ANDROİD UYGULAMALARI BELLEK HATALARI YAKALANMASI VE ETKİLERİ

İsmail Alper Sağlam[1], Aysu Betin Can[2]

[1]Bilişim Sistemleri, Enformatik Enstitüsü, ODTÜ, Ankara, Türkiye
alper.saglam@metu.edu.tr

[2]Bilişim Sistemleri, Enformatik Enstitüsü, ODTÜ, Ankara, Türkiye
betincan@metu.edu.tr

Öz: Günümüzde mobil uygulamalar oldukça yaygın kullanılmakta, birçok kurum servislerini mobil alanlara taşımaktadır. Bu uygulamalar için diğer yazılımlardan farklı gereksinimler bulunmaktadır. Bunların en başta gelenleri bellek kısıtı ve işlemci kullanımıdır. Bellek sızıntısı olan, hızlı yanıt veremeyen uygulamalar kullanıcı memnuniyetini düşürmektedir. Kullanıcıların mobil uygulamadan kolay vazgeçebilmeleri, bu çeşit yazılımlarda kullanıcı memnuniyetinin önemini artırmaktadır.

Bu çalışmada Android uygulamalarında bellek sızıntısına ve yetersiz bellek, uygulama yanıt vermiyor (ANR) mesajlarına yol açan sıkça yapılan yanlışları otomatik yakalayan bir araç geliştirilmiştir. Bu araç açık kaynaklı 100 Android uygulaması üzerinde çalıştırılmıştır. Bulunan yanlışlar ile uygulamaların kullanıcı puanlaması ve kullanımda olma sayıları ile karşılaştırılmıştır.

Bu çalışma ile geliştirilen araç sayesinde geliştiriciler kodlarındaki hataları daha kolay bulabilecek ve uygulamayı piyasaya sürdüklerinde bu tip sorunları en aza indirgemiş olacaklardır.

Anahtar Kelimeler: Android, bellek-sızıntısı, ANR, otomasyon

1 Giriş

Mobil uygulamalar, özelde Android uygulamaları, günümüzde sayıca oldukça artmış ve yaygınlaşmıştır. Bir Android uygulama marketi olan Google Play[1], üzerinde 2013 yılı haziran ayında yaklaşık 1 milyon adet uygulama bulunmaktadır[1]. Bu durum pek çok kurumun servislerini mobil uygulamalar aracılığı ile sunmaya başlamalarına neden olmaktadır; Turkcell Cüzdan, İşcep, Markafoni vb. kurumların yanı sıra geliştiriciler ticari anlamda mobil uygulamalar sunmaktadırlar.

Bir uygulamanın en çok satılanların arasında bulunması, ya da bir kurumun itibarını etkileyecek durumda olmaması için uygulamadan kaynaklanan sistem hata mesajlarının en az olması gerekmektedir. Mobil uygulamalarda kullanıcı memnuniyeti

[1] https://play.google.com

özellikle önemlidir, çünkü kullanıcı diğer yazılımlara kıyasla kolayca uygulamadan vazgeçebilmekte ve eleştirilerini ortak alanda duyurabilmektedir.

Mobil uygulamalardan beklenen özellikler şöyle sıralanabilir[5]: **Kararlı**: Uygulama göçmemeli, kapatılmaya zorlanmak zorunda kalmamalı, donmamalı ve hedeflenmiş cihazlardan hiç birinde anormal şekilde çalışmamalıdır. **Cevap verici**: Uygulama potansiyel olarak uzun sürebilecek işlemleri kullanıcı ara yüzü iş parçacığında yapmamalıdır ve bu sayede "ANR" (Uygulama Cevap Vermiyor) diyaloğu kullanıcıya gösterilmek zorunda kalmamalıdır. **Hızlı**: Uygulama hızlı bir şekilde yüklenmeli ve gereksiz işlemler ve özellikler barındırmamalıdır.

Bu çalışmada Android uygulamaları için yukarda sayılan kalite parametrelerini düşürecek genel hataları topladık. Bunun için Android geliştiricilerine sunulmuş çeşitli kitapları [1] [2], çevrimdışı [3] ve çevrimiçi [4] derslerin yanı sıra bazı blogları kullandık. Derlenen genel hataları kaynak kodunda tanınabilecek şekilde örüntülere dönüştürdük. Bu dönüşümün sonrasında örüntüleri kaynak kodunda otomatik olarak tanıyabilecek bir araç geliştirdik.

Araç geliştirmenin yanı sıra, açık kaynak kodlu uygulamalarının kaynak kodları ve Google Play istatistikleri toplandı. Burada hedeflenen soru şudur: "Bir uygulamanın kaynak kodunda belirlenen hata örüntüleri, o uygulamanın düşük puanlar almasına sebep olur mu?". Geliştirdiğimiz araç indirilen 100 Android uygulamasına uygulandı ve hata örüntüsü sayıları çıkartıldı. Google Play sayfasından ise kullanıcıların verdikleri puanlar ve indirme sayıları elde edildi.

Bildiri şu şekilde organize edilmiştir. Bölüm 2 Android hakkında ön bilgi ve literatür çalışmasını özetlemektedir. Bölüm 3 hata örüntülerini ve Bölüm 4 geliştirilen aracı sunmaktadır. Bölüm 5 denek programları ve sonuçları içermektedir. Son olarak Bölüm 6 bildiriyi sonuçlandırmaktadır.

1.1 Benzer Çalışmalar

Android uygulamalarının sınanması ve hata ayıklama için birkaç araç bulunmaktadır. En sık kullanılan araçlardan biri olan Android Lint, bir kaynak kod tarama aracıdır. ADT (Application Development Tools) ile birlikte gelen Android Lint'in kontrollerinin tam listesi resmi sitesinde [6] bulunmaktadır. Android Lint ile tespiti hedeflenen sorunlar şunlardır: performans problemleri, kullanılmayan kaynaklar, erişim problemleri, kullanılabilirlik problemleri, dilden dile dönüşümlerdeki eksiklikler ve internalizasyon problemleri.

UI/Application Exerciser Monkey rastgele tıklamalar, dokunuşlar, işaretler ya da sistem seviyesinde olaylar üretmek gibi kullanıcı olayları oluşturabilen bir araçtır[7]. Monkey, "Uygulamanın çökme ya da uygulamada beklenmeyen olağandışılıklar oluşması" ve "Uygulamanın ANR hatası vermesi" durumlarını denetlemektedir. Monkey verilen Android uygulamasında bir problemin sadece varlığını gösterir. Mesela, Monkey kullanarak Bellek Yetersizliği olağandışılığı sonucuna erişilebilir. Ancak, bu olağandışılığın Bölüm 3.1'de açıklanan ömrü dıştaki sınıftan daha uzun olan statik olmayan bir iç sınıf kullanmaktan kaynaklandığını bilinemez. Aksine, bizim aracımız uygulamadaki bellek yetersizliği olağandışılığı, ANR hatası veya yavaşlığın sebeplerini bulmayı hedeflemektedir.

Systrace bir Android uygulamasının performansını, uygulamanın işlemlerinin ve diğer Android sistem işlemlerinin çalışma zamanlarını yakalayarak analiz eder. Bu araç da Monkey gibi neyin CPU aşırı kullanımına neden olduğu hakkında bilgi vermez. Sonuç olarak, Systrace bir hatanın varlığını kanıtlamak ve kodda hatanın yerini bileşen bazında bulmak için kullanılır.

FindBugs [8] Java programları için bir hata örüntüsü bulucudur. Hovemeyer ve arkadaşları gerçek programlardan hem basit hem de karmaşık olan birçok hata deseni elde etmişlerdir. Google [9] ile FindBugs kullanılan bir deneyimde, Google'ın kodlarına uygulanan FindBugs sorunların önemsenecek çoğunluğunu çözmüş ve Java'da uzmanlaşmış geliştiricilerin bile bilgilerinde hataya sebep olan çok önemli bir açıklık olduğunu göstermişlerdir. Biz de aynı şekilde düşünüyoruz ve Android uygulamalarına özel benzer bir araç geliştirdik.

Hata örüntülerini bulacak araca bir diğer örnek olarak Lee ve arkadaşlarının [13] çalışması verilebilir. Bu çalışmanın hedef alanı taşıtlarda kullanılacak mobil uygulamalardır. Bu çalışma uygulama geliştiricilerinin uyabileceği ve yararlanabileceği iyi yöntemleri sağlayan çalışmalar yapılması gerekliliğinin kanıtıdır. Ayrıca, bu çalışma göstermiştir ki Android marketindeki hiçbir uygulama bu prensiplere tam anlamıyla uymamaktadır.

Liu ve arkadaşları da [18] çalışmalarında 70 adet performans hatasını toplamış ve bu hataları kategorilendirmiştir. Sonrasında hataların sebebi olabilecek iki örüntüyü ("Ana iş parçacığında çok uzun sürme ihtimali olan veri tabanı, ağ işlemlerinin yapılması", "Liste görünümünde görünümlerin yeniden kullanılmaması") kaynak kodunda arayan bir araç geliştirmiş ve bu aracı (PerfChecker) 29 adet popüler Android uygulamasında denemişlerdir. Sonuç olarak bu uygulamalar içinde 126 adet performans hatası bulmuş ve bunların 68 tanesi uygulamaların kendi geliştiricileri tarafından kabul görmüş ve bir sorun olarak gösterilmiş ve 20 tanesi programın verdiği önerilerle kısa sürede çözülmüştür. Liu ve arkadaşlarının bir diğer çalışmalarında VeriDroid isimli araç geliştirilmiş [17] ve bu araç boş işaretçi referansları ve veri tabanı işleyicisi gibi kaynakları sızdırılması hatalarını bulması beklenmiştir. VeriDroid 5 Android uygulamasında denenmiş ve 7 adet hata bulunmuştur. Liu ve arkadaşlarının bu çalışmaları bizim çalışmamızla hata örüntülerini yakalaması yönünden benzerlik gösterse de, bizim çalışmamızda PerfChecker'da bulunmayan hata örüntüleri ("Statik Olmayan İç Sınıf Kullanımı", "İş Parçacıklarında İptal Mekanizması Belirlememe" ve "İş Parçacığı Önceliklerinin Ayarlanmaması") konu edilmekte ve daha fazla uygulama üzerinde test yapılmaktadır.

Android uygulamalarındaki hataları bulan daha fazla örnek verilebilir. Mesela, [14] uygulamalar arasındaki haberleşme sırasında zayıflık oluşabilecek güvenlik ve gizlilik ihlallerini bulur. Bir diğer örnek ise, [15] cihazın bataryasının boşalmasına sebep olacak hataları ve sebeplerini bulur. Bir diğeri [16] ise manifest dosyasında bulunan gereksiz izinleri gösterir. Bu çalışmaların hepsi gerekli ve yararlıdır fakat hiçbiri hafıza sızıntıları ya da ANR hatasının sebebini bulmayı hedeflememektedir.

2 Android

Android açık kaynak kodlu, Linux çekirdeği üzerine inşa edilmiş bir mobil cihazlar için işletim sistemidir. Android, derlenmiş Java kodunu çalıştırmak için dinamik çevirmeli (JIT) Dalvik sanal makinasını kullanır. Geliştirilen Android uygulamaları 4 temel kısımdan oluşur:

1. Aktiviteler (Activities): Aktiviteler çalıştırılabilir kodun belirli kısımlarını oluşturan ve zamanın belirli bölgelerinde kullanıcıyla ve sistemle etkileşime geçerek gerekli veriyi sağlayan, sonunda da kullanılmadıkları zaman sistem tarafından sonlandırılan parçalardır.
2. Servisler (Services): Servisler arka planda çalışan ve uygulamanın bir parçası olan kısımlardır. Cihaz kapanana kadar arka planda hazır olarak çalışırlar ve verilerin ve hizmetlerin sağlanmasında kullanılırlar.
3. Yayın Alıcılar ve Olaylar (Broadcast Receivers and Intents): Yayın uygulamaları aygıtın temel mesajlarının (düşük pil uyarısı vb.) tüm sisteme gönderilmesidir. Olay (Intent) alıcıları ise belirli bir amaca göre bazı uygulamalardan ve servislerden bilgi toplanmasıdır.
4. İçerik Sağlayıcılar (Content Providers): İçerik sağlayıcılar uygulamaların dosya sisteminde ya da veri tabanı üzerindeki verilere erişimini sağlar.

Android uygulamalarında bunun dışında "kontekst" (context) kavramı bulunmaktadır. Kontekstler isminden de anlaşılacağı gibi uygulamanın ya da objenin güncel durumunu yansıtırlar. Yeni bir kullanıcı ara yüz birimi yaratılması, aktivite başlatılması, ortak kaynakların erişimi gibi durumlarda kontekst kullanılır. Her aktivite sınıfı aynı zamanda bir kontekstir ve bu "Aktivite Konteksti" (Context-activity) olarak adlandırılır. Uygulama içerisindeki bir diğer kontekst ise "Uygulama Konteksti"dir (Context-Application). "Uygulama Konteksti" bütün aktivitelerin yaşam döngülerinden bağımsız, uygulama çalıştığı sürece yaşayan bir kontekstir.

Bir Android uygulaması çalıştırıldığında, sistem ana iş parçacığı adı verilen bir iş parçacığı yaratır. Bu iş parçacığı olayları uygun kullanıcı ara yüzlerine dağıtmak ve çalıştırmak açısından çok önemlidir. Android'de de ana iş parçacığı kullanıcı ara yüzü iş parçacığıdır. Sistem uygulamanın her bir bileşeni için farklı bir iş parçacığı yaratmaz, bütün işlemler ana iş parçacığı üzerinden yürütülür.

3 Hata Örüntüleri

Android geliştirme dünyasındaki hata örüntülerinden bir liste oluşturabilmek için; dersleri, uzmanların bloglarını, StackOverflow[2], CodeRanch[3] ve Android Developers[4] gibi tartışma forumlarını inceledik. Bunun dışında "Android programming best practices", "Android programming bad practices" anahtar kelimelerini kullanarak

[2] http://stackoverflow.com/
[3] http://www.coderanch.com/forums
[4] http://developer.android.com/google/index.html

arama yaptık. Bu aramayı yapmak "kullanıcı ara yüzlerindeki iyi ve kötü yöntemler", "güvenlik için iyi ve kötü yöntemler", ve "performans için iyi ve kötü yöntemler" gibi çalışmaları gösterdi. Ek 1 içinde derlediğimiz hata örüntülerinin listesi bulunmaktadır.

Bu çalışmada, sözü geçen listeden otomasyona elverişli bir altküme seçildi. Bu altkümedeki hata örüntüleri ciddi problemler doğurabilmektedir ve hata örüntülerinin hâlihazırda çözümleri bulunmaktadır. Bahsedilen altkümedeki hata örüntüleri ve kategorileri aşağıdaki gibidir:

- Bellek kullanımını etkileyen hatalar
  - Statik olmayan iç sınıfların kullanımı
  - Listelerde görünümlerin tekrar kullanılmaması
- İşlemci kullanımı performansını etkileyen hatalar
  - İş parçacıkları için iptal mekanizması belirlenmemesi
  - İş parçacığı önceliklerinin ayarlanmaması

Bu listedeki öğeler hafıza sızıntılarına, iş parçacığı sızıntılarına (özel bir hafıza sızıntısı), ANR hatalarına ve uygulamanın yavaşlamasına sebep olur. Bu yöntemlerin iyi tanımlanmış olmaları ve kolayca saptanabilmeleri bu çalışma için çok önemlidir.

3.1 Bellek Kullanımını Etkileyen Hatalar

3.1.1. Statik Olmayan İç Sınıf Kullanımı

Java dilinde, iç sınıf bir başka sınıfın içinde tanımlanan sınıftır. Bir iç sınıf nesnesi yaratılması için önce dış sınıf nesnesi yaratılmalıdır. Dış sınıf nesnesinin iç sınıf nesnesine referansı olduğu gibi, iç sınıf nesnesinin de dış sınıf nesnesine referansı vardır. Bu durum iç sınıfın statik olmaması durumunda geçerlidir. Atık toplama (garbage collection) sırasında bu içten dışa referanslama sorun olabilmektedir [10].

```
public class ActivityWithTask extends Activity {
 private Button aButton; private View.OnClickListener anOnClickListener =
    new View.OnClickListener() {
     public void onClick(View view) {
        new ATaskLongRunning().execute();     } };
 private class ATaskLongRunning extends AsyncTask<Void, Void, Boolean> {
     protected Boolean doInBackground(Void... voids) {
        try {      // Just sleep.
           Thread.sleep(30000);
        } catch (InterruptedException e) {}
        // finish the work.
        return true;
     }
     protected void onPostExecute(Boolean aBoolean) {
        aButton.setText(aBoolean ? "Great" : "Not Good");     }
  }
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     aButton = (Button) findViewById(R.id.btnSave);
     aButton.setOnClickListener(anOnClickListener);   }
}
```

Şekil 1. AndroidWithTask.java sınıfı kaynak kodu

İç sınıf nesnesinin referanslandığı nesnelerin işgal ettikleri bellek, atık toplama sırasında gerekli olmasa bile toplanmaz. Bu durum bellek sızıntısına neden olur.

Android uygulamalarında bu durum özellikle aktivite kodlamalarında görülmektedir. "Runnable", "AsyncTask" ve "Handler" gibi ara yüzlerden türetilmiş anonim sınıfların aktivite kodlamalarında sıkça iç sınıf olarak kullanımı söz konusu olduğu için bu durum önemlidir. Şekil 1'de örnek bir aktivite kodlaması verilmiştir. "AndroidWithTask" sınıfının kodu incelendiğinde Android ile bağlantılı bir örnek görülecektir. Bu örnekte, aktivite nesnesi içerdeki sınıflar tarafından iki defa referanslanmıştır. Bunlardan biri "OnClickListener" ara yüzünü gerçekleştiren "anOnClickListener" sınıfının diğeri ise "AsyncTask" ara yüzünü gerçekleştiren "AtaskLongRunning" sınıfının nesneleridir. Bu durum oryantasyon değişikliklerinde "Activity" nesnelerinin bellek sızıntısına sebep olabileceği iki kritik bölüm olabileceğini göstermektedir.

Cihazın döndürülerek oryantasyonu değiştirildiğinde Android çatısı yeni bir aktivite nesnesi yaratacaktır ve önceki aktivite nesnesine gerek kalmayacaktır. Şekil 1 içindeki, "anOnClickListener" nesnesi uzun sürecek bir kod bulundurmadığı için aktivite nesnesi ile birlikte imha edilecektir. "ALongTaskRunning" nesnesi için ise durum daha farklıdır çünkü iş parçacığı çalışmaya devam ettiği sürece dış nesnesi olan eski aktivite nesnesine bir referans bulunacaktır ve işi bittiği halde atık toplama ile temizlenmeyecektir. Bu durum da aktivite nesneleri bellek sızıntısına sebep olmuş olur. Sonuç olarak, bu örüntüdeki kodları kullanmak "Yetersiz Bellek" (Out of Memory Exception)'in en önemli nedenlerinden biridir.

3.1.2. Liste Görünümünde Görüntüleri Yeniden Kullanmama

Listeleri göstermek, mobil cihazın kısıtlı kaynaklarının performanslı kullanılması gerektiği için önemli bir konudur. Bu problemin arkasındaki sebep listede birçok eleman olmasıdır. Ancak, bu durum liste üzerinde kaydırmanın pürüzsüz ve hızlı olmasını önlememelidir, bu işlem pili bitirmemelidir ve CPU veya RAM gibi kaynakları gereğinden fazla kullanmamalıdır [12].

```
public class WrongListAdapter extends BaseAdapter {
 private LayoutInflater mInflater;
 public int getCount(){ // get total number of items in this list}
 public Object getItem(int position){// get an item from a list}
 public long getItemId(int position) { // get id of an item}
 public View getView(int position, View convertView, ViewGroup parent) {
   View item = mInflater.inflate(R.layout.activity_main, null);
   ((TextView) item.findViewById(R.id.txtLastSavedRecord)).setText(atext);
   Bitmap mIcon1 = null;    Bitmap mIcon2 = null;
   ((ImageView) item.findViewById(R.id.action_settings))
          .setImageBitmap((position & 1) == 1 ? mIcon1 : mIcon2);
     return item;
 }}
```

Şekil 2. WrongListAdapter.java sınıfı (her seferinde yeni bir görüntü oluşturan liste adaptörü)

Android geliştirme uygulama programlama ara yüzünde (API), ölçeklenebilirlik ve performans için hazırlanmış "ListView" bileşeni bulunmaktadır. Bu bileşen görünen ya da görünecek olan ve liste elemanı olarak anılan çocuklarını ekrana çizer ve hazırlar. Programcılar "ListView"'in çocuklarını üretirken yeni görüntü üretimini tutabildikleri kadar minimumda tutmaya çalışmalıdırlar. "ListView"'in ekranda her yeni liste elemanını gösterme ihtiyacında, kayıtlı bağdaştırıcısındaki "getView" metodu çağırılır. Eğer bir geliştirici bu bağdaştırıcıyı Şekil 4 ile gösterilen "WrongListAdapter"'deki

gibi gerçekleştirirse, "getView" metodunun her çağırılmasında yeni bir görünüm üretilecektir. Bu mekanizma sistemin kaynaklarını sömüreceğinden başarısız olur.

Yeni bir görünüm üretmek yerine, programcı üretilmiş olan görünümleri yeniden kullanmalıdır. Eğer programcı "getView" metodunun ikinci argümanı olan "convertView" (görüntü) nesnesini kullanır ve yeni bir görünüm yaratmaktansa, bunu yeni gösterilecek bilgilerle güncellerse, "ListView"'in yüzlerce veya binlerce elemanı bile olsa, sadece ekranda görünen liste elemanları ve geri dönüştürülebilen diğer görünümler kadar bellek tutar.

3.2 İşlemci Kullanımı Performansını Etkileyen Hatalar

3.2.1. İş Parçacıklarında İptal Mekanizması Belirlememek

Java atık toplama mekanizması aktif iş parçacıklarının, kendilerine bir referans olmasa bile, sahip olduğu belleği serbest bırakmaz [11]. Eğer bir iş parçacığı "while(true)" veya benzeri bir mekanizma ile uzun süre çalışmaya bırakılırsa, bu iş parçacığı nesnesi atık toplama mekanizması tarafından toplanamaz. Örnek olarak, Şekil 2 ile sunulan "ThreadLeakedActivity" sınıfında uygulamayı veya aktiviteyi kapattığımızda, aktivite nesnesi ve bu aktivite ile çalışan iş parçacıkları sonlanıp atık toplama için elverişli hale gelir diye düşünülür. Aksine, Java iş parçacıkları bütün işlemleri bitene kadar yaşayan nesnelerdir. Bu durum iş parçacıkları harici olarak kapatılana veya tüm işlemler Android sistemi tarafından bitirilene kadar devam eder. Bu durumdan kurtulmanın yolu iş parçacıkları için kapatma ya da iptal mekanizması yerleştirerek aktivitenin döngüsü üzerinde kontrol kurmaktır. Örneğin Şekil 3 ile gösterilen "ThreadNotLeakedActivity" tanımında olduğu gibi "onDestroy" metodunda "close" metodunu çağırmak uygulamada kazara iş parçacığı sebebi ile bellek sızıntısı olmasını önler.

```
public class ThreadLeakedActivity extends Activity{
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      new AThread().start();
   }
   private static class AThread extends Thread {
      public void run() {
         while (true) {
            try { Thread.sleep(1000);
            } catch (InterruptedException e) {e.printStackTrace();}
         }
      }
   }
}
```

Şekil 3. ThreadLeakedActivity.java sınıfı (sızıntı yapan sınıf örneği)

```
public class ThreadNotLeakedActivity extends Activity {
   private AThread aThread;
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      aThread = new AThread().start();}
   private static class AThread extends Thread {
      private boolean running = true;
      public void run() {
         while (running) {
            try { Thread.sleep(1000);
            } catch (InterruptedException e) { e.printStackTrace(); }
      }}
      public void close() { running = false;}
   }  protected void onDestroy() {
      super.onDestroy();
      aThread.close();
   }
}
```

Şekil 4. ThreadNotLeakedActivity.java sınıfı (sızıntı yapmayan sınıf örneği)

3.2.2. İş Parçacığı Önceliklerinin Ayarlanmaması

Performansı artırmak için çoklu kullanım (multithreading) gerekli bir durumdur. Ancak uygulamada bulunan bu iş parçacıkları kaynaklar için ana iş parçacığı ile yarışabilir. Bu durumdan kaçınmak için iş parçacıklarının önceliği "android.os.Process.setThreadPriority" metodu ile ana iş parçacığından daha düşük seviyeye (arka planda çalışması istenen bir iş parçacığı için: "android.os.Process.THREAD_PRIORITY_BACKGROUND) ayarlanmalıdır. Bu düzenleme "run" metodunun başında yapılmalıdır.

İş parçacıkların önceliklerini belirlememek ANR (Uygulama Cevap Vermiyor) ekranının ortaya çıkmasına sebep olabilir. Örnek olarak, ana iş parçacığına cihazın önemsenecek miktarda kaynağını kullanmak zorunda olduğu görevlerin verildiği durumu düşünelim. Aynı zamanda, ağ işlemleri yapan bir iş parçacığı başlarsa ve bu önemsenecek miktardaki kaynağı uzun süre hiç bırakmazsa, iki iş parçacığının öncelikleri eşit olduğu için, aynı kaynağa ihtiyaç duyduklarında, ağ işlemleri yapan iş parçacığı bu kaynağı elinde tutar ve ana iş parçacığı çalışmasına diğer iş parçacığı işini bitirene kadar devam edemez. Bu durum ara yüz işlemleri dâhil bütün işleri geciktirir ve kullanıcı uygulamadan bir süre cevap alamaz.

4 Hata Örüntüsü Yakalama Aracı

Bölüm 3 ile açıklanan hata örüntülerini yakalamak için bir Java programı geliştirilmiştir. Bu araç Android Lint (bakınız Bölüm 1.1) çerçevesi üzerine kendi özel detektörlerimizi gerçekleştirerek oluşturulmuştur. Lint çerçevesinin özellikleri kullanan araç uygulamanın sınıflarını, bayt kodlarını ve XML dosyalarını tarayabilmektedir. Araç içinde dört detektör bulunmaktadır ve bu detektörler Bölüm 1.1 de bahsedilen araçların bulmadığı üç hata örüntüsünü ("Statik Olmayan İç Sınıf Kullanımı", "İş Parçacıklarında İptal Mekanizması Belirlememek", "İş Parçacığı Önceliklerinin Ayarlanmaması") bulabilmektedir.

"InnerClassLeakDetector", statik olmayan iç sınıfları bulmak için ".class" türünden olan bütün dosyaları tarar. Tarama sırasında Şekil 5'deki algoritmayı uygular.

```
function checkclass(classNode) {
   if name of classNode contains "$" -1 then return;
   if classNode is an irrelevant android class then return;
   if classNode is android handler class then return;
   if classNode is not a static class then
      if classNode has a reference to outer class then
         report issue location in code with className
```

Şekil 5. Sızıntı yapan iç sınıf bulma algoritması

"ThreadPriorityNotSetDetector", önceliği ayarlanmamış iş parçacıklarını arar. Bu detektör uygulamanın bütün Java dosyalarını tarayarak "Runnable" türünden sınıfların "run" metotlarını incelenmek üzere bir ziyaretçiye gönderir. Bu ziyaretçi "run" metotlarının gövdesinde "setThreadPriority" geçen cümleyi arar.

"ThreadNoCancelationPolicyDetector", iş parçacıklarında döngü var iken iptal mekanizmasının kullanmamasını arar. Uygulamanın tüm Java dosyalarını tarayarak iş parçacığı kodlamasında iptal mekanizmasını arayan bir ziyaretçi oluşturur ve bu mekanizmanın "run" metodu içinde kullanılıp kullanılmadığını kontrol eder. Bunu yapmak için döngü cümleleri ve metotlar ziyaret edilerek iptal mekanizmasının olup olmadığına kestirilir. Bu detektörün tespit ettiği problem eksiksiz olarak çözülebilecek bir problem değildir. Bu yüzden burada kullanılan algoritma bir kısım durumlar göz ardı edilerek keşifsel bir yaklaşım kullanılmıştır.

"ListViewNoReuseDetector", uygulamanın tüm Java dosyalarını tarayarak liste adaptörlerinin "getView" metotlarını ziyaret eder. Bu metotta görünümün yeniden kullanılmasını arayan bir ziyaretçi başlatır.

5 Deneyler

Sunulan hata örüntüleri ile Android uygulamasının kullanıcı derecelendirmeleri arasındaki ilişki araştırıldı. Geliştirilen araç 100 açık kaynaklı Android uygulaması üzerinde çalıştırılarak her bir uygulama için bulunan hata örüntü sayıları elde edildi. Kullanıcı derecelendirmesi için her uygulamanın Google Play sayfasından değerlendirme verileri elde edildi. Bu veriler: indirilme sayısı, toplam puanlama sayısı, ortalama puan, her bir puan seçeneği için (1 den 5 e) toplam sayılarıdır. Tablo 1'de indirilen uygulamaların toplu istatistikleri görülebilir.

Tablo 1. İndirilen uygulamalar hakkında istatistiksel bilgiler

Kod satırı ortalaması	20801
5 puan alma ortalaması	1453
4 puan alma ortalaması	377
3 puan alma ortalaması	139
2 puan alma ortalaması	64
1 puan alma ortalaması	142
En düşük kod satırı sayısı-En yüksek kod satısı sayısı	~0,5 k – ~31,5 k

5.1 Puanlama sayıları ile Hata sayısı ilişkisi

İlk olarak uygulamaların kaç adet 1 puan aldığı ile içerdiği her bir hata örüntü sayısı arasında bir ilişki olup olmadığı incelendi. Uygulamalar arası tutarlılık olması açısından

puan adetleri min-max normalizasyonu yapıldı ve Pearson çarpım-moment korelasyon katsayısı hesaplandı. Aynı şekilde 2, 3, 4 ve 5 puan adetleri için de korelasyon katsayısı hesaplandı. Bunların yanı sıra uygulamaya verilen yüksek (4 ve 5) puan sayıları, uygulamaya verilen düşük (2 ve 1) puan sayıları ve indirilme sayısı için de korelasyon incelemesi yapıldı.

Tablo 2. Hata örüntü sayıları ile uygulamaya verilen puanların, uygulamaya verilen yüksek ve düşük puanların, indirilme sayılarının korelasyon değerleri

	Puan Sayısı					**Yüksek Puan sayısı**	**Düşük Puan sayısı**	**İndirilme sayısı**
	5	**4**	**3**	**2**	**1**			
İş parçacıklarında iptal mekanizması belirlememek	0,217	0,261	0,447	0,536	0,814	0,229	0,752	0,364
	p:0,030	p:0,009	p:0,000	p:0,000	p:0,000	p:0,023	p:0,000	p:0,000
Statik olmayan iç sınıf kullanımı	-0,073	-0,082	-0,092	-0,082	-0,072	-0,075	-0,076	-0,095
	p:0,471	p:0,420	p:0,363	p:0,416	p:0,478	p:0,456	p:0,451	p:0,348
Liste görünümünde görüntüleri yeniden kullanmama	0,116	0,158	0,156	0,157	0,059	0,125	0,089	0,120
	p:0,249	p:0,117	p:0,122	p:0,119	p:0,557	p:0,215	p:0,390	p:0,235
İş parçacıklarında önceliklerin ayarlanmaması	0,543	0,514	0,566	0,570	0,706	0,546	0,681	0,545
	p:0,000	p:0,000	p:0,000	p:0,000	p:0,000	p:0,000	p:0,000	p:0,000

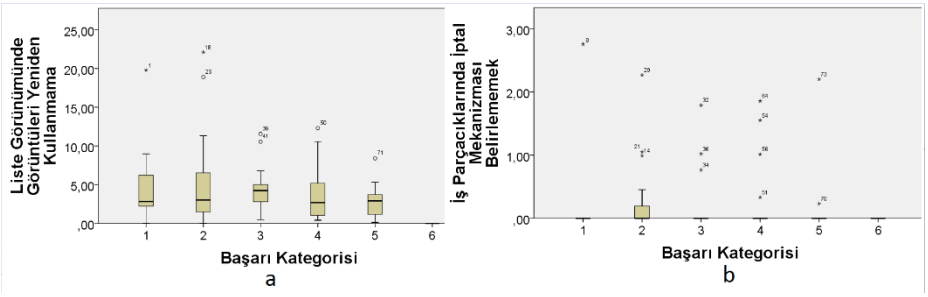
Tablo 2'ye bakıldığında "İş Parçacıklarında İptal Mekanizması Belirlememek" ve "İş Parçacığı Önceliklerinin Ayarlanmaması" durumları için p değerleri korelasyon katsayılarının kayda değer olduğunu göstermektedir. Bu katsayılar ise bu iki hata örüntüsünün olumsuz puanlarla (1'e yakın olan puanlar) olumlu puanlara (5'e yakın puanlar) göre daha pozitif bir doğrusal bağlantı içerisinde olduğunu anlatmaktadır.

5.2 Ortalama Puanlar ile Hata sayıları ilişkisi

Bir diğer analiz de uygulamaların ortalama puanları üzerinden yapıldı. Bunun için uygulamalar başarılılık oranlarına göre 6 kategoriye ayrıldı. Bu kategorilendirme Tablo 3'de gösterilen koşullara göre yapıldı.

Tablo 3. Kategorilendirme koşulları

Kategori	**Koşul**
Çok Başarısız (1)	Puan ortalaması < 3,5
Başarısız (2)	3,5 <= Puan Ortalaması < 3,8
Orta Başarılı(3)	3,8 <= Puan Ortalaması < 4,1
İyi (4)	4,1<= Puan Ortalaması < 4,4
Başarılı (5)	4,4 <= Puan Ortalaması < 4,7
Çok başarılı (6)	4,7 <= Puan Ortalaması

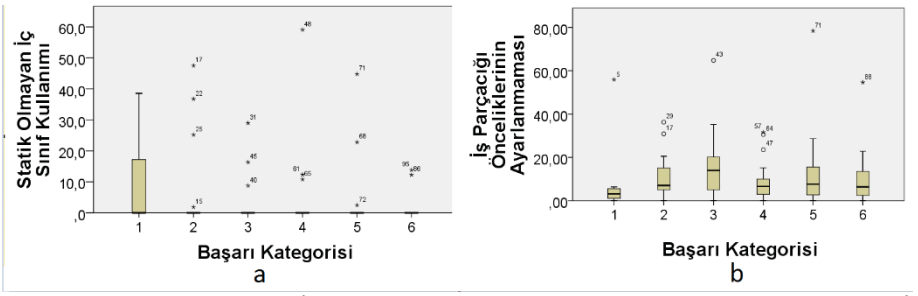


Şekil **6. a**. "Liste Görünümünde Görüntüleri Yeniden Kullanmama" ve Başarı Kategorisi **b**. "İş Parçacıklarında İptal Mekanizması Belirlememek" ve Başarı Kategorisi

Bu kategoriler ile her bir hata örüntüsü için kutu grafikleri Şekil 6 ve 7 ile gösterilmiştir.

Şekil 6.a "Liste Görünümünde Görüntüleri Yeniden Kullanmama" sayısının uygulamanın satır sayısına göre normalize edilmiş değerini ile başarı kategorileri arasındaki ilişkiyi vermektedir. Grafikte görüldüğü üzere bu hata sayısı 1. kategoriden (Çok başarısız) 3. kategoriye (Orta başarılı) kadar artış göstermekte fakat sonrasında düşüşe geçmektedir. Ayrıca "Çok başarılı" kategorisinde bu tip hatanın da hiç bulunmaması, bu hata örüntüsünün bulunma miktarının kullanıcıların uygulamaya verdikleri puanı etkilediğini göstermektedir. Bu kutu grafiği üzerinde ayrıca Kruskal Wallis testi uygulanmış ve grafiğin kayda değer olduğu görülmüştür (p değeri 0,0001).

Şekil 6.b'de verilen grafikte sıfır değerli ortalama çizgileri görülmektedir ki, "İş parçacıklarında İptal Mekanizması Belirlememek" durumu genel olarak uygulamalarda az sıklıkla bulunmaktadır. Bu grafikte başarı kategorisi ile bir korelasyon görülememektedir.



Şekil 7. a. "Statik Olmayan İç Sınıf Kullanımı" ve Başarı Kategorisi Kutu Grafiği **b.** "İş Parçacığı Önceliklerinin Ayarlanmaması" ve Başarı Kategorisi Kutu Grafiği

Şekil 7.a'da görüldüğü üzere "Statik Olmayan İç Sınıf Kullanımı" sayısı "Çok Başarısız" kategorisinde nispeten yüksek, diğer kategorilerde sıfır seviyesindedir. Bu hata örüntüsü uygulamaların "Çok Başarısız" kategorisine girmesinin sebeplerinden biri olabilir. Şekil 7.b'de ise "İş Parçacığı Önceliklerinin Ayarlanması" sayısının, başarı kategorilerine göre artıp azalma şeklinde düzensiz bir değişim gösterdiği gözlenmektedir. Bu grafik üzerinde Kruskal Wallis testi uygulanıp bu grafiğin kayda değer olduğu gözlenmesine rağmen, görülen düzensiz değişim bu tip hata ile başarı

kategorilerinin arasındaki bağlantının bu grafik ile ortaya çıkartılamayacağını göstermektedir.

5.3 Kategorilerin Ortalama Puanları ile Hataların Ortalama Sayıları İlişkisi

Tablo 3'de gösterilen her grup için her hata örüntüsünün ortalama sayıları bulundu. Normalizasyonu sağlayabilmek için, her uygulamada bulunan hata örüntüsü sayısı bu uygulamaların satır sayısına bölündü. Grupların hata örüntü ortalamaları ile uygulamaların kullanıcılardan aldığı puanlar arasındaki ilişkiyi incelemek için Kendal tau, Pearson çarpım-moment ve Spearman'ın sıralama korelasyon katsayısı değerleri bulundu. Sonuçlar Tablo 4 ile gösterilmiştir. Bu verilere göre, "Statik Olmayan İç sınıf Kullanımı", "İş Parçacıklarında İptal Mekanizması Belirlememe" ve "Liste Görünümünde Görüntüleri Yeniden Kullanmama" sayıları ile kullanıcılardan alınan puanlar arasındaki korelasyon katsayısı -1'e yakındır ve hepsi kayda değerdir (p değer< 0.05). Katsayının -1'e yakın olması gösteriyor ki bu üç hata örüntüsünün uygulamalarda bulunması kullanıcıların verdiği oylarda negatif doğrusal bir etkiye sebep olmaktadır. Öte yandan "İş Parçacığı Önceliklerinin Ayarlanmaması" ile kullanıcı puanları arasında bir ilgileşim olup olmadığına karar verilememektedir.

Tablo 4. Grupların normalize edilmiş ortalama hata örüntüsü sayılarının Kendal's tau, Pearson çarpım-moment ve Spearman'ın sıralama korelasyon

	Korelasyon türü					
	Kendall tau		Pearson çarpım-moment		Spearman'ın sıralama	
Hata örüntüsü sayısı türü	Katsayı	p değeri	Katsayı	p değeri	Katsayı	p değeri
Liste görünümünde görüntüleri yeniden kullanmama	-0,867	0,015	-0,833	0,039	-0,943	0,005
İş parçacıklarında iptal mekanizması belirlememe	-1	0	-0,84	0,036	-1	0
Statik olmayan iç sınıf kullanımı	-0,867	0,015	-0,945	0,005	-0,943	0,005
İş parçacığı önceliklerinin ayarlanmaması	0,067	0,851	0,058	0,913	0,143	0,0787

6 Sonuç

Android işletim sistemi hitap ettiği kitlenin büyüklüğü ile bilişim dünyasındaki yerini sağlamlaştırırken uygulama sayısı da artmaktadır. Bu çalışma bir uygulamada uygulamanın içeriğinin ve kullanışlılığının yanı sıra, geliştiricinin yaptığı yanlışlardan kaynaklanan sorunların kullanıcı derecelendirmelerinde ne kadar etkili olabileceği incelenmiştir. Android uygulama geliştirme dünyasında birçok sayıda öğretici ve tecrübe artırıcı materyal bulunmaktadır. Bu çalışma bahsedilen materyallerde şimdiye kadar statik analiz araçlarının tespit etmediği hata örüntülerini otomatik olarak bulan bir aracın geliştirilmesini sağlayarak büyük bir boşluğu doldurmuştur. Bunun dışında geliştirilen araç 100 uygulamanın üzerine uygulanmış ve geliştirilen aracın uygulanabilirliği gösterilmiştir. Yapılan analizlerde görülmüştür ki, otomatik tespit ettiğimiz 4 hata örüntüsünden 3'ü kullanıcıların uygulamaya verdiği puanı doğrudan veya dolaylı olarak etkilemektedir. Bahsedilen hata örüntülerinin uygulamanın

çökmesini, donmasını ya da yavaşlamasını tetikleyen yöntemler olması, kullanıcıların uygulamalara puan verirken ki kıstaslarının da görülmesi açısından önemli bir çıktı olmuştur. Bu durum kullanıcıların daha kararlı uygulamaları tercih etmesine ve bu tercih de kalburüstü geliştiricilerin el üstünde tutulmasına sebep olmaktadır.

Bu çalışma için gelecek planları arasında, geliştirilen aracın optimize edilerek değerlendirilmesi, tespit edilen hata örüntüsü sayısının arttırılması, daha büyük bir denek uygulama kümesi üzerinde analizlerin yapılması bulunmaktadır.

7 Referanslar

1. M. Burton and D. Felker, Android Application Development for Dummies, Wiley, 2012.
2. R. Meier, Professional Android 2 Application Development, Wrox, 2010 .
3. "Creative Bloq," [Online]. Available: http://www.creativebloq.com/app-design/how-build-app-tutorials-12121473. [Accessed 11 11 2013].
4. C. Smith, "Free online Android programming". [Online]. Available: http://www.androidauthority.com/free-online-android-programming-course-327826/. [Accessed 18 01 2013].
5. Android Open Source Project, "Core App Quality Guidelines," [Online]. Available: http://developer.android.com/distribute/googleplay/quality/core.html. [Accessed 2013].
6. "Android Tools Project Site," [Online]. Available: http://tools.android.com/tips/lint. [Accessed 02 12 2013].
7. Android Open Source Project, "UI/Application Exerciser Monkey," [Online]. Available: http://developer.android.com/tools/help/monkey.html. [Accessed 25 12 2013].
8. D. Hovemeyer and W. Pugh, "Finding Bugs is Easy," in OOPSLA 2004, Vancouver, BC, Canada, 2004.
9. N. Ayewah and W. Pugh, "The Google FindBugs Fixit," in ISSTA'10, Trento, 2010.
10. Cunningham & Cunningham, Inc., "Inner Class," 16 4 2010. [Online]. Available: http://c2.com/cgi/wiki?InnerClass. [Accessed 05 12 2013].
11. Compuware, "Java Enterprise Performance Book (dynatrace)," [Online]. Available: http://javabook.compuware.com/content/memory/how-garbage-collection-works.aspx. [Accessed 2012].
12. L. Rocha, "Lucas Rocha Blog," 05 04 2012. [Online]. Available: http://lucasr.org/2012/04/05/performance-tips-for-androids-listview/. [Accessed 06 12 2013].
13. K. Lee, J. Flinn, T. Giuli, B. Noble and C. Peplin, "AMC: verifying user interface properties for vehicular applications," in MobiSys '13, New York, 2013.
14. E. Chin, A. P. Felt, K. Greenwood and D. Wagner, "Analyzing inter-application communication in Android," in *MobiSys '11*, New York, 2011.
15. A. Pathak, A. Jindal, Y. C. Hu and S. P. Midkiff, "What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps," in *MobiSys '12*, New York, 2012.
16. A. P. Felt, E. Chin, S. Hanna, D. Song and D. Wagner, "Android Permissions Demystified," in *CCS 2011*, Chicago, 2011.
17. Y. Liu, C. Xu, "VeriDroid: Automating Android Application Verification", MDS '13 Proceedings of the 2013 Middleware Doctoral Symposium.
18. Y. Liu, C. Xu, S. Cheung, "Characterizing and Detecting Performance Bugs for Smartphone Applications", *ICSE '14, Hyderabad, India, 2014*

EK 1

Hata Örüntüleri	Hata Örüntü Türü	Karşılaşılma Sayısı	Karşılaşılma Yüzdesi
Statik olmayan iç sınıf kullanımı	Bellek kullanımını etkileyen hata	1047	8,01
İş parçacığı önceliklerinin ayarlanmaması	İşlemci kullanımını etkileyen hata	290	2,22
Liste görünümünde görünümlerin yeniden kullanılmaması	Bellek kullanımını etkileyen hata	622	4,76
Düzen tasarımlarının iç içe kullanımı	Bellek kullanımını etkileyen hata	966	7,39
Aktivite kontekstlerinin geçirilmesi	Bellek kullanımını etkileyen hata	5929	45,61
Çok fazla çok büyük resimler yüklenmesi	Bellek kullanımını etkileyen hata	601	4,60
Ana iş parçacığında çok uzun sürme ihtimali olan veri tabanı, ağ işlemlerinin yapılması	İşlemci kullanımını etkileyen hata	270	2,07
Performans hassasiyeti olan birimlerde çok uzun sürme ihtimali olan veri tabanı, ağ işlemlerinin yapılması	İşlemci kullanımını etkileyen hata	60	0,46
Görünüş değişimlerinin ele alınmaması	Bellek kullanımını etkileyen hata	1203	9,21
İzin hataları	Konfigürasyon hatası	331	2,53
Statik görüntülerin tutulması	Bellek kullanımını etkileyen hata	323	2,47
Doğrudan programın içine gömülen dosya adresleri	Konfigürasyon hatası	61	0,47
Cihazın API uygunluğunun o API'yi kullanılmadan önce kontrol edilmemesi	Konfigürasyon hatası	1139	8,72
Çalışma zamanının servisleri öldürmesine izin verilmemesi	Bellek kullanımını etkileyen hata	194	1,48