Yazılım Geliştirme ve Test Otomasyon ile Verimlilik Artışı: General Mobile

Productivity Increase with Software Development and Test Automation: General Mobile

Halil Ahmet YENER[1], Furkan BAŞTÜRK[2], Berkin MEÇİK[3]

[1]Kalite Standartları ve Test Takım Lideri, General Mobile, Türkiye

[2]Kalite Standartları ve Test Mühendisi, General Mobile, Türkiye

[3]Kalite Standartları ve Test Mühendisi, General Mobile, Türkiye

*Sorumlu Yazar: ahmet.yener@generalmobile.com

Öz

Günümüzde birçok sektör, yazılım sistemlerini temel bileşen olarak kullanmaktadır. Yazılım üretmenin en kritik ölçütü olan kalitenin gerçekleştirilebilmesi için gereksinimlerin doğru anlaşılması ve testlerin hatasız yapabilmesi gerekmektedir. Yazılım kalitesinden emin olmak için gerçekleştirilen manuel testler süresince zaman kaybı, hata oranı ve test süresi artabilmektedir. Bu tip durumları en aza indirmek için test otomasyon çalışmaları yapılmaktadır. Testlerin verimli ve hızlı gerçekleşmesi, yazılım geliştirme süreçlerinde önemli bir katkı sağlamaktadır. Test otomasyonu bu ihtiyacın karşılanması için önemli bir yazılım unsuru olarak görülmektedir. Otomasyon ise zaman ve maliyet tasarrufu ile birlikte dikkat çekici bir fayda sağlamaktadır. Ülkemizde ise geçmişte yazılım-test ve kalite departmanlarına verilen önem, uygulama ve ürünlerdeki kalite sorunlarıyla beraber son kullanıcıların yaşadığı olumsuz deneyimler nedeniyle her geçen yıl daha fazla artmaktadır. Bu çalışmada, General Mobile Test ve Kalite ekibi olarak son kullanıcı testlerinde otomasyon testlerinin, manuel testlere göre verimlilik karşılaştırılması yapılmış ve mobil cihaz yaşam döngüsünde otomasyon temelli verimlilik artışı modeli üzerine çalışılmıştır.

Anahtar Kelimeler: Yazılım, Otomasyon, Test, Verimlilik, General Mobile

Abstract

Many sectors use software systems as a basic component. To realize the quality, which is the most critical criterion of software production, the requirements must be understood and the tests must be issue-less. During manual tests to ensure quality, time loss, error rate and test time may increase. To minimize such situations, automation studies are carried out. Efficient testing is an important for software development processes. Automation is seen as an important software element. Automation provides a significant benefit with time and cost savings. The importance given to software-test and quality departments in the past increases with each passing year due to the negative problems experienced by end-users along with quality problems in applications and products. As General Mobile Test and Quality team, efficiency comparison of automation tests in end user tests compared to manual tests has been made and automation-based productivity increase model has been studied in mobile device.

Keywords: Software, Automation, Testing, Productivity, General Mobile

1. Giriş

Dünyada gelişen teknolojiyle birlikte yazılım sistemleri, birçok sektörün temel bileşenlerini oluşturmaktadır. Bankacılık, telekomünikasyon, savunma sanayisi gibi bazı iş alanlarının, gerekli yazılımlar olmadan yaşamını sürdürmesi mümkün gözükmemektedir. Her geçen gün artan rekabet ortamı ve gelişen teknolojinin etkisiyle yazılım sistemlerine duyulan ihtiyaç artmaktadır. Yazılım sektörü de diğer sektörlerde olduğu gibi müşterinin, son kullanıcının ihtiyaçlarını doğru gereksinimler ile analiz ederek ürün çıkarmaktadır. Bu analiz sonucunda geliştirilen ürün veya uygulamaların son kullanıcıyı memnun etmesinin başlıca unsurlarından biri de kalitedir.

İlk olarak, Martin Fowler mevcut kodların iyileştirilmesi için Kalitesiz Kod Göstergeleri kavramını öne sürmüştür [1]. Bu kavram göz önünde bulundurulduğunda, yazılım maliyetlerini azaltmanın yollarından biri de kaliteli yazılım üretilmesiyle veya mevcut uygulamaların kalitesini artırmakla sağlanabildiği görülmektedir.

Test, ürün veya uygulamanın istenen kriterlere uygunluğu, belirlenen prosedürlere göre işletilen bir süreçtir. Sürecin doğruluğu, uygulanan testin kalitesini verir [2]. Yazılım geliştirme süreci ile birlikte test faaliyetlerinin başlaması gerekmektedir. Bu sayede geliştirilen ürün veya uygulamanın kalitesinde artış sağlanmaktadır. Ayrıca bu çalışma, projenin başarısız olma riskini azaltabilir.

Yazılım test süreci, geliştirilen yazılımdaki hataların varlığına odaklanır. ISO/IEC yazılım testini, uygulamanın veya sistemin belirli koşullara göre denetlenebilir, işletilebilir olması ve sonuçların değerlendirilmesi olarak tanımlamıştır [3]. Yazılım testinin amacı kaliteyi arttırmak, istenilen kriterlerin sağlanıp sağlanmadığını kontrol etmek ve bu sonuçları raporlamaktır.

Fakat test otomasyonu yapılırken karşılaşılan sorunlar da yapılan test otomasyonun etkinliğini azaltabilmektedir. Örneğin, test yapan uzmanın kodlama gibi teknik alt yapı gerektiren konularda yazılım geliştirici kadar bilgi sahibi olması gerekmektedir. Yazılan kodların bakımının yapılması, otomasyonda kullanılan test verisinin kodla iç içe geçmesi ve yazılan otomasyon kodlarının çalıştırılacağı ortamların yönetilmesi, sonuçların doğru takip edilebilmesi gibi sorunlar da bulunmaktadır.

2. Test Otomasyon İhtiyacı

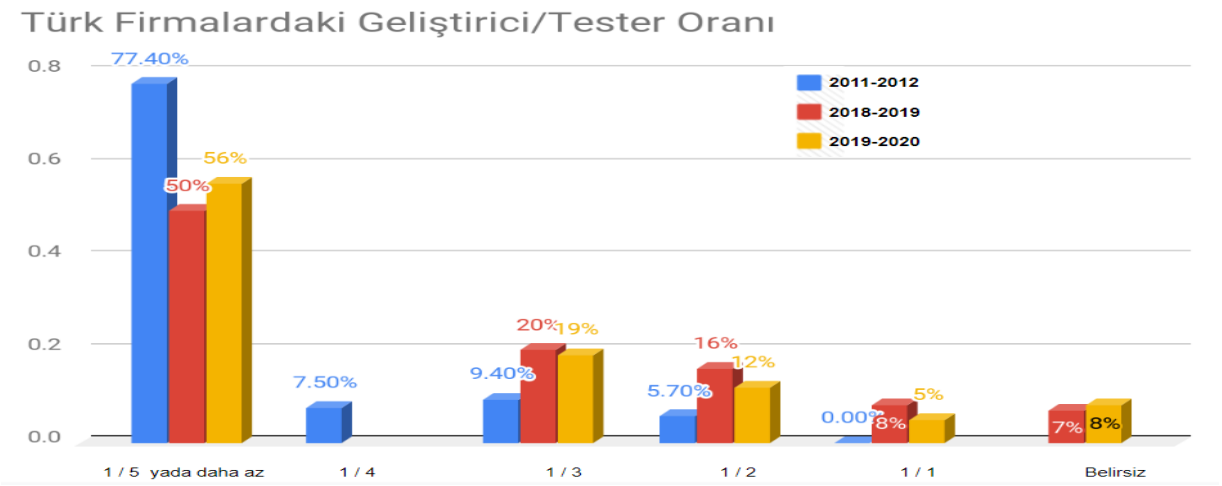
Günümüz teknolojisinde mobil cihazlar giderek daha etkin kullanım alanları bulmakta, kullanıcılar kişisel bilgisayarlarını açmadan hızlı işlem yapma ihtiyacını mobil cihazlar ile gidermek istemektedirler.

Akıllı evler ve ev aletleri, saatler, küçük sağlık cihazları, klimalar, ulaşım alanındaki otonom araçlar gibi pek çok araç mobil cihazlarla entegre edilmeye başlanmıştır. Bu durum mobil cihaz üretiminde sık sık güncelleme gerektirmekte, daha kaliteli ve daha etkin yazılım ihtiyacı doğurmaktadır. Mobil cihaz çeşitliliği giderek artmakta bu nedenle cihazlarda manuel olarak test yapmak giderek zorlaşmaktadır. Akıllı telefon kullanıcılarının en temel beklentileri: sahip olduğu ürünlerin hatasız çalışmasını, son teknoloji özelliklerine sahip olmasını ve bununla birlikte ürünün düşük maliyette olmasını isterler.

Üretici firmaların son teknoloji ürününü pazardaki rakiplerine göre daha hızlı ve kaliteli çıkarabilmesi önemlidir. Bu sebeple cihazın kalitesinde donanım güvenliği kadar uyumluluk, yazılımın güvenilirliği, uygulamanın işlevselliği gibi özellikler önem kazanmaktadır. Bunları sağlayabilmek adına yazılım testlerinin önemi giderek artmaktadır. Yazılım testleri, yazılımın zayıf yönlerini ve içerisinde barındırdığı hataları tespit ederek yazılımın kalitesini artırmak, güvenilirliğini sağlamak, doğruluğunu ve geçerlemesini sağlamak içindir [4]. Mobil uygulamalar için Mahadev Satyanarayanan "kısıtlı kaynak, güvenlik ve performans, güvenilirlik değişkenliği ve sonlu enerji kaynağı" adlarında dört kısıtlama belirtmiştir [5]. Bu kısıtlamaların doğurduğu zorluklar neticesinde her yeni çıkan cihazda testleri yapmak daha karmaşık ve pahalı olmaktadır. Mobil uygulamalar mobil test GUI uygulamalarının kullanıcı etkileşimine odaklanmış olduğundan özel zorluklar doğurmaktadır. GUI uygulamaları olay güdümlüdür ve kararsız bir yapıya sahiptir. Bu durum da mobil uygulamaların testini zorlaştırmaktadır [5]. Pek çok kişi tarafından mobil uygulamalar için test ihtiyaçları ve bu ihtiyaçların uygulanabilirliği konusunda çalışmalar yapılmış, mobil uygulama testlerinde karşılaşılan zorluklar ve getirilen yenilikler ile zaman faktörü göz önüne alınarak analiz edildiğinde otomasyonun gerekliliği önem arz etmektedir. Test senaryosunun sayıca çok olduğu bir projede regresyon testlerinin manuel ve sürekli olarak koşulması iş yükünü arttıracaktır. Manuel test süreçlerinin yerine test otomasyonu ile senaryoların otomatize edilmesinin ardından şu faydaları sağlayacaktır:

- Regresyon testlerinin daha sık koşulabilmesini sağlar.
- Gerçekleştirilen testlerin kapsama alanını ve derinlik seviyesini yükseltir.
- Manuel teste göre gözden kaçabilecek hataları bulabilir.
- Uygulamaya olan güveni arttırır.

Benzer sebeplerden dolayı firmalar son dönemde bu konuya ağırlık vermiş bunun sonucunda da test ve kalite ekiplerine verilen önem ve ayrılan bütçe artmıştır.



Şekil 1. "Turkish Testing Board" tarafından her sene yayınlanan "Türkiye Yazılım Kalite Raporu" adlı yayınının 2011,2018 ve 2019 yayınları incelenerek oluşturulan firmaların yazılım geliştiricisi ve yazılım test alanında çalışan işçi oranları göstermektedir [6].

Şekil 1'de görülebileceği üzere Türkiye'deki firmaların birçoğu artık bütçe anlamında test ekiplerine daha fazla kaynak ayırmaktadır.

3. Mobil Test Otomasyon Geliştirilmesi

Ülkemizde ve dünyada mobil yazılım geliştiricilerinin kalite ve verimi artırmak adına edindikleri deneyimler sonucunda belli ilkeler ortaya konmuştur. Bu doğrultuda mobil test otomasyonunu en etkin şekilde gerçekleştirebilmek adına bazı tekniklerin olduğu bilinmektedir.

3.1 Test Sürecindeki İlkeler

3.1.1 Test Edilebilir Hale Getirmek

Testin neden yapılamadığını çözmek için uygulamanın özelliklerini iyi bilmek, yeniden düzenleme yapabilmek ve sonrasında yazılım kodlarını basitleştirmek gerekir.

3.1.2 Solid Kurallarını Uygulamak

SOLID, programlama ve yazılım tasarımlarını daha anlaşılır, esnek ve sürdürülebilir hale getirmeyi amaçlayan beş tasarım ilkesinin hatırlatıcı bir kısaltmasıdır.

- Single responsibility principle: Mobil uygulamanızın her bir sınıfının yalnızca bir ve tek sorumluluğu olmalıdır.

- Open–closed principle: Nesneye yönelik programlamanın en önemli adımıdır. Yazılım varlıkları (sınıflar, modüller vb.) diğer varlıklar tarafından kullanıma açık olmalı ancak her yeni eklenen özellik ile de varlık yapısının değişmemesi gerekmektedir [7].
- Liskov substitution principle: Uygulamanızın kodundaki nesneler/objeler başka örneklerle değiştirilebilir.
- Interface segregation principle: Genel amaçlı oluşturulan ara yüzde değişiklik yapıldığı zaman istemcilerde sorun yaşanmaktadır. Bunun için genel ara yüz altında istemciye özel ara yüzler geliştirilmelidir [8].
- Dependency inversion principle: Bir sınıfı birkaç parçaya bölerek onları bağımsızca test edebilmektir.

Kalite güvencesi noktasında S, L ve D harfleri test vakalarınızı sağlamlaştırmak için en yararlı olanlarıdır.

3.1.3 Gerçek Cihazlar İle Test Etmek

Testlerin gerçek cihazlar ile yapılması fiziksel cihaz kullanımına bağlı hataların tespit edilmesi ve sorunların giderilmesi konusunda sanal cihaz kullanımına göre daha avantajlıdır.

3.1.4 Nerede Otomasyon Nerede Manuel Test Yapılacağına Karar Vermek

Manuel ve otomasyon test yöntemlerinin kendi aralarında güçlü ve zayıf yönleri vardır. Bu sebeple iki yöntemde güçlü oldukları kısımlarda kullanılmalıdır. Otomasyon testleri kullanıcıların deneyimlerini destekleyemez. Pestisit etkisine göre; Otomasyon testleri uzun dönemlerde etkilerini kaybeder bu sebeple test senaryo kodlarının (script) devamlı güncellenmesi gerekir

3.1.5 Doğru Test Otomasyon Aracını Seçilmesi

Yapılacak manuel ve otomasyon testlerinde kullanılacak olan araçların ve/veya uygulamaların doğru bir şekilde seçilmesi ve efektif olarak kullanılması.

3.1.6 İyi ve Kaliteli Test Verisi Oluşturma

Yapılan testler sonucu elde edilen verileri düzgün bir şekilde kayıt altına almak ve sonrasında yapılan olan test verileri ile kıyaslama yapabilmek.

3.1.7 Değişikliklere Karşı Adaptasyon

Test edilen uygulamada meydana gelen herhangi bir değişime karşı yeni test senaryolarının oluşturulması.

3.2 Manuel ve Otomasyon Testlerinin Karşılaştırılması

Firmalar ürünlerini farklı standartlara uygun olarak ürün amacına uygun testlere tabi tutarlar. Bu testler, eskiden sadece manuel olarak insan gücüyle gerçekleştirilebiliyor iken günümüzde sağladığı hız ve verimlilik gibi başlıca sebepler dolayısıyla otomasyon testi rağbet görmektedir [9].

Manuel Test: Fiziksel çaba ve zaman gerektirir. Bulgular test eden kişi tarafından kayıt altına alınmalıdır. Kısa süreli bazı proje ve testlerde manuel test tercih edilse bile uzun süreli testlerde testi yapan kişide tünel etkisi gibi sebeplerden dolayı insandan kaynaklı hatalar görülebilir.

Tünel Etkisi: Bir olayın tek yönüne odaklanıldığında diğer faktörlerin gözden kaçırılmasıdır.

Otomasyon Testi: Kod yazma kısmı uzun sürer ve devamlı olarak bakım gerektirir. Manuel teste oranla daha hızlı ve verimlidir. Kolay raporlama yapılır. Uzun süreli ve tekrar eden testler için manuel teste göre oldukça avantajlıdır.

Tablo 1. Otomasyon ve manuel testin özellikleri

	Otomasyon	**Manuel**
Maliyet	Kullanılan araçlar ve devamlı olarak bakım gerektirmesi dolayısıyla pahalı olabilir. Fakat açık kaynaklı araçlar kullanılır ise maliyet düşer.	İnsan gücüne başvurduğu için personel maliyeti doğurur.
Verimlilik	Tekrar eden testler için uygundur.	Tekrar eden testler için uygun değildir.
Raporlama	Çeşitli uygulamaları desteklediği için otomatik bir şekilde düzenli rapor tutulabilir.	Veriler testi yapan kişi tarafında kayıt edilmelidir.
Kapsam	Sadece kodlanan kısımları kontrol eder.	En küçük ayrıntı bile test edilebileceği için daha geniş kapsamlıdır.
Zaman	Hızlıdır.	Yavaştır.
Gözlem	Kullanıcı deneyimi sağlayamaz.	Kullanıcı deneyimi sağlar.
Güvenilirlik	İnsan kaynaklı hatalar görülmez.	Manuel testler hiçbir zaman tam olarak tekrar edilemez. İnsan hataları görülebilir.
Hata bulma oranı	Manuel teste göre daha fazla hata bulunmasını sağlar.	Otomasyon sonucu bulunan bazı hatalar bulunamayabilir.

Tablo 1'de görülebileceği üzere otomasyon testi manuel teste göre daha avantajlıdır. Ancak bazı test senaryolarının insan gücü ile koşulması gerekliliği aşikârdır. Gerçekleştirilen testlere bakıldığında manuel ya da otomasyon kullanımı Tablo 2'de gösterilmiştir.

Tablo 2. Otomasyon ve manuel olarak yapılabilecek testler

Otomasyon	**Manuel**
Regression	Araştırma Testi (Exploratory)
Yük Testi (Load)	Kullanılabilirlik Testi (Usability)
Tekrarlanan Testler (Repeated)	Geçici Test (Ad-Hoc)
Performans Testi	Beta Testi
Stress Testi	GUI yazılım Testi
Sanity Testi	

Regression: Yapılan değişikliklerin kontrol edilmesidir.

Load Test: Kapasite ölçümü için yapılan testtir.

Repeated(Loop) Test: Devamlı olarak yapılan testtir.

Performans Testi: Sistemin performans ve sistem bileşenlerinin uygunluğunun kontrol edilmesidir.

Stress Testi: Belirtilen sınırlar dışında sistemin test edilmesidir**.**

Sanity Testi: Yazılım güncellemesi sonrası yapılan performans testidir.

Exploratory Test: Keşif testidir.

Usability Test: Merkezli etkileşim tasarımında kullanılan ve son kullanıcılar üzerinde kullanılan test tekniğidir.

Ad-Hoc Test: Bütün test sürelerinin belirlenmesidir.

Beta Test: Yazılımın ilk sürümü sırasında yapılan testlerdir.

GUI Yazılım Testi: Kullanıcı ara yüzünün, uygulamanın kullanıcı dostu olup olmadığının test edilmesidir.

Tablo 1 ve Tablo 2'de belirtilen durumlardan dolayı her ne kadar çoğu test otomatize edilebilse bile mevcut durumda manuel testin varlığını sürdüreceği öngörülmektedir.

3.2 Gelecekteki Test Trendleri

Teknolojik gelişimin baş döndürücü ilerlemesi göz önüne alındığında yazılımcıların ve kalite birimlerinin işinin gittikçe zorlaştığı görülmektedir. Bu ilerlemeye istinaden otomasyon ihtiyacı da artmaktadır. Test yazılımı ihtiyacının da teknoloji ile paralel ilerlediği görülmektedir. Cep telefonlarının günlük hayatta giderek daha çok alanda ve daha çok cihazla entegre kullanılması için sadece işlevsellik ve güvenlik testi değil mobil uygulamalarında test edilmesi gerekecektir. Piyasada çok hızlı değişen mobil uygulamalar için şirketlerin odaklandığı konu müşterilerin deneyimlerini sorunsuz kullanılabilmesidir. Kullanıcıların deneyimlerinin test edilmesi, insan deneyimlerinin test edilmesi demektir. Otomasyon testleri ne kadar hızlı gelişirse gelişsin insan faktörüne dokunulduğu için kalite ve güvence konusunda manuel testler önemini yitirmeyeceği düşünülmektedir. Bu bağlamda hem manuel hem otomasyon testlerini doğru yaklaşımlarla kullanabilmek daha fazla önem kazanacaktır. Şekil 2'de gelecekte önem kazanacağı düşünülen test yöntemleri yüzdesel bazda gösterilmiştir.

Şekil 2. 2018 yılında "Turkish Testing Board" tarafından yapılan araştırmaya göre geleceğin popüler başlıkları [6].

3.3 Otomasyon Teknikleri

Gelişen teknoloji ile hızlanan modern cihazlar otomasyonun önemini gün geçtikçe arttırmaktadır. Otomasyon cihazlarının daha hızlı, daha hatasız ve daha hassas şekilde

çalışması, firmaların verimliliğini büyük oranda arttırmaktadır. Ayrıca kaliteli ürün çıkarılması, üretim maliyetinin düşmesi ve iş kazalarının azalması otomasyonun önemine bir kez daha dikkat çekmiştir. Otomasyonun en büyük dezavantajı ise kurulum maliyetidir.

Yazılım otomasyonlarında gelişen teknolojiler ve dünya çapında ortak çalışılan yazılım geliştirme çabaları sayesinde birçok yazılım araçları ortaya çıkmıştır. Bu yazılım araçları geliştirilecek yazılımın daha hızlı ve daha verimli yapılmasına yardımcı olmaktadır. Geliştirilen yazılım araçları firmanın ürün olarak çıkardığı lisanslı yazılım araçları veya birçok bağımsız yazılım geliştiricilerinin ortak çalışmaları ile geliştirilen açık kaynak yazılım araçları olarak iki çeşide ayrılır. Günümüzde etkin olarak kullanılan test yönetim araçlarından TET ve Test Manager, fonksiyonel test araçlarından Selenium, Solex ve Watir ve yük testi araçlarından Jmeter ve FunkLoad araçları açık kaynaklı yazılım araçlarıdır. Her yazılım aracının kendi içerisinde avantaj ve dezavantajları bulunmaktadır. Açık kaynaklı yazılım araçlarının ücretsiz olması, çok geniş çeşitliliğe sahip geliştiricileri ile önemli bir avantaj elde etmektedir.

Lisanslı olarak kullanılan test otomasyon araçlarından HP Quality Center, SMARTS ve TestLog, fonksiyonel test araçlarından Sahi, QuickTest Pro ve QA Wizard ve yükleme test araçlarından WebLOAD Professional ve HP LoadRunner yazılım araçlarını kurumsal düzeyde firmalar tarafından geliştirilmiştir. Lisanlı yazılım araçlarının en önemli avantajları hızlı entegre edilmesi ve teknik destek sağlanmasıdır. En büyük dezavantajı ise maliyettir. Lisanslı yazılım araçlarında lisansın alınması ve farklı platformlarda geliştirmek istenmesi ek maliyetler çıkarmaktadır. Açık kaynaklı yazılım araçlarında ise ücretsiz olması kendisini bir adım öne çıkarmasının yanı sıra teknik destek imkânı yazılım geliştirici topluluğu ile sınırlıdır.

General Mobile firması üretilen mobil cihazların fonksiyonel testlerini yapabilmek için yazılım ağırlıklı otomasyon geliştirmiştir. Bu otomasyon çalışmasında yapılacak olan fonksiyon testleri popüler olan sosyal medya uygulamalarını kapsamaktadır. Sürekli yeni özellikler eklenmesi ve cihaz teknik desteğinin sağladığı özelliklerden dolayı otomasyon yazılımını daima bir yenilik içerisine itmektedir.

Yazılım test araçların karşılaştırılması sonucunda desteklenen dil ve platform, performans, raporlama, açık kaynaklı olma ve çoklu test yapabilme özelliklerinden

dolayısıyla Selenium kullanılmasına karar verilmiştir. Selenium dışında kullanılmasına karar verilen diğer araçlar Appium, Cucumber, TestNG ve Maven'dır [10].

3.3 General Mobile'da Mobil Test Otomasyonu için Yazılım Geliştirme İhtiyaçları

Birçok firma güçlerini arttırmak için bilgi teknolojilerinden faydalanmaktadır. Bilgi teknolojilerini kullanırken sadece donanımsal cihazları kullanmak firmanın gücünü arttırmada bir noktaya kadar fayda sağlayabilir. Firmaların artan teknolojik gelişmeler ile dünya çapında bir geliştirme topluluğuna sahip yazılım kullanması büyük öneme sahiptir. Yazılım geliştirmek, firmaların sürdürülebilirliğini, verimlilik artışını ve büyüyebilmesinin en önemli etkenlerindendir. Yazılım geliştirmek için yazılım dillerinden faydalanılmaktadır. Yazılım dili, standart bir yapıda komutlar ve yazılımlar geliştirme olanağı sağlayan bilgisayar dili olarak tanımlanmaktadır [11]. Programlama dilleri geliştirici ile bilgisayarın verimli bir iletişim kurmasını sağlar. Geliştiricinin program dilleri ile daha verimli çalışmasına, işlemlerin daha hızlı yapılmasına yardımcı olan yazılıma Tümleşik Geliştirme Ortamı(IDE) denir. IDE'nin sağladığı faydaların yanında açık kaynak veya lisanslı üretilmiş yazılım parçaları ile geliştiricinin daha hızlı ve verimli yazılımlar geliştirmesini sağlayan yapılara yazılım araçları denir.

3.3.1 Programlama Dilleri

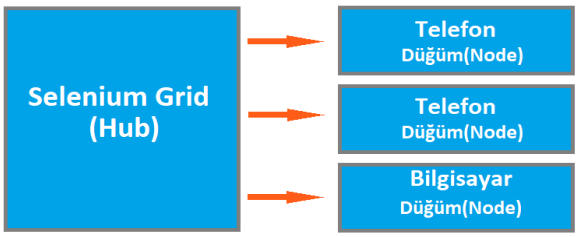
3.3.1.1 Java: Java, 1995 yılında ortaya çıkan bir programlama dilidir. Java yapmış olduğu yazılımlar ile bilgisayarlarda, internette, oyun konsolları ve süper bilgisayarlara kadar geniş çaplı uygulama çalıştırma desteği sağlamaktadır [12].

3.3.1.2 Gherkin: Gherkin, davranış odaklı geliştirme sürecini destekleyen Cucumber yazılım aracı için kullanılan yazılım dilidir. Yazılım metotlarını müşteriler tarafından anlaşılabilir cümlelere çevirir. Türkçe dâhil birçok dili desteklemektedir [13]. Gherkin ile yazılım metotlarını anlaşılabilir cümlelere çevirmesi sayesinde birçok bölüm ile anlaşılabilir bir dil oluşturmasının yanı sıra yazılan testlerin dokümantasyonu oluşturulmuş olur.

3.3.2 Yazılım Geliştirme Araçları

3.3.2.1 Selenium: Selenium, web uygulamalarını birçok platform üzerinden istenilen testleri optimize şekilde yapılmasını sağlayan açık kaynaklı pakettir. Yazılım kodu ile web tarayıcıları arasında haberleşmeyi sağlar [14]. Bu sebepten dolayı web tarayıcısında bulunan butonların, başlıkların, metinlerin vb. gibi birçok elementin özelliklerinden konumuna kadar çok sayıda bilgisini yazılım koduna aktarır. Yazılım kodu ise aktarılan bu bilgiler ile yapılması gereken işlemleri gerçekleştirir.

3.3.2.2 Selenium Grid: Selenium-Grid, testlerin paralel olarak birden fazla ortamda, farklı tarayıcılarda ve istenilen sayıda testin koşmasını sağlayan bir sunucudur [15]. Selenium-Grid, zaman açısından verimliliği artıran en etkili araç olarak bilinmektedir. Selenium-Grid bir bağlantı göbeği (Hub) olarak çalışır. Test yapılacak ortamlara düğüm (Node) denir. Düğümlerin bağlantı göbeğine bağlanması ile yapı oluşturulur. Bu yapıda koşulan testler paralel olarak düğümlere bağlantı göbeğinden dağıtılarak yapılır. Şekil 3'de Selenium Grid yapısı gösterilmiştir.



Şekil 3. Selenium Grid için bağlantı göbeği ile düğümlerin paralel çalışması

3.3.2.2 Appium: Appium, Android ve IOS mobil cihazlarında mobil web ve melez (hibrit) uygulamalarında istenilen testlerin optimize şekilde yapılmasını sağlayan açık kaynaklı bir pakettir [16]. Bir web sürücüsü olan Selenium'u kullanarak mobil cihazdaki uygulamalar ile yazılımsal kodu haberleştirir. Appium, mobil için Selenium'un özelleştirilmiş hali denebilir. Bu sebeple Appium, Selenium eklentileri olmadan kullanılamaz.

3.3.2.3 Cucumber: Cucumber, davranış odaklı geliştirme sürecini benimseyen yazılımcılar tarafından kullanılan bir yazılım paketidir. İstenilen davranışların müşteriler tarafından anlaşılabilir mantıksal bir dilde oluşturulmasını sağlar [17].

3.3.2.4 Test NG: Test NG, Java programlama dili için bir test aracı olarak tasarlanmıştır. Diğer test araçlarından esinlenerek ortaya çıkarılmıştır. Test NG, daha güçlü ve daha kolay bir kullanım sunmanın yanı sıra çok geniş bir test çeşitliliğine sahiptir [18].

3.3.2.5 Maven: Maven, projenin otomatik derlenmesini sağlar. Maven, projeyi derlemenin yanı sıra eklenecek kütüphanelerin de yönetimini sağlar. Bu sayede geliştiriciye birçok kolaylığı getirmiş olur. İçerisinde bulunan POM yapısı ile eklenen kütüphaneleri -yerelde bulunmaz ise- otomatik indirerek projeye ekler [19].

3.3.3 Yazılım süreçleri

Bir yazılım projesinin başarılı şekilde tamamlanması için doğru ve profesyonel şekilde yapılması gerekmektedir. Her adımın ayrıntılı şekilde planlanması hata ve kayıpları engeller. Bu sebeple başarılı bir süreç ile yazılım projelerinde maliyet, zaman israfı ve hataların önüne geçilmiş olur. Yazılım topluluğunda birçok kabul görmüş süreç vardır. Bu süreçlerin bir adımı da test olması nedeni ile ortaya birçok yazılım test süreci çıkmıştır. En kabul gören iki tanesi test odaklı geliştirme (TDD) ve davranış odaklı geliştirme (BDD) süreçleridir [20].

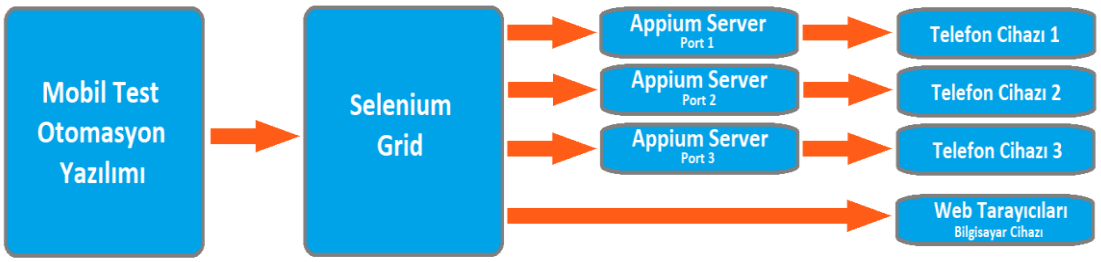
3.3.3.1 TDD: TDD, yazılımda yapılması istenilenlerin testinin oluşturulmasıyla bu testlerin çalıştırılıp hata vermesi sağlanır. Bu hataları çözecek şekilde kodlama yapılır ve yazılan testlerin hepsi başarılı olana kadar süreç devam eder. Testler başarılı olduktan sonra kodlar gözden geçirilerek yazılım projesi bitirilir [21].

3.3.3.2 BDD: Kullanıcıların talepleri üzerine ortaya çıkan davranışlar ile test oluşturularak yazılım tasarlanmaya çalışılır. Bu tarz geliştirilen yazılım sürecine davranış odaklı geliştirme denir. Davranış Odaklı Geliştirme, departmanlar veya kişiler arasında anlaşılabilir olmasını ve devamlılığını sağlar [20].

3.4 Otomasyon Temelli Yazılım Geliştirme Mimarisi

Günümüzde otomasyon birçok firma için verimliliği artıran en önemli etkenlerden biri olarak öne çıkmaktadır. General Mobile tarafından mobil cihazların testlerini daha hızlı, daha doğru ve daha kontrollü yapabilmek için otomasyon projesi geliştirilmiştir.

Geliştirilen bu proje için popülerliğin yanı sıra güvenli olması, geniş bir platforma sahip olması, birçok geliştiricinin kullanması ve desteklenmesinden dolayı Java yazılım dili tercih edilmiştir. Java dili kodlama işlemini yapabilecek geliştirme ortamı olarak IntelliJ IDEA kullanılmıştır. Gerekli kütüphanelerin projeye eklenebilmesi için Maven altyapısı tercih edilmiştir. Mobil Test Otomasyon için oluşturulan akış şeması Şekil 4'de gösterilmiştir.



Şekil 4. Mobil test otomasyonunun akış şeması

Projede ilk olarak bir Maven projesi oluşturulmuştur. Bu projede kullanılması gereken araçlar, Selenium ve Appium kütüphaneleri eklenmiştir. Selenium ve Appium, ücretsiz, açık kaynak ve birçok yazılım dilimini desteklemesinden dolayı test otomasyonunda bu yazılım araçları kullanılmıştır. Mobil Test projesine başlarken Selenium ve Appium araçlarını deneme çalışmaları sırasında TDD yazılım süreci ile projeyi oluşturmaya başlanmıştır.

Appium bir sunucu olarak çalışır ve cihazdan aldığı bilgileri projeye iletirken projeden aldığı bilgi ve kodları cihaza iletmesi ile çalışmaktadır. Appium sunucu içerisinde test edilen cihazı bulması için yer alan Appium sürücüsüne (AppiumDriver) girilen telefon bilgileri ve Appium sunucu bilgileri eklenmesi ile Mobil Test Otomasyonu çalışır duruma gelmektedir.

Appium, mobil cihazdan istenilen uygulamaya ait element bilgilerini öğrenir. İşlem yapılmak istenilen elementin, öğrenilen element bilgileri arasından bulunması işlemi kodlama ile yapılmaktadır. Yazılan kod, elemente tıklama ve elementin yazısını öğrenmek gibi birçok işlem Appium üzerinden mobil cihaza iletilir. Bu haberleşme ile kodlanan testler koşulmaktadır. Ancak hali hazırda bulunan test senaryolarının yanı sıra yapılacak test senaryolarının anlaşılabilmesi için BDD yazılım süreci ile projenin oluşturulması gerekmektedir.

Mobil test otomasyonunda BDD mantığının uygulanabilmesi için Cucumber Kütüphanesi projeye eklenmiştir. Cucumber sayesinde proje içerisine yazılan senaryolara göre test koşturulmaktadır. Proje, bu yapıda çalışması için "Features", "Otomasyon" ve "TestCases" isimli üç ana klasör (paket) üzerine kurulmuştur. Features klasörü test senaryolarının yazıldığı dosyaları içermektedir. Otomasyon klasörü uygulamanın ve Cucumber'in ayarlarının özelleştirildiği kısımdır. TestCases klasörü ise senaryoların adımlarının yazılımsal kod şeklinde yazıldığı alandır. Bu yapıdaki projede ilk olarak projenin otomasyon ayarları olarak tabir edilen otomasyon klasörünün içerisine senaryoları içeren klasörün tanımlanması ve senaryolara göre yapılacak kodlamaları içerecek klasörün tanımlanması yapılmaktadır. Ayrıca cihazla ilgili gerekli ayarlamalar yapılmaktadır. Şekil 5'de örnek bir çalışmanın bilgileri yer almaktadır. Yapılan ayarlar test edilecek cihazın bilgileri, test için oluşturulan Appium sunucu portu ve test edilecek uygulamanın tanımlanmasıdır.

```
DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, deviceName);
capabilities.setCapability(MobileCapabilityType.UDID, udid);/// adb devices
capabilities.setCapability( capabilityName: "systemPort" ,systemPort);/// adb devices
capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, value: "android");
capabilities.setCapability(MobileCapabilityType.APP, value: "com.whatsapp");
capabilities.setCapability(AndroidMobileCapabilityType.APP_WAIT_ACTIVITY, value: "com.whatsapp*");
capabilities.setCapability(MobileCapabilityType.NO_RESET, value: true);
capabilities.setCapability(MobileCapabilityType.FULL_RESET, value: false);
capabilities.setCapability(AndroidMobileCapabilityType.AUTO_GRANT_PERMISSIONS, value: true);
ThreadLocalDriver.setTLDriver(new AndroidDriver<>(new URL(url), capabilities));
```

Şekil 5. Appium sunucusu için mobil cihaz bağlantısı için gerekli ayarlar

Features klasörü içerisinde senaryolar Cucumber dosyası olan "feature" uzantılı olarak oluşturulur. Oluşturulan bu Cucumber dosyasına senaryoların yazılması Cucumber aracı için oluşturulmuş Gherkin dili ile yapılmaktadır. Gherkin dili Türkçe dâhil birçok dili desteklemesi büyük bir avantaj sağlamaktadır. Gherkin, senaryo yazılımını neredeyse bir kalıp yokmuş gibi yazılmasını sağlayacak kadar özgür bırakmıştır ancak proje geliştirilirken sürdürülebilirlik olması için senaryo yazımında belirli bir kalıp oluşturulmuştur. Bu sayede ayrı ayrı uygulamaların veya aynı uygulamanın parça parça kısımlarının yazılımında ortak çalışma sağlayacak bir standart yapı oluşturulmuştur. Burada tıklama, yazı yazma ve kontrol etme gibi ana fonksiyonlar bir başlık altında toplanmıştır. Bu ana fonksiyonların alt bölümü de uygulamalara göre özelleştirme

yapabilmek için bir amaç yazılması şeklinde standart senaryo yazımı oluşturulmuştur. Şekil 6'da oluşturulan standart yapının bir senaryo örneği gösterilmiştir.

```
Feature: WhatsApp automation test
  Scenario:At WhatsApp Send a new message
    Given At "WhatsApp" Click "Home Page CHATS"
    Given At "WhatsApp" Click "Home Page New Chat"
    When At "WhatsApp" Select "1 Person" "New Message"
    When At "WhatsApp" SetText "New Message" Type "Automation Test Message"
    Given At "WhatsApp" Click "Send"
    Then At "WhatsApp" Check "New Message"
    Given At "WhatsApp" Click "Back"
```

Şekil 6. BDD mantığında Cucumber yazılım aracı ile Gherkin dilinde oluşturulan senaryo örneği

Senaryolar oluşturulduktan sonra senaryo içerisindeki adımların kodlanması yapılır. Bu kodlama için Java yazılım dilini kullanılmıştır. Cucumber özelliklerinden olan "Before" ve "After" parametreleri Java test dosyasında kullanılmıştır. Bu parametreler dosya çalışmaya başladığında yapılacak iş ve dosya çalışması bitince yapılması istenilen işlerin kodlandığı bölümdür. "Before" bölümünde senaryo adımlarını çalıştıracak kodların tekrar edenlerin ortak kullanımını sağlayacak nesne veya metotların bulunduğu proje kütüphanesi denebilecek Java kodları eklenmektedir. "After" bölümünde ise istenilen raporlama işlemleri ve ekstra yapılmak istenen işlemler tanımlanır. Şekil 7'de Senaryoların başlamadan önce gerekli olan kütüphane bilgisini ve donanım bilgisinin oluşturulduğu kısım gösterilmektedir.

```
@Before
public void InitializeTest(Scenario scenario) {
    appiumElements = new AppiumElements(ThreadLocalDriver.getTLDriver());
    chipSet = ThreadLocalDriver.getChipSet();
}

@After
public void TearDownTest(Scenario scenario) {...}
```

Şekil 7. Bir test sınıfının çalışmaya başlamasında ve bitiminde yapılması gereken ayarlar bölümü

Bu tanımlamalardan sonra senaryoda yazılan adımların sırası ile kodlanma kısmı yapılır. Standart oluşturulan senaryo yazımı mantığı ile metotlar oluşturulurken Cucumber'in senaryolar ile metotların haberleşmesini sağlayan yapısı gereği- metodun başına parametreleri tanımlayabilecek kod yapısı eklenmiştir. Senaryoda değişken

kısımlar hariç diğer yazılan kelimeler ile oluşan cümlelerden adımların kodlandığı bölümdeki cümleler ile eşleşen metotlar çalışmaktadır.

```
@Given("^At \"([^\"]*)\" Scroll \"([^\"]*)\"$")
public void at_Scroll(String applicationName, String purpose) {...}

@When("^At \"([^\"]*)\" GetSize \"([^\"]*)\"$")
public void at_GetSize(String applicationName, String purpose) {...}

@When("^At \"([^\"]*)\" SetText \"([^\"]*)\" Type \"([^\"]*)\"$")
public void at_SetText_Type(String applicationName, String purpose, String setMessages) {...}
```

Şekil 8. Cucumber yazılım aracı ile oluşturulan senaryo adımlarının yazılım kodu örneği

Şekil 8'de Cucumber ile yazılan senaryoların ve kod metotlarının haberleşme yapısını gösteren örnektir. Yazılım kodlarını içeren metotlar, senaryolardan gelen değişkenler, uygulama adı, senaryo adımının amacını ve gerekli olan diğer yapıları temsil etmektedir. Metot, ilk olarak Java'nın "Switch" parametresini kullanarak uygulama adlarına göre ayrılmasını sağlamaktadır. İkinci değişken, senaryoda yer alan amacı temsil eden yapılara göre ayrılmasını sağlamaktadır. Bu sayede uygulamada yapılmak istenilenler, senaryo adımlarına ve uygulama isimlerine göre gruplama yapılmış olacaktır.

```
@Given("^At \"([^\"]*)\" Click \"([^\"]*)\"$")
public void at_Click(String applicationName, String purpose) {
    switch (applicationName) {
        case "Twitter":
            switch (purpose) {

            }
            break;
        case "WhatsApp":
            switch (purpose) {
                case "Home Page CHATS":
                    appiumElements.ClickXPathText(WhatsAppID.TEXTVIEW, Text: "CHATS");
                    break;
                case "Home Page STATUS":
                    appiumElements.ClickXPathText(WhatsAppID.TEXTVIEW, Text: "STATUS");
                    break;
```

Şekil 9. Oluşturulan standart senaryo adımının yazılım kod içeriğinin örneği

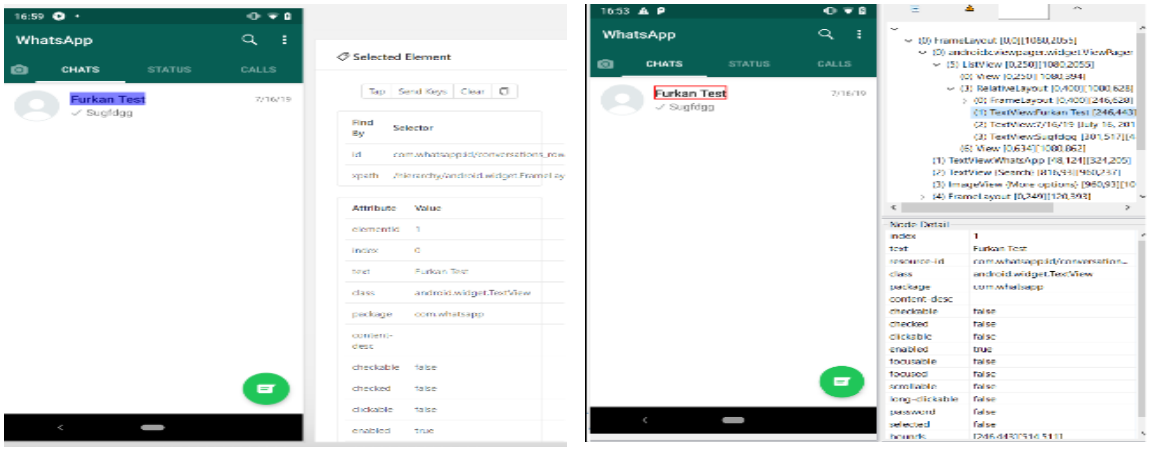
Şekil 9'da uygulamalara göre gruplanan ve uygulama içerisindeki amaca göre gruplandırmayı gösteren kod bloğu yer almaktadır. Senaryo adımlarının kodlandığı

metotlar içerisinde yapılacak kodlamayı kolaylaştırmak ve yazılan kodun tekrarlanmaması amacı ile "AppiumElements" adında bir nesne oluşturulmuştur. Bu nesne içerisinde uygulama elementine işlem yapabilmek için gerekli metotlar kategorize edilmiştir. Ayrıca bu nesne üzerinde otomasyonu çalıştırıp yönetecek olan bilgisayarın hızı yüksek düzeyde olduğu için uygulamadaki sayfa geçişlerinde veya uygulama elementindeki değişiklikler geç kalmaktadır ve otomasyon projesi görevi yapmadığı için hata vermektedir. Bu yüzden Selenium'un "WebDriverWait" nesnesi kullanılmıştır. Bu nesne istenilen elementi ekran üzerinde görülebilir olduğu ana kadar beklemektedir. Bu bekleme için WebDriverWait nesnesi maksimum bekleyeceği süreyi talep eder. Bu nesnenin diğer sistem uyutmasından farkı sabit bir süre olmamasıdır. Elementler işlemlere göre farklı bekleme sürelerine sahiptir. Bu yüzden sabit bir bekleme süresi yerine elementi beklemek zaman açısından büyük bir verimlilik sağlamaktadır.

```
public AppiumElements(AndroidDriver<AndroidElement> androidDriver) {
    this.androidDriver = androidDriver;
    this.wait = new WebDriverWait(androidDriver, timeOutInSeconds: 30);

    if (this.androidDriver.isDeviceLocked())
        this.androidDriver.unlockDevice();
}

private MobileElement waitForElementID(String selector) {
    return (MobileElement) wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(selector)));
}

private MobileElement waitForElementClassName(String selector) {
    return (MobileElement) wait.until(ExpectedConditions.visibilityOfElementLocated(By.className(selector)));
}
```

Şekil 10. Yazılım kodunun geliştirilmesi sırasında oluşturulan kütüphanenin örnek metotlar

Şekil 10'da metot isimleri ve gerekli olan element özelliği içeren metot yapıları gözükmektedir. Elementin bulunması için yazılan metotların elementin eşsiz özelliklerini bilmesi gerekmektedir. Uygulama geliştiricileri tarafından farklı metotlar da uygulama geliştirildiği için her element, farklı bir yapıya sahiptir. Bu elementlerin işlem yapabilmesi için öncelikle Android'in sağlamış olduğu SDK ile uygulama ara yüzünden elementin bilgilerine erişilmelidir. Şekil 11'de Appium veya Android SDK aracı ile uygulamalardaki element bilgisine ulaşılması gösterilmiştir.



a-) Appium b-) Andorid SDK

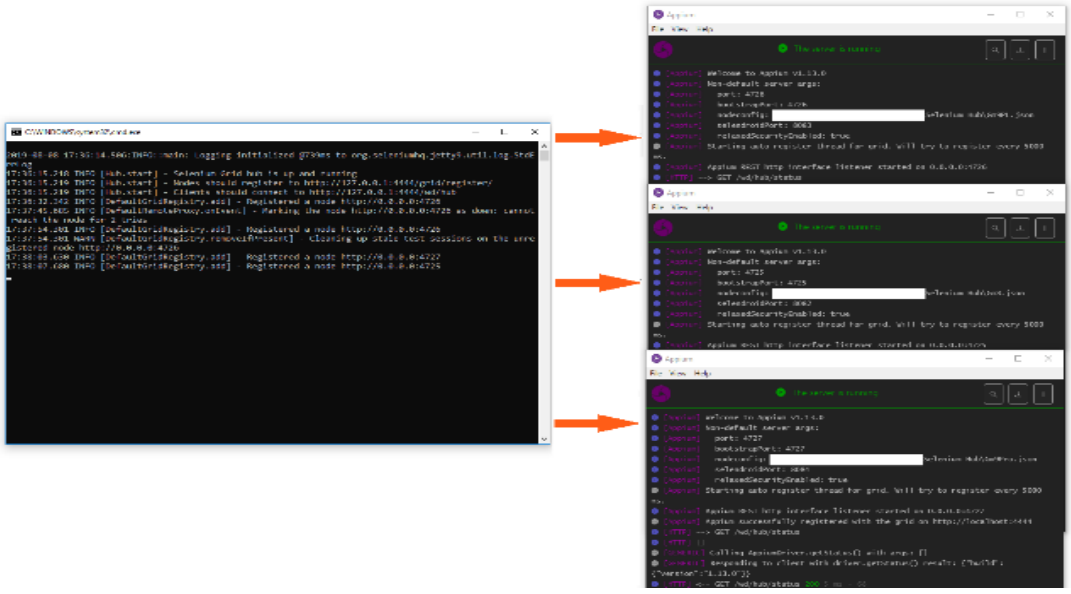
Şekil 11. Ara yüz elementlerinin bilgilerine ulaşım ekranı

Gherkin dili ile tanımlanan senaryoların Java ile kodlanma kısmında uygulama ara yüzünden alınan bilgilerin eklenmesi ile Mobil Otomasyon Test Projesi tamamlanır. Bu projede TestNG yazılım aracının görevi istenilen sayıda mobil cihaz ve istenilen testlerin paralel olarak koşulmasını sağlamaktır. İlk olarak TestNG Kütüphanesi projeye eklenir. Ardından TestNG için ".xml" uzantılı dosya içerisine hangi mobil cihazlar üzerinde çalışacağını ve mobil cihaz bilgilerin parametre olarak projeye aktarılması için gerekli olan bilgiler eklenir. TestNG, istenilen cihazların mobil otomasyon test projesinde gerekli olan bilgilerin eklenmesi ile tanımlanan bütün cihazların paralel şekilde çalışmasını sağlar. Şekil 12'de Paralel çalışmayı sağlayan TestNG yapısı için istenilen cihazlar eklenmesi ve bir cihaz için gerekli bilgilerin yer aldığı kod bloğu gösterilmiştir.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="OtomasyonCucumber" parallel="tests" data-provider-thread-count="1">

    <test name="Test-GM 9 Plus ">
        <parameter name="deviceName" value="GM9PL"/>
        <parameter name="udid" value="0123456789ABCDEF"/>
        <parameter name="platformVersion" value="9.0"/>
        <parameter name="url" value="http://0.0.0.0:4726/wd/hub"/>
        <parameter name="systemPort" value="8083"/>
        <parameter name="ChipSet" value="MediaTek"/>
        <parameter name="ApkFile" value="Facebook"/>
        <classes>
            <class name="Otomasyon.RunCucumberTest" />
        </classes>
    </test>

    <test name="Test-GM 9 Pro "...>
    <test name="Test-GM 9 Plus "...>
    <test name="Test-GM 9 "...>
    <test name="Test-GM 8 Go "...>
```

Şekil 12. Mobil cihazların paralel şekilde çalışmasını sağlayan "TestNG" için test cihazlarının bilgisi

Cihaz bilgilerinin eklenmesi ile mobil otomasyon test projesi gerekli cihazlarla birlikte tamamlanmıştır. Mobil otomasyon test projesi ile mobil cihaz arasındaki bağlantıyı kuracak olan Appium sunucusu her cihaz için farklı portta ve gerekli cihaz bilgileri ile çalıştırılır. Çoklu cihaz ve paralel test koşulması sırasında çalışacak olan Appium sunucuları birer düğüm (Node) halindedir. Bu düğümleri ortak alandan çalıştıracak yönetici ihtiyacını Hub karşılamaktadır. Bu hub ihtiyacı Selenium Grid ile oluşturulmaktadır. Açılmak istenen Appium sunucularına oluşturulmuş olan hub bilgisi eklenir. Sırası ile Selenium Grid ve Appium sunucuları çalıştırılır.



Şekil 13. Selenium Grid için bağlantı göbeği ile bağlantı kuracak Appium sunucuların paralel çalışmasının canlı örneği

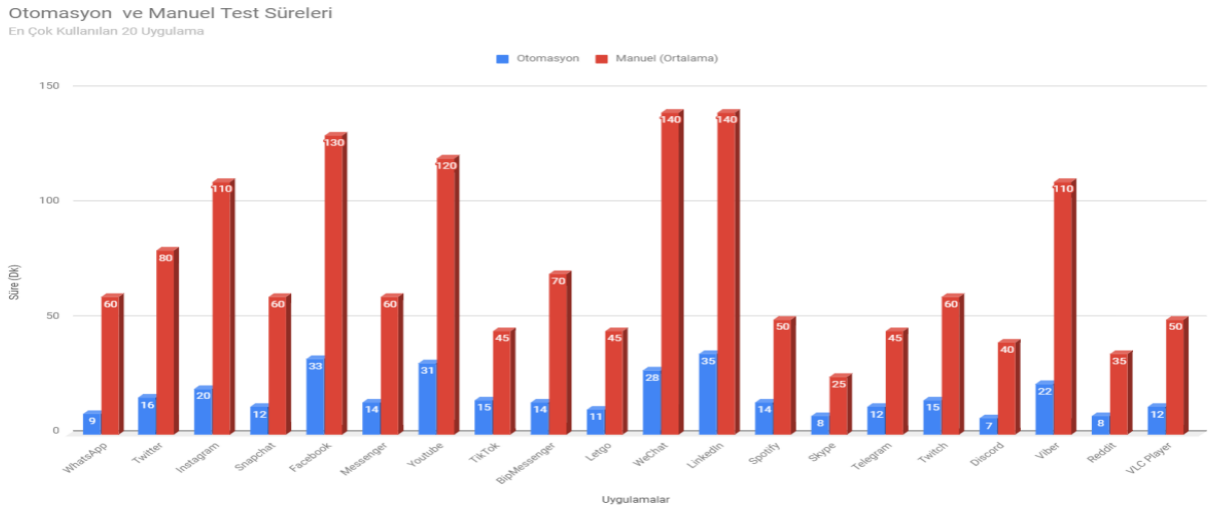
Şekil 13'de Şekil 3'de gösterilen yapının canlı ve çalışır haldeki yapısı gösterilmiştir. Test yapılması istenilen cihazların bağlantısı, Selenium Grid aracılığı ile mobil cihaza göre özelleştirilmiş Appium sunucuların açılması, mobil cihazların bilgilerinin TestNG dosyasına girilmesi ve istenilen testlerin senaryoları ve kod kısımlarının tamamlanması ile mobil otomasyon test projesi başarılı şekilde tamamlanmıştır.

4. General Mobile'da Otomasyon & Manuel Karşılaştırması

General Mobile, daha kaliteli ve verimli test yapabilmek adına test otomasyonu çalışmalarını başlatmıştır. Gelecekteki trendlere ve manuel yapılma zamanına bağlı olarak otomasyon çalışmalarına sosyal medya uygulamaları ile başlanmasına karar verilmiştir. Devamında Google Play Store Türkiye'de en çok indirilen uygulamalar için otomasyon çalışması yapılmış ve bu uygulamalardan en çok kullanılan yirmi tanesi seçilmiştir.

Söz konusu olan uygulamaların manuel olarak test edildiği zaman çok fazla zaman harcadığı görülmüştür. Otomasyon sonrası ise test süresi belirgin bir şekilde azalmıştır. Test için gereken sürenin ve insan gücü ihtiyacının azalması sonucu sadece regression testleri değil performans, loop ve stress testleri yapmak mümkün olmuştur.

Söz konusu uygulamalardan bazılarının otomasyon öncesi ve sonrasında gerektirdiği süreler Şekil 14'deki gibidir.



Şekil 14. Manuel ve otomasyon test süresi karşılaştırması

5. Sonuç

Gerçekleştirdiğimiz otomasyon çalışmaları sayesinde testlerin hızı, test edilen mobil cihazın özelliklerine ve test verilerine göre değişkenlik göstermekle birlikte yaklaşık 1/5 oranında artış sağlanmıştır. Test sürelerinin azalması, otomasyon senaryolarının detaylandırılmasına imkân vermiştir.

Şekil 14'deki veriler ele alındığında, bir Test Uzmanı, en popüler 20 uygulamanın manuel testini ortalama 24 saat 35 dakikada (1475 dakika) tamamlarken, otomasyon testi ile bu sürenin 5 saat 36 dakikaya (336 dakika) düştüğü görülmüştür.

Selenium Grid ile 20 uygulamayı paralel şekilde istenildiği kadar farklı cihazlarda çalışması sağlanmıştır. Farklı model cihazlar için her uygulamaya bir cihaz olacak şekilde ayarlanması ile otomasyon testi paralel şekilde çalışması sağlanacak ve bu sebeple otomasyon süresi en uzun süreye sahip olan uygulama kadar olacaktır. Bu çalışma ile de en popüler 20 uygulamanın manuel testi ortalama 24 saat 35 dakikada (1475 dakika) tamamlarken, otomasyon testi ortalama 35-40 dakikada tamamlanmıştır. Kaynak maliyeti büyük ölçüde azalmıştır.

Test otomasyonu ile üretilen mobil cihazların veya uygulamaların güvenirliği ve doğrulanabilirliği artmıştır.

Teşekkürler

Makalenin yayınlanmasında ve kabul süreçlerinde verdiği desteklerinden dolayı Karamanoğlu Mehmet Bey Üniversitesi idari kadrosuna, ayrıca UMTK-2019 programında emeği geçen herkese teşekkürü borç bilirim.

Kaynaklar

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

[2] Yazılım Test Nedir?, http://www.yazilimtest.com/yazilim-test-nedir-neden-gereklidir/ [20 Ağustos 2019]

[3] ISO/IEC 9126 Software Engineering - Product quality (2001-06)

[4] Güneş Kuday, 2014. "Android Mobil Uygulamalar için Yazılım Testi", 1 Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Düzce Üniversitesi, Düzce — Türkiye.

[5] Büşra Takgil, Resul Kara, 2016. "YAZILIM MÜHENDİSLİĞİ YÖNTEMLERİYLE YAZILIM TEST SÜRECİ", Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Bölümü, T.C. Haliç Üniversitesi, İstanbul — Türkiye.

[6] Yazılım Test ve Kalite Derneği, Türkiye Yazılım Kalite Raporu, 2019-2020, pp 08 / 2018-2019, pp 10,24 / 2011-2012, pp 09

[7] Meyer B. Object-oriented software construction. In: ISE Inc. Santa Barbara (California), 1997. pp. 57-61.

[8] Robert C. Martin. Design Principles and Design Patterns In:Ebook, 2000. pp. 14-16.

[9] Daniel L Asfaw. Benefits of Automated Testing Over Manual Testing. (2015) ISSN: 2349-7017

[10] Yazılım Test Araçları, https://fikirjeneratoru.com/yazilim-test-araclari-ve-bazilarinin-karsilastirilmasi/ [03 Eylül 2019]

[11] Veysel DEMİRER and Nurcan SAK, Programming Education And New Approaches Around The World And In TURKEY, 2016, pp. 525-530.

[12] What is Java?, Oracle, https://java.com/tr/download/faq/whatis_java.xml [11 Nisan 2019]

[13] What is Gherkin?, https://cucumber.io/docs/guides/overview/ [11 Nisan 2019]

[14] What is Selenium?, https://www.seleniumhq.org [11 Nisan 2019]

[15] Selenium Grid., https://www.seleniumhq.org/projects/[17 Haziran 2019]

[16] Introduction to Appium., http://appium.io/docs/en/about-appium/intro/?lang=tr [21 Mayıs 2019]

[17] What is Cucumber?, https://cucumber.io/docs/guides/overview/ [11 Nisan 2019]

[18] What is TestNG?, https://www.built.io/blog/what-is-testng [13 Ağustos 2019]

[19] What is Maven?, https://maven.apache.org/what-is-maven.html [08 Temmuz 2019]

[20] Melanie Diepenbeck, Ulrich Kühne, Mathias Soeken and Rolf Drechsler, Behaviour Driven Development for Tests and Verification, 2015, pp. 5-7.

[21] Kent Beck, Test-driven Development: By Example, 2003, pp.121-130.