sayıda ve farklı UPA'lar üzerinde kullanıcı deneyleri yapılarak bu faktörlerin geçerlenmesi önemlidir. Çünkü uygulama programlama arayüzlerinin kullanılabilirliği üzerindeki çalışmalar henüz yeterince olgunlaşmamıştır. Konu üzerinde yapılacak çalışmalarla şüphesiz yeni faktörler ortaya çıkacaktır. Kullanılabilir UPA'lar geliştirmek için, akademik ve endüstriyel çevrelerden kabul gören metodolojilere ihtiyaç vardır.

Bölüm 3.2'de UPA kullanılabilirliğini etkileyen faktörler açıklanırken, yerine getirilmesi gereken pratiklere değinilmiştir. Bu pratiklerin, tasarım kılavuzları haline getirilmesi, UPA tasarımcılarına doğrudan kullanabilecekleri kaynaklar sunacaktır. Ayrıca, söz edilen pratiklerin bir kısmı, yazılım geliştirmede kullanılan tasarım kalıplarına entegre edilebilir. Böylece UPA'larda kullanılabilirlik pratikleri çok daha geniş çevrelerce uygulanmış olacaktır.

Tez çalışmasında yapılan deneyler sırasında bir takım sınırlamalar olduğu gözlenmiş ve bunlar Sınırlamalar bölümünde (Bkz. Bölüm 8) sunulmuştur. Gelecek deneylerin, bu kısıtları ortadan kaldırmaya yönelik tasarlanması ve sonuçların karşılaştırılması, verilen sınırlamaların etkisini ortaya koymak yönünde katkı sağlayacaktır. Ayrıca deneyler sonucunda benzer eğilimler gösteren kullanıcılar arasında ikinci tur deneyler yapıp birbirlerinin kodlarını değiştirmeleri istenerek de önemli sonuçlar elde edilebilir.

Testler sırasında Java Graphics 2D UPA'sının tümünü kapsamak hedeflenmemiştir. Bunun sebebi, bu UPA'nın bütün olarak kullanılabilirliğinin değerlendirilmesi değil, seçilen üç etmenin kullanılabilirliği gerçekten etkileyip etkilemediğidir. Fakat bir sistemin bütün olarak kullanılabilirliğinin değerlendirilmesi sırasında, kullanıcı testleri ile değerlendirilen işlevlerin UPA'nın ne kadarını kapsadığını ortaya koyabilecek yöntemlere ve/veya araçlara ihtiyaç vardır.

Tez çalışması sırasında 12 kullanıcıyla yapılan ve her biri ortalama iki buçuk saat süren kullanıcı testlerinin yüksek maliyetli bir değerlendirme yöntemi olduğu tecrübe edilmiştir. Özellikle büyük sistemlerin değerlendirilmesinde kullanılabilecek, maliyeti dengeli olan yeni yöntemler geliştirilmesi, kullanılabilirlik değerlendirmesinin yaygınlaşmasını sağlayacaktır.

KAYNAKLAR

- [1] Ardiç, B. ve Göktürk, M., “Kullanılabilir Uygulama Programlama Arayüzleri”, 4. Ulusal Yazılım Mühendisliği Sempozyumu – UYMS’09, s 91-97, 8-10 Ekim 2009.
- [2] Beaton, J., Myers, B. A., “Usability Evaluation for Enterprise SOA APIs”, SDSOA’2008, 2008.
- [3] Blackwell, A., Green, T., “A cognitive dimensions questionnaire optimised for users”, Proceedings of the 12th Annual Meeting of Psychology Programming Interest Group, s 137-152, 2000.
- [4] Bloch, J., “How to design a good API and why it matters”, OOPSLA’06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, s 506-507, 2006.
- [5] Carroll, J. M, Rosson, M. B., “The paradox of the active user”, Interfacing thought: Cognitive aspects of human computer interaction, 1987.
- [6] Clarke, S., “Describing and Measuring API Usability with the Cognitive Dimensions”, Cognitive Dimensions of Notations 10th Anniversary Workshop, 2005.
- [7] Clarke, S., “Measuring API Usability”, Doctor Dobbs Journal, 2004.
- [8] Checkstyle, <http://checkstyle.sourceforge.net>.
- [9] Ellis, B., Stylos, J., Myers, B., “The Factory Pattern in API Design: A Usability Evaluation”, International Conference on Software Engineering, 2007.
- [10] Fowler, M, Beck, K., “Bad smells in code” in Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.
- [11] Henning, M., “API Design Matters”, Communications of the ACM, Vol 52, s 46-56, 2009.
- [12] “ISO/IEC 9241-11 Ergonomic requirements for office work with visual display terminals (VDT)s - Part 11 Guidance on usability”, ISO/IEC 9241-11, 1998.

[13] Lewis, C., Polson, P. G. , Wharton, C., Rieman, J, Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces, Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people, p.235-242, 1990.

[14] Nielsen, J., “Usability Engineering”, Academic Press, San Diego, s 34-37, 1993.

[15] Nielsen, J., Molich, R., “Heuristic evaluation of user interfaces”, Proc. ACM CHI'90 Conf., s: 249-256, 1990.

[16] Pemberton, S., “Programmers are Humans, Too”, ACM SIGCHI Bulletin, 1996.

[17] Preece, J., Human Computer Interaction, Addison Wesley, 1994.

[18] Psychology of Programming Interest Group, <http://www.ppig.org>.

[19] Shneiderman, B., “Designing the User Interface”, Addison Wesley, 3. Baski, 1997.

[20] Stylos, J., Clarke, S, “Usability Implications of Requiring Parameters in Objects' Constructors”, Proc. ICSE, ACM Press, s 529-539, 2007.

[21] Stylos, J., Myers, B. A., “The Implications of Method Placement on API Learnability”, Symp. on the Foundations of Software Engineering FSE '08, 2008.

[22] Stylos, J., Myers, B. A., Yang, Z., "Jadeite: improving API documentation using usage information", Proceedings of the 27th international conference extended abstracts on Human factors in computing systems, s 4429-4434, Nisan 2009.

[23] Thomas T. Hewett, ACM SIGCHI curricula for human-computer interaction, ACM, New York, 1992.

[24] Zibran, M. F., “What Makes APIs Difficult to Use?”, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, Nisan 2008.

ÖZGEÇMİŞ

1983 yılında Ankara’da doğdu. Lise öğrenimini Ankara Fen Lisesi’nde 2001 yılında tamamladı. 2005 yılında Orta Doğu Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü’nden mezun oldu. 2006 yılında Türkiye Bilim ve Teknoloji Araştırma Kurumu (TUBİTAK) – Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü’nde (UEKAE) görev yapmaya başlayana kadar özel sektörde bilgisayar mühendisi olarak çalıştı. 2007-2008 öğrenim yılı güz döneminde Gebze Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü’nde yüksek lisans eğitimine başladı. 2009-2010 öğrenim yılı güz dönemini değişim öğrencisi olarak Barcelona’daki Universitat Politècnica de Catalunya’da geçirdi. Halen TÜBİTAK-UEKAE’de araştırmacı olarak görev yapmaktadır.

Yayınlar:

1. Ardıç, Burcu; Göktürk, Mehmet, “Kullanılabilir Uygulama Programlama Arayüzleri”, 4. Ulusal Yazılım Mühendisliği Sempozyumu – UYMS’09, 8-10 Ekim 2009.
2. Annunziata, Francesca; Ardic, Burcu; Denis, Xavier; Fletcher, Graham; Hafsøe, Trude; Hernández Novo, Ignacio; Jansen, Norman; Johnsen, Frank Trethan; Meiler, Peter-Paul; Owens, Ian; Sasioglu, Betül; Sliwa, Joanna; Stavnstrup, Jens; Tokuz, Akif, “SOA Challenges for Disadvantaged Grids”, 15th ICCRTS, 2010

EKLER

10.1. Ek-1: Deney Kılavuzu

Bu deneylerde siz değil, kullanacağınız uygulama programlama arayüzlerinin kullanılabilirliği test edilmektedir. Lütfen görevleri rahatça gerçekleyiniz. Deneyde elde edilen veriler ticari herhangi bir amaçla kullanılmayacaktır. Kişisel bilgiler üçüncü partiler ile paylaşılmayacaktır.

İşbirliğiniz için teşekkür ederim.

Burcu Ardiç

1. Giriş

- Lütfen öncelikle görevleri okuyup anlamadığınız bir husus varsa gözetmene sorunuz.
- Deneyin ilk aşaması yalancı kod (“pseudo code”) yazılmasıdır. Lütfen aklınızdaki kütüphane ile görevleri istediğiniz sentaks ile gözetmenin vereceği kâğıda yazınız.
- Deneyin ikinci aşamasına geçmeden önce gözetmen bir briefing verecektir. Lütfen hazır olduğunuzda gözetmene bildiriniz.
- Deneyin ikinci aşaması görev gerçeklemedir. Her 4 kütüphane için 3. bölümde belirtilen görevler gerçekleştirilmelidir.
- Deneyin üçüncü aşamasında kullanıcının bir anketi yanıtlanması istenir.

2. Briefing Özeti

Eclipse programının açık olduğundan emin olunuz. Eclipse programında “1”, “2”, “3” ve “4” isimli JAVA projeleri bulunmaktadır. Her bir projede farklı birer kütüphane eklenmiştir. Sırasıyla bu projelerde 2. bölümde verilen aynı görevleri gerçekleyiniz. Her proje için süre yarım saattir. Tüm gerçekleştirme, her proje içindeki src/userTest dizinindeki APIUsabilityTest.java sınıfı içinde yapılacaktır. Bu sınıf, size verilen haliyle boş bir frame oluşturarak içine çizilecek şekilleri size bırakmaktadır. Çizilecek olan tüm şekiller, eşit ölçüdeki dikdörtgen ile çevrelenebilmektedir. Bu nedenle her şekil için sabit alana ihtiyaç vardır.

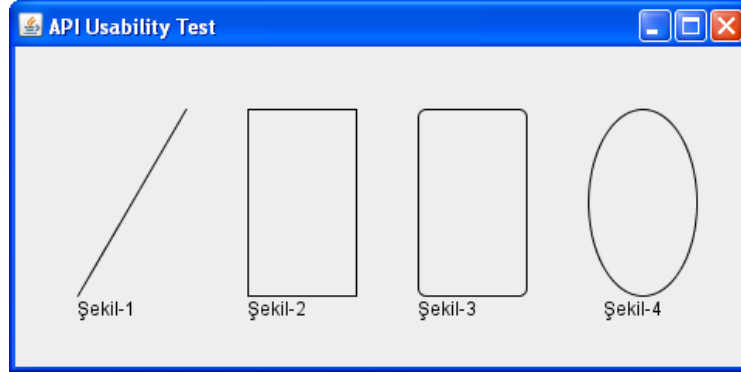
Gerçekleme sırasında ihtiyaç duyabileceğiniz dokümantasyon (Javadoc) masaüstündeki *doc* klasöründe ye almaktadır.

Gözetmen tarafından aksi belirtilmedikçe java.awt paketindeki sınıflar kullanılmamalıdır.

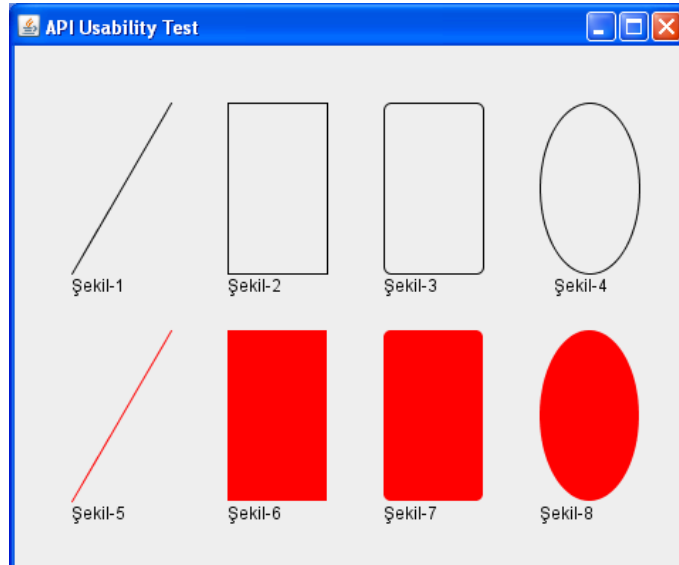
3. Görevler

Lütfen her görev tamamlandığında gözetmeni uyarınız.

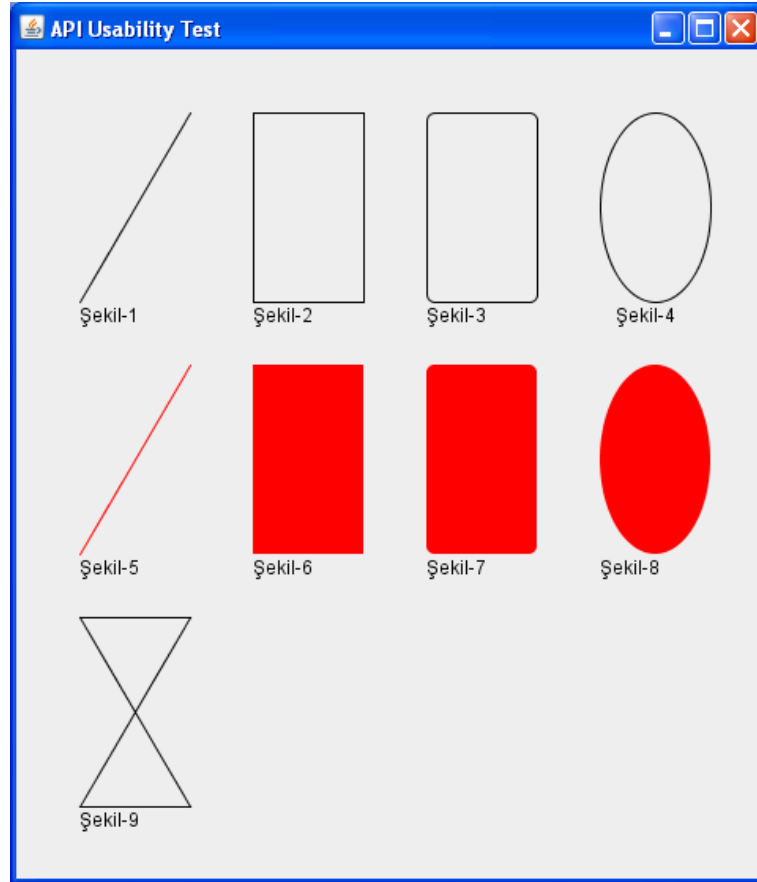
1. Gözetmen tarafından gösterilen API'yi kullanarak aşağıdaki şekildeki gibi bir çizgi, bir dikdörtgen, bir yuvarlatılmış dikdörtgen ve bir elips çiziniz.
Çizdiğiniz şekillerin altına ne olduklarını yazınız.



2. Verilen renk ile şekillerin içini doldurunuz. 1. görevde çizilmiş olan şekiller boyanabileceği gibi, verilen renk ile doldurulmuş yeni şekiller de çizilebilir.



3. 3. satırda gösterilen şekli çiziniz. Eski şekillerinizi silebilir ya da şekillerin sonuna ekleyebilirsiniz.



10.2. Ek-2: Anket

Tarih:

(Gözetmen tarafından doldurulacaktır.) UPA:

Uygulama Programlama Arayüzlerinin Kullanılabilirlik Deneyleri

Anket

1. UPA üzerinde bazı sınıfları veya metotları bulmakta zorlandınız mı?

Çok zor ☐ Zor ☐ Orta ☐ Kolay ☐ Çok kolay ☐

Bulmakta en çok zorlandığınız kısımları hatırlıyorsanız yazınız:

.....

2. Yazdığınız kodu değiştirmekte zorlandınız mı?

Çok zor ☐ Zor ☐ Orta ☐ Kolay ☐ Çok kolay ☐

Değişiklik yaparken özellikle zorlandığınız kısımlar nelerdir?

.....

3. Kullandığınız UPA ile üzerinde uğraştığınız görevler makul uzunlukta kod ile gerçekleştirilebiliyor muydu? Yoksa ufak işler için fazla satır sayısı mı gerekti?

Çok uzun ☐ Uzun ☐ Orta ☐ Kısa ☐ Çok kısa ☐

.....

4. Kullandığınız UPA üzerinde en çok zihinsel uğraş harcamanız gerekti mi?

Çok fazla ☐ Fazla ☐ Orta ☐ Az ☐ Hiç ☐

Zihinsel uğraş harcamanıza neden olan kısımlar nelerdir?

.....

5. Kullandığınız UPA'nın bazı hatalara sık düşülecek şekilde tasarlandığını düşündüğünüz oldu mu?

Çok fazla ☐ Fazla ☐ Orta ☐ Az ☐ Hiç ☐

Sıklıkla ya da kolaylıkla düşülebilecek hatalara örnek verebilir misiniz?

.....

6. Bazı bölümlerin özellikle garip çalıştığını ya da garip tarif edildiğini düşündüğünüz oldu mu?

Çok fazla ☐ Fazla ☐ Orta ☐ Az ☐ Hiç ☐

Size tuhaf gelen bölümler nelerdir?

.....

7. Sizce yazdığınız koddaki kısımların her birinin hangi işi yerine getirdiği açık mı?

Hiç anlaşılıyor ☐ Anlaşılıyor ☐ Orta ☐ Anlaşıyor ☐ Çok Açık ☐

Bir başka geliştiricinin anlamakta zorlanacağı kısımlar neler olabilir?

.....

8. Bir görevi yerine getirirken geliştirmeyi bırakarak o ana kadar gelinen aşamayı kontrol ettiğiniz oldu mu? İsteddiğiniz zaman geldiğiniz aşamayı değerlendirmek ne kadar zordu?

Çok zor ☐ Zor ☐ Orta ☐ Kolay ☐ Çok kolay ☐

9. Herhangi bir görevi yerine getirirken, çağrılarınız üzerinde istediğiniz sırayı izleyebilirsiniz mi? Yoksa UPA çağrılarının sırası konusunda ne kadar katı?

Çok katı ☐ Katı ☐ Orta ☐ Esnek ☐ Çok esnek ☐

10. UPA üzerinde farklı oldukları halde benzer gibi ifade edilen kısımlarla karşılaştınız mı? Ya da benzer oldukları halde farklı gibi ifade edilen kısımlarla karşılaştınız mı?

Çok fazla ☐ Fazla ☐ Orta ☐ Az ☐ Hiç ☐

Bunlar nelerdir?

.....

11. UPA ile ilgili diğer görüşleriniz:

.....

.....

.....

.....

.....

12. (NOT: Tüm UPA'lar kullanıldıktan sonra yanıtlanacaktır):

Kullandığınız tüm UPA'ları kullanım kolaylığı açısından sıraladığınızda bu UPA kaçınıcı sırada yer almaktadır? (1: en kolay – 4: en zor)

1 ☐ 2 ☐ 3 ☐ 4 ☐

Teşekkürler...