Görme Engelli Yazılım Geliştiricilerin Yazılım Geliştirme Araçları Kullanırken Karşılaştıkları Problemler

Çağdaş Evren Gerede*, Görkem Mülayim

TOBB Ekonomi ve Teknoloji Üniversitesi, Ankara, Türkiye
Web sayfası: http://cegerede.etu.edu.tr

Özet. Yazılım geliştirme süreçleri sırasında kullanılan yazılım geliştirme araçlarının birçok özelliği maalesef görme engelli yazılım geliştiriciler tarafından kullanılamaz. Yazılım geliştirme araçlarının yeterli oranda erişilebilir olmaması görme engellilerin profesyonel olarak yazılımcı olmasının önünde önemli bir engel teşkil etmektedir. Diğer taraftan bu engeli aşabilen geliştirici sayısı az olduğu için de bu geliştiricilerin ihtiyaçlarının karşılanması önceliklendirilmemektedir. Erişilebilirlik sorunlarının neler olduğu anlaşılmadan bu engelleri kaldıracak yeni destek teknolojilerinin geliştirilmesi mümkün olamaz. Bu çalışmayla amacımız görme engelli geliştiricilerin yazılım geliştirme araçları kullanırken karşılaştıkları sorunların neler olduğunu incelemektir.

Anahtar Kelimeler: Yazılım geliştirme araçları, Erişilebilirlik, Görme engelli yazılım geliştiriciler

* Başlıca yazar

Issues Blind Software Developers Experience When Using Software Development Tools

Çağdaş Evren Gerede*, Görkem Mülayim

TOBB University of Economics and Technology, Ankara, Turkey
Web page: http://cegerede.etu.edu.tr

Abstract. Unfortunately many features of software development tools are not accessible by blind users. This forms a major obstacle for them to become professional software developers. On the other hand, many users cannot surpass this obstacle which prevents enough demand to form for tool developers to improve the current situation. Without understanding what accessibility problems there are for blind users, it would not be possible to develop appropriate assistive technologies. With this exploratory study, we investigate the issues blind software developers experience when using software development tools.

Keywords: Software development tools, Accessibility, Blind software developers

1 Giriş

Yazılım geliştirme süreçleri sırasında yazılımcılar tarafından sıkça kullanılan yazılım geliştirme araçlarına örnek olarak şunlar sayılabilir: kaynak kodların incelendiği ve yazıldığı Eclipse, NetBeans, IntelliJ benzeri entegre yazılım geliştirme ortamları (İng., integrated development environments, kısaca IDE); kaynak kod değişim önerilerinin denetlendiği (İng. code review) Gerrit[4], Crucible[1] benzeri araçlar; kaynak kod dosyalarının değişim tarihçesinin incelenip kodun evriminin incelenebildiği Git benzeri kod versiyon yönetim sistemleri. Bu tip araçların birçok özelliği maalesef görme engelli yazılım geliştiriciler tarafından kullanılamamaktadır. Görme engelli yazılım geliştiricilerin yazılım geliştirme araçlarını kullanmada birçok erişilebilirlik (İng. accessibility) problemi ile karşılaştıkları hem akademik çalışmalar ile gösterilmiş[10, 11, 13, 23] hem de bu problemlerin görme engelli geliştiricilerin çalışma performansını oldukça negatif etkilediği görme engelli yazılım geliştiriciler tarafından rapor edilmiştir[5, 8, 9].

Örneğin, kaynak kod değişimi denetim araçlarında iki metin arasındaki değişimi tespit etmek için yaygın olarak düzeltme mesafesi (İng. edit distance) temelli metinsel "diff" algoritması [20] veya türevleri[16] kullanılmaktadır. Geleneksel olarak bu algoritmanın ürettiği çıktı, kaynak kodun kıyaslanan iki versiyonu üzerine yansıtılır ve aradaki değişimler renkler yardımıyla ifade edilir. Şekil 1'de Gerrit[4] aracından alınmış bu tür bir kullanımı gösteren bir ekran görüntüsü yer almaktadır.

* Primary author

Şekil 1. Gerrit[4] kod denetim aracı ekran görüntüsü - PatchSetInserter.java dosyasının iki versiyonu arasındaki değişim farklı renk tonları kullanılarak gösterilmektedir. Soldaki kırmızılar ve sağdaki açık yeşiller değişime uğrayan satırları; sağdaki koyu yeşiller ise eklenen yeni karakterleri göstermektedir. Sağ ekranda sarı arkafonlu kutuda iki yazılım geliştirici arasındaki kodun değişimi üzerinde yapılan tartışmaya dair mesaj dizisi de görülmektedir.

Burada solda "PatchSetInserter.java" adlı kaynak dosyasının eski hali, sağda ise yeni hali gösterilmektedir. Sol tarafta açık kırmızı ile sağ tarafta ise açık yeşil ile boyanan satırlar değişime uğramış satırları belirtmektedir. Sağ tarafta koyu yeşil ile boyalı karakterler ise yeni versiyonda eklenmiş karakterleri vurgulamaktadır. Bu tür bir gösterim büyük bir kaynak kod dosyasında meydana gelen kısıtlı değişikliklerin nerelerde gerçekleştiğinin hızlıca farkedilmesini ve değişimin etkilerinin anlamlandırılmasını hızlandırmaktadır. Fakat yazılım geliştirici görme engelliyse bu tür bir gösterim geliştiriciye beklenilen faydayı sağlayamaz.

Yazılım geliştirme araçlarının yeterli oranda erişilebilir olmaması görme engellilerin profesyonel olarak yazılımcı olmasının önünde önemli bir bariyer teşkil etmekte, diğer taraftan bu bariyeri aşabilen geliştirici sayısı az olduğu için de bu geliştiricilerin erişilebilirlik ihtiyaçlarının karşılanması endüstri tarafından önceliklendirilmemektedir[5, 8, 9].

Erişilebilirlik engellerinin neler olduğu anlaşılmadan bu engelleri ortadan kaldıracak yeni destek teknolojilerinin geliştirilmesi ve yazılım geliştirme araçlarına entegre edilmesi mümkün olamayacaktır. Bu hedefe yönelik olarak, bu bildiri-deki amacımız görme engelli geliştiricilerin yazılım geliştirme faaliyetlerini ve kullandıkları güncel yazılım geliştirme araçlarını inceleyerek, bu geliştiricilerin yazılım geliştirme faaliyetleri sırasında karşılaştıkları erişilebilirlik problemlerini rapor etmektir.

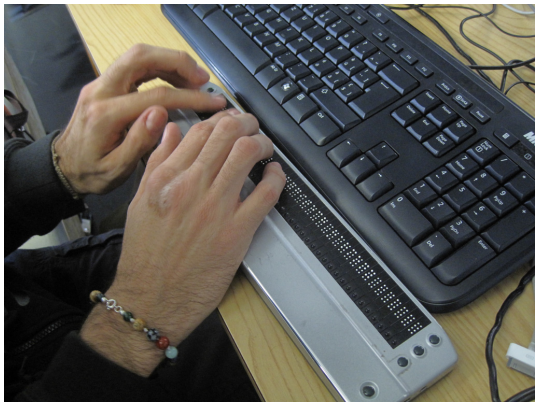
2 Görme Engelli Yazılımcıların Kod Geliştirme Süreci

2.1 Kaynak Kodun Okunması

Kod geliştiricilerin kod geliştirme süreçleri sırasında gerçekleştirdiği iki temel faaliyet bulunmaktadır. Birinci faaliyet bir kod bloğunun işlevini anlamaktır. Görme engeli olmayan geliştirici bu faaliyeti kod metnini programın kontrol akışı yönünde satır satır okuyarak yapar. Görme engelli bir geliştirici aynı adımı iki tür temel araç kullanarak yapabilir[5, 8, 9]. Birinci tür araç JAWS[6] veya NVDA[7] benzeri bir ekran okuyucu yazılımdır (İng. screen reader). Bu tür bir yazılım

ekrandaki metinleri konuşmaya çevirerek kullanıcıya seslendirir. Görme engelli yazılım geliştirici seslendirilen metni dinler ve kodun bir modelini zihninde oluşturmaya çalışır. Ekran okuyucular metinleri değişik hızlarda seslendirebilmektedir. Ekrandaki metne karşılık gelen sentetik konuşmanın üretilmesi sırasında kullanılacak birim eleman olarak harfler, heceler veya kelimeler seçilebilmektedir

Kaynak kod metnine ulaşmakta kullanılabilecek ikinci tür araç ise kabartmalı ekran cihazıdır (İng. braille display) (Bakınız Şekil 2). Bilgisayara bağlanan bu mekanik araç vasıtasıyla ekrandaki metinler bir levhaya kabartmalı fiziksel şekiller olarak yansıtılır. Böylece geliştirici kabartmalara dokunarak kod metnine ulaşmış olur.



Şekil 2. Klavyenin altında iğneli bir levha olarak kabartmalı ekran cihazi (İng. braille display) görülmektedir[2].

Kabartmalı ekranlar herhangi bir anda tipik olarak 40 ila 80 arası karakteri kabartabilmektedir. Kullanıcı ekranda imleci hareket ettirdikçe mekanik kabartmalar otomatik olarak yenilenir. Kabartmalı ekranlar, kabartmalı etiketli tuşlara sahip bir klavyenin yanında konumlandırılır ve eş zamanlı olarak kod yazma işlemi bu klavye vasıtasıyla yapılır. Kabartmalı ekran cihazlarının ekran okuyuculara göre farkı duyu hissi yerine dokunma hissini kullandırmasıdır.

2.2 Kaynak Kodun Yazılması

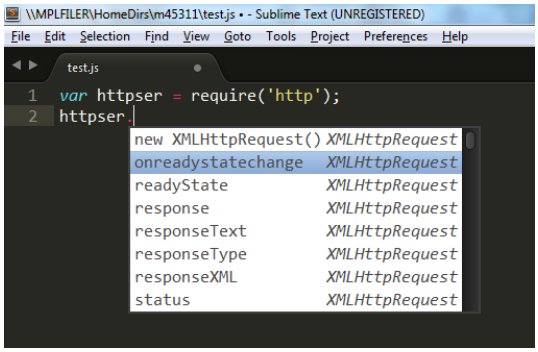
Kod yazma işlemi sırasında görme engelli geliştiriciler kabartmalı tuşlar içeren bir klavye kullanarak kod yazımı yapabilirler. Eş zamanlı olarak yazdıkları metnin doğruluğunu ekran okuyucu veya kabartmalı ekran cihazı kullanarak teyit edebilir.

[2] Kaynak: `https://goo.gl/kMV4z8`

3 Kaynak Kod Geliştirme Araçlarında Görme Engelli Kullanıcıların Karşılaştıkları Problemler

3.1 Görsel Etkileşim Tekniklerinin Kullanımının Yol Açtığı Sorunlar

Kod geliştirme araçlarında bazı fonksiyonlar yalnızca "pencereler, simgeler, menüler, işaretçiler", kısaca PSMİ (İng. "windows, icons, menus, pointer", kısaca WIMP[17]) etkileşimi düşünülerek tasarlanmıştır [12, 24, 25]. Bu da kullanıcının imleci ekrandaki görüntünün belirli bir koordinatına götürmesini gerektirir. Görme engelli kullanıcının bu işlemi yapması oldukça zordur. Genelde fare kullanımına alternatif olarak erişilebilirliğin arttırılması amacıyla aynı fonksiyonlar için klayve kısayolları da sağlanır. Fakat zaman zaman bu tür kısayolların eklenilmesi unutulmaktadır[2]. Bazen de kullanıcılar görsel diyaloglar üzerinden daha karmaşık bir etkileşim yürütmek durumunda kalırlar. Bu durumlar için tek bir klavye kısayolu tanımlamak imkansız hale gelmektedir. Örneğin, Şekil 3'de Sublime Text adlı editörün sağladığı otomatik tamamlama (İng. auto-completion) ekranı görülmektedir.



Şekil 3. Sublime Text Auto-completion Ekranı[3].

Bu ekranın başlatılması bir klavye kısa yolu arkasına konulabilir. Fakat görme engelli geliştiricinin kullandığı ekran okuyucu yazılım veya kabartmalı ekran cihazı vasıtasıyla diyalogta gösterilen tamamlama önerilerinin neler olduğunu anlaması ve bunlardan uygun olanı seçmesi oldukça yorucu hale gelmektedir. Her liste elemanı birden fazla metin gurubundan oluşur. Bu örnekte sağdaki metin gurubu "httpser" adlı değişkenin veri tipini, soldaki metin gurubu ise önerilen metni göstermektedir. Dolayısıyla diyalog oldukça fazla sayıda metin içermektedir. Editörde her yeni karakter yazımı ile beraber öneri listesinin otomatik olarak güncelleniyor olması bu etkileşimi daha da yorucu hale getirir. Sonuç olarak bir fonksiyon bu şekilde bir diyalog ile sunulduğunda, her ara etkileşim erişilebilir

[3] Kaynak: `https://goo.gl/RHBXQo`

olsa dahi, tüm etkileşim görme engelliler tarafından etkin olarak gerçekleştirilemediğinden, bu fonksiyon erişilemez hale gelmektedir.

3.2 Kod Parçalarının Görsel Konumlandırılmasından Faydalanıldığında Ortaya Çıkan Problemler

Geleneksel olarak kaynak kodlar yazılırken geliştiriciler tarafından girintiler (İng. indentation) kodun hiyerarşik yapısının daha rahat anlaşılması amacıyla sıkça kullanılmaktadır. Örnek Kod 2.1 bu şekilde bir kullanımı göstermektedir.

```
1  public class BinarySearch {
2      public int find (final int[] data, final int key) {
3          int low = 0, high = data.length - 1;
4
5          while (low <= high) {
6              final int i = (low + high) >> 1;
7              final int v = data[i];
8
9              if (v == key) return i;
10             else if (v < key) low = i + 1;
11             else // v > key
12               high = i - 1;
13         }
14
15         return -1;
16     }
17 } // end of class
```

Örnek Kod 2.1. Girintilerin ve boşlukların kodun anlaşılırlığını arttırmak için kullanımı[5]

Örneğin altıncı ile on ikinci satırlar arasındaki kod bloğunun beşinci satırdaki "while" ifadesi tarafından kapsandığı kolaylıkla görülebilmektedir; çünkü altıncı ve on ikinci satırlar beşinci satıra göre daha sağda konumlandırılmıştır. Bu tür bir görsel gösterim sayesinde, örneğin, şu kanıya çok hızlı bir şekilde varılabilir: Program, beşinci satırdaki şart sağlanmadığında dokuzuncu satıra uğrayamaz. Maalasef bu yöntem görme engelli geliştiriciler için aynı faydalı etkiyi yaratamaz çünkü bu geliştiricilerin kullanabildikleri metne ulaşma mekanizmaları (ekran okuyucu ve kabartmalı ekran cihazı) metnin görsel konumlandırmasını geliştiriciye aynı biçimde aktaramaz (Bakınız Tablo 1).

Girinti kullanımına ek olarak programda kullanılan andaçlar (İng. token) arasında anlaşılırlığı arttırmaya yönelik boşluk kullanımı da oldukça yaygındır. Örneğin, yedinci satırda "v = data[i]" ifadesinde "v", "=" ve "data[i]" arasında birer boşluk bırakılmıştır. Fakat görme engelli kullanıcı bu metne ekran okuyucu ile ulaştığında boşluk karakteri metni anlamada kullanıcıya ek kognitif yük getirmektedir (Şöyle bir seslendirme yapılır: "ve boşluk eşittir boşluk data köşeli parentez aç i köşeli parentez kapa"). Bunu önlemek için boşluk karakterlerinin

[5] Kaynak kod Emma[3] adlı projedeki bir kaynak kod dosyasından uyarlanmıştır

Tablo 1. Örnek Kod 2.1'deki metnin görme engelli geliştirici tarafından ulaşıldığındaki hali

public boşluk class boşluk binary search boşluk dalgalı parentez aç yeni satır boşluk boşluk boşluk boşluk public boşluk int boşluk find boşluk aç parentez final boşluk int köşeli parentez aç köşeli parentez kapa boşluk data, boşluk final boşluk int boşluk keykapa parentez boşluk dalgalı parentez aç yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk int boşluk low boşluk eşittir boşluk sıfır virgül boşluk high boşluk eşittir boşluk data nokta length boşluk eksi boşluk bir noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk while boşluk aç parentez low boşluk küçüktür eşittir boşluk highkapa parentez boşluk dalgalı parentez aç yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk final boşluk int boşluk i boşluk eşittir boşluk aç parentez low boşluk artı boşluk high kapa parentez boşluk büyüktür büyüktür boşluk bir noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk final boşluk int boşluk ve boşluk eşittir boşluk data köşeli parentez aç i köşeli parentez kapa noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk if boşluk aç parentez ve boşluk eşittir eşittir boşluk key boşluk return boşluk i noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk else boşluk if boşluk aç parentez ve boşluk küçüktür boşluk keykapa parentez boşluk low boşluk eşittir boşluk i boşluk artı boşluk bir noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk else boşluk slaş slaş boşluk ve boşluk büyüktür boşluk key yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk high boşluk eşittir boşluk i boşluk eksi boşluk bir noktalı virgül yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk dalgalı parentez kapa yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk yeni satır boşluk boşluk boşluk boşluk boşluk boşluk boşluk boşluk return boşluk eksi bir noktalı virgül yeni satır boşluk boşluk boşluk boşluk dalgalı parentez kapa yeni satır dalgalı parentez kapa boşluk slaş slaş boşluk end boşluk of boşluk class

ekran okuyucu tarafından görmezden gelinmesi istenebilir (Bu defa şöyle bir seslendirme yapılır: "ve eşittir data köşeli parentez aç i köşeli parentez kapa"). Fakat bu durumda da boşluk kullanımı konusunda takım geleneğine aykırı yapılan hatalar geliştirici tarafından tespit edilemez. Örneğin, kodda "v" ile "=" arasında yanlışlıkla boşluk bırakılmadıysa veya iki boşluk karakteri bırakıldıysa, bu durum görme engelli geliştirici tarafından tespit edilemez.

Bir başka ilginç durum ise on birinci satırda gözlemlenmektedir. Bu satırın sonunda "else" ifadesine hangi durumda ulaşılacağını açıklayan bir yorum ifadesi bulunmaktadır. Görsel olarak bu tür bir yorum ifadesini satırın geri kalanından ayırmak çok fazla efor gerektirmez çünkü konum olarak yorum ifadesi satırın en sağında bulunmaktadır. Fakat aynı durum görme engelli bir kullanıcı için geçerli değildir çünkü bu kullanıcıya ulaşan metin görsel konum özelliklerini kaybeder (Bakınız Tablo 1).

Sonuç olarak sıradan bir ekran okuyucu veya kabartmalı ekran cihazı kullanıldığında kod metninin hiyerarşik yapısının, kod parçaları arasındaki ilişkilerin ve program akışının anlaşılması, engelsiz bir geliştiriciye kıyasla, daha fazla zaman ve daha yoğun bir konsantrasyon gerektirir.

3.3 Kısıtlı Bağlam

Kaynak kodlar üzerinde çalışan görme engelli bir geliştiricinin karşılaştığı bir diğer zorluk ise kullandıkları metne ulaşma araçlarının birim zamanda çok kısıtlı bir bağlam bilgisini sağlıyor olmasıdır. Örnek Kod 2.2'de görüldüğü üzere dokuzuncu satırı inceleyen geliştirici bu satırın yazılıma olan etkisini anlamak için "v", "k",

ve “i” değişkenlerinin nerede tanımlandıklarını, tanımlandıkları yerde olan ilgili yorumları, değişkenlerin veri tiplerinin ne olduklarını, dönülen “i” değişkeninin daha önce nerede hesaplandığını incelemeye ihtiyaç duyabilir. Dolayısıyla yalnızca bu satırın metin olarak ne içerdiğini bilmek satırın program üzerindeki etkilerini anlamaya yetmemektedir.

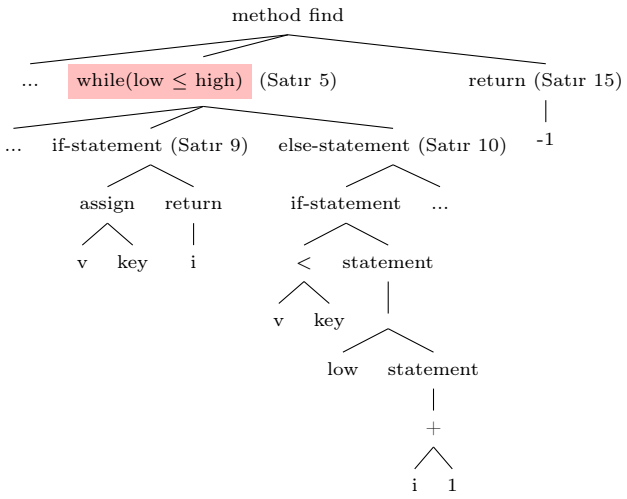
```
1
2
3
4
5
6
7
8
9               if (v == key) return i;
10
11
12
13
14
15
16
17  .
```

Örnek Kod 2.2. Ekran okuyucu ve kabartmalı ekran cihazı ile görme engelli bir geliştirici herhangi bir anda kod hakkında çok daha kısıtlı bir bağlam bilgisine ulaşabilmektedir. Bu deneyimi görsellemek için Örnek Kod 2.1’deki dokuzuncu satır hariç diğer satırlar gizlenmiştir.

3.4 Kodun Hiyerarşik Yapısı Üzerinde Gezinmek

Bir kaynak kod dosyasının içeriği düz bir metin olarak görülebilir. Fakat aslında dilin dil bilgisi kuralları (İng. grammar) düşünüldüğünde, aslında kod metni hiyerarşik bir yapıyı tanımlar. Şekil 4’de bu hiyerarşik yapı gösterilmiştir.

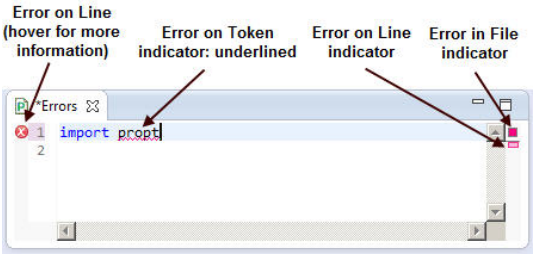
Geliştirici program kodunu anlamak için koddaki satırlar arasında ardışık olarak gezinmek yerine kodun hiyerarşik yapısı üzerinde gezinmek isteyebilir. Örneğin, Örnek Kod 2.1’de beşinci satırdaki döngü başlangıcını inceleyen geliştirici döngü içerisindeki ifadeleri incelemeye devam edebilir veya döngü detayları yerine bir sonraki kod bloğunun yer aldığı on beşinci satıra göz atmak isteyebilir. Fakat sıradan ekran okuyucular ve kabartmalı ekran cihazları bu tür bir rastgele erişime (İng. random access) olanak sağlayamazlar.



Şekil 4. Örnek Kod 2.1'de yer alan kodun hiyerarşik yapısı (Burada derleyicinin inşa edeceği soyut sözdizimi ağacının basitleştirilmiş hali resmedilmiştir).

3.5 Görsel Geri Bildirimler

Kod yazım araçları kod yazımı sırasında geliştiriciye sık sık görsel geri bildirimde bulunur. Örneğin Şekil 5'de Eclipse kod geliştirme aracı editöründe kodda yapılan bir hata sonucu ortaya çıkan görsel geri bildirimler görülmektedir.



Şekil 5. Eclipse kod geliştirme aracı editöründe kodda yapılan bir hata sonucu ortaya çıkan geri bildirimler görülmektedir[7].

Ekranın sol ve sağ bölümünde hatalı satır işaretlenmiş, hatayı içeren andaç ise altı kırmızı bir çizgiyle çizilerek gösterilmiştir. Buradaki görseller kullanıcının talebi üzerine değil, kullanıcının metni değiştirmesine tepki olarak olarak ortaya çıkarlar. Ayrıca ekranda kod metni ve bu metin ile ilgili geri bildirimler iki farklı görsel kanal aracılığıyla geliştiriciye sunulmaktadır. Görme engelli

[7] Kaynak: `https://goo.gl/dq9spp`

geliştirici ekran okuyucu yazılımlar ve kabartmalı ekran cihazları vasıtasıyla kod metnine ulaşırken tek bir kanala sahiptir. Dolayısıyla metin ve metne ait geri bildirimler aynı kanal üzerinden sunulmak durumundadır. Bu harmanlamayı geliştirme aracı yapar. Bu harmanlama doğru biçimde yapılmadığında görme engelli geliştiricinin kafasını karıştıran bir deneyim ortaya çıkmaktadır.

4 İlgili Çalışmalar

Görme engelli yazılım geliştiricilerin yazılım geliştirme sırasında karşılaştıkları engellerin neler olduğu ile ilgili güncel bazı çalışmalar mevcuttur[10, 11, 19, 23]. Mealin vd.[19] sekiz görme engelli geliştirici ile mülakatlar yapıp kod geliştirme süreçlerinde yaşadıkları problemleri anlamaya çalışmış, sonuç olarak geliştiricilerin var olan IDE'leri kullanmadıklarını rapor etmiş, fakat bunun nedenlerini detaylı olarak incelememiştir. Albusays vd. [10] 69 görme engelli geliştirici ile bir anket çalışması yapmıştır. Anket sonucunda geliştiricilerin IDE'lerin yerine metin editörleri kullandıkları, büyük kodlar üzerinden gezinme konusunda problem yaşadıkları, UML gibi kod yapısını anlatan görsel diyagramları çizmekte veya anlamakta zorluk çektikleri, hata ayıklama (İng. debugging) araçlarını kullanamadıkları rapor edilmiştir. Armaly vd. [11] görme engelli geliştiricilerin bir programın ne yaptığını anlamaya çalışırken (İng. program comprehension) engelsiz geliştiricilere göre farklı yöntemler izleyip izlemediğini araştırmıştır. Bunun için dört kör geliştiriciye programlama görevleri verilmiş ve geliştiriciler bu görevleri gerçekleştirirlerken ekran imleci (İng. cursor) ile kod üzerinde daha çok nereleri gezdikleri incelenmiştir. Sonuç olarak geliştiricilerin diğer kod bölgelerine kıyasla en çok metod imzalarını (İng. method signatures) ziyaret ettikleri ve bu ziyaret zamanının tüm görev zamanının %25'ini oluşturduğu gözlemlenmiştir. Stefik vd. [23] görme engelli orta okul ve lise öğrencilerine programlama öğretmek için gerekli teknolojileri üretmek için üç yıllık bir araştırma projesi yürütmüşler. Proje sonunda Hop adında bir programlama dili ile Sodbeans adında işitsel bir programlama ortamı üretmişlerdir.

Görme engelli geliştiricinin, kaynak kodların hiyerarşik yapısı üzerinde gezinmesini sağlayacak bazı teknikler Smith vd. [21] ile Baker vd. [14] tarafından önerilmiştir. İlk çalışmada görsel olmayan bir editör geliştirilmiştir. Burada kaynak kod bir ağaç şeklinde geliştiriciye sunulur ve kullanıcı ağaç üzerinde çeşitli klavye komutları ile gezip kod parçalarını sorgulayabilir. İkinci çalışmada Eclipse aracı içerisine kod metni ile eş zamanlı olarak görsellenen ve koda karşılık gelen bir soyut sözdizimi ağacının yeraldığı bir pencere yerleştirilmiştir. Bu ek pencere ile kullanıcı ağaç üzerinde gezebilir ve bu sırada otomatik olarak kaynak kod metni ağaçta ziyaret edilen düğüme göre güncellenir.

Görme engelli yazılım geliştiricilerin kaynak kodların yapısını daha kolay anlamalarına yönelik olarak, kodun melodilerle ve seslerle nasıl temsil edilebileceği üzerine bazı ön çalışmalar yapılmış ve umut verici sonuçlar elde edilmiştir [15, 18, 22]. Fakat henüz bu çalışmalar pratik araçlara dönüşememiştir.

5 Sonuçlar ve Gelecek Çalışmalar

Bu çalışmada görme engelli yazılım geliştiricilerinin günümüz yazılım geliştirme araçlarını kullanırken karşılaştıkları fakat literatürde detaylı incelenmemiş engellerin neler olduğunu inceledik. Bu engellerin neler olduğunun anlaşılmasının bu engelleri ortadan kaldıracak yeni araştırmalara faydalı bir girdi olacağını umuyoruz. Yazılım geliştirme araçlarındaki erişilebilirlik problemleri giderildiğinde yazılım geliştirmeye meraklı görme engelli bireylerin geliştirme faaliyetlerini etkili yapabilmelerinin önündeki önemli bir bariyer ortadan kalkacaktır. Gelecekte çalışmalarımızda bildirdiğimiz erişim problemlerinden bir bölümüne çözüm üretmeyi hedeflemekteyiz. Üretilen çözümlerin hem görme engelli geliştiricilerin hem de engelsiz geliştiricilerin ortaklaşa çalıştıkları araçlar içerisine entegre olacak şekilde tasarlanması gerektiğine inanıyoruz. Bu yaklaşım ile görme engelli kullanıcıların deneyimlerini iyileştirme amacıyla üretilecek kod geliştirme teknolojilerinin engelsiz geliştiricilerin de işine yarayabileceğini ve bu alanda yapılacak ilerlemelerin aslında tüm geliştiricilerin yazılım geliştirme süreçlerini iyileştirebileceğini düşünüyoruz.

Kaynakça

1. Crucible (jun 2017), https://www.atlassian.com/software/crucible
2. Eclipse bug - forgotten keyboard shortcut for switching between pages (jun 2017), https://bugs.eclipse.org/bugs/show_bug.cgi?id=22004
3. Emma (jun 2017), http://emma.sourceforge.net
4. Gerrit (jun 2017), https://www.gerritcodereview.com/
5. How can you program if you are blind? (jun 2017), https://stackoverflow.com/questions/118984/how-can-you-program-if-youre-blind
6. Jaws (jun 2017), http://www.freedomscientific.com/Products/Blindness/JAWS
7. Nvda (jun 2017), https://www.nvaccess.org/
8. Tools of blind programmer (jun 2017), https://www.parhamdoustdar.com/2016/04/03/tools-of-blind-programmer
9. A vision of coding, without opening your eyes (jun 2017), https://medium.freecodecamp.com/looking-back-to-what-started-it-all-731ef5424aec
10. Albusays, K., Ludi, S.: Eliciting programming challenges faced by developers with visual impairments: Exploratory study. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 82–85. CHASE '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2897586.2897616
11. Armaly, A., McMillan, C.: An empirical study of blindness and program comprehension. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 683–685. ICSE '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2889160.2891041
12. Arnold, S.C., Mark, L., Goldthwaite, J.: Programming by voice, vocalprogramming. In: Proceedings of the Fourth International ACM Conference on Assistive Technologies. pp. 149–155. Assets '00, ACM, New York, NY, USA (2000), http://doi.acm.org/10.1145/354324.354362

13. Baker, C.M.: Increasing access to computer science for blind students. SIGACCESS Access. Comput. (117), 19–22 (Feb 2017), http://doi.acm.org/10.1145/3051519.3051523
14. Baker, C.M., Milne, L.R., Ladner, R.E.: Structjumper: A tool to help blind programmers navigate and understand the structure of code. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. pp. 3043–3052. CHI '15, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2702123.2702589
15. Berman, L.I.: Program comprehension through sonification. Ph.D. thesis, Durham University, England, UK (2011), http://etheses.dur.ac.uk/1396/
16. Canfora, G., Cerulo, L., Penta, M.D.: Ldiff: An enhanced line differencing tool. In: 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings. pp. 595–598 (2009), https://doi.org/10.1109/ICSE.2009.5070564
17. Dix, A., Finlay, J.E., Abowd, G.D., Beale, R.: Human-Computer Interaction (3rd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2003)
18. Finlayson, J.L., Mellish, C., Masthoff, J.: Fisheye views of java source code: An updated LOD algorithm. In: Universal Access in Human-Computer Interaction. Applications and Services, 4th International Conference on Universal Access in Human-Computer Interaction, UAHCI 2007 Held as Part of HCI International 2007 Beijing, China, July 22-27, 2007 Proceedings, Part III. pp. 289–298 (2007), https://doi.org/10.1007/978-3-540-73283-9_33
19. Mealin, S., Murphy-Hill, E.: An Exploratory Study of Blind Software Developers. In: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 71–74. IEEE (2012), http://people.engr.ncsu.edu/ermurph3/papers/vlhcc12.pdf
20. Miller, W., Eugene, Myers, W.: A file comparison program. Software: Practice and Experience p. 1040 (1985)
21. Smith, A.C., Cook, J.S., Francioni, J.M., Hossain, A., Anwar, M., Rahman, M.F.: Nonvisual tool for navigating hierarchical structures. In: Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility. pp. 133–139. Assets '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1028630.1028654
22. Stefik, A., Hundhausen, C., Patterson, R.: An empirical investigation into the design of auditory cues to enhance computer program comprehension. Int. J. Hum.-Comput. Stud. 69(12), 820–838 (Dec 2011), http://dx.doi.org/10.1016/j.ijhcs.2011.07.002
23. Stefik, A.M., Hundhausen, C., Smith, D.: On the design of an educational infrastructure for the blind and visually impaired in computer science. In: Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education. pp. 571–576. SIGCSE '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1953163.1953323
24. Wagner, A.: Automation of vui to gui mapping. In: CHI '13 Extended Abstracts on Human Factors in Computing Systems. pp. 1941–1944. CHI EA '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2468356.2468706
25. Wagner, A., Gray, J.: An empirical evaluation of a vocal user interface for programming by voice. Int. J. Inf. Technol. Syst. Appoach 8(2), 47–63 (Jul 2015), http://dx.doi.org/10.4018/IJITSA.2015070104