T.C
DOKUZ EYLÜL ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
YÖNETİM BİLİŞİM SİSTEMLERİ ANABİLİM DALI
YÖNETİM BİLİŞİM SİSTEMLERİ PROGRAMI
YÜKSEK LİSANS TEZİ

YAZILIM TESTİ SÜREÇ YÖNETİMİ VE RAPORLAMA SİSTEMİNİN GELİŞTİRİLMESİ

Beytullah UZUN

Danışman
Dr. Öğr. Üyesi Kutan KORUYAN

İZMİR - 2019

YÜKSEK LİSANS
TEZ ONAY SAYFASI

Üniversite	: Dokuz Eylül Üniversitesi
Enstitü	: Sosyal Bilimler Enstitüsü
Adı ve Soyadı	: Beytullah UZUN
Öğrenci No	: 2016800829
Tez Başlığı	: Yazılım Testi Süreç Yönetimi ve Raporlama Sisteminin Geliştirilmesi
Savunma Tarihi	: 18/06/2019
Danışmanı	: Dr.Öğr.Üyesi Kutan KORUYAN

JÜRİ ÜYELERİ

Ünvanı, Adı, Soyadı	**Üniversitesi**	**İmza**
Dr.Öğr.Üyesi Kutan KORUYAN	-Dokuz Eylül Üniversitesi	.....
Prof.Dr.Kaan YARALIOĞLU	-Dokuz Eylül Üniversitesi	.....
Dr.Öğr.Üyesi Mehmet GÜVEN KOÇAK	- İzmir Katip Çelebi Üniversitesi	.....

Beytullah UZUN tarafından hazırlanmış ve sunulmuş olan bu tez savunmada başarılı bulunarak oy birliği(X) / oy çokluğu( ) ile kabul edilmiştir.

Prof. Dr. Metin ARIKAN
Müdür

YEMİN SAYFASI

Yüksek Lisans Tezi olarak sunduğum "Yazılım Testi Süreç Yönetimi ve Raporlama Sisteminin Geliştirilmesi" adlı çalışmanın, tarafımdan, akademik kurallara ve etik değerlere uygun olarak yazıldığını ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve bunu onurumla doğrularım.

...../...../2019

ÖZET

Yüksek Lisans Tezi

Yazılım Testi Süreç Yönetimi ve Raporlama Sisteminin Geliştirilmesi

Beytullah UZUN

Dokuz Eylül Üniversitesi
Sosyal Bilimler Enstitüsü
Yönetim Bilişim Sistemleri Anabilim Dalı
Yönetim Bilişim Sistemleri Programı

Bu çalışmada yazılım süreçlerinden test süreç yönetimi detaylıca incelenip test süreci, test yönetimi, test tipleri, yazılım hata yönetimi ve test raporlamaları hakkında temel bilgiler açıklanmıştır. Mevcut yazılım test yönetim uygulamaları ve iş zekâsı çözümleriyle sunulmakta olan detaylı test durum raporlamalarının kolaylıkla erişilebileceği ve yöneticilerin doğru ve zamanında karar vermelerine olanak sağlayacak raporların anlık ve güncel olarak sunulacağı bir raporlama sistemi geliştirilmesi amaçlanmıştır.

Bu test raporlama sistemi ile, testi gerçekleştiren ve testi takip eden kişilerin yapacağı sürekli raporlamaların doğru ve gerçek zamanlı olarak oluşturulması, raporlamalarda harcanan efor ve maliyetlerin düşürülmesi hedeflenmektedir. Yöneticiler ve yazılım projesinde bulunan paydaşların kullanacağı tüm raporlama tiplerine uygulama içerisinde yer verilmiştir.

Anahtar Kelime: Yazılım Testi, Yazılım Test Durum Raporlaması, Yazılım Test Süreçleri, Hata Durum Raporlaması.

ABSTRACT

Master's Thesis

Software Test Process Management and Development of Reporting System

Beytullah UZUN

Dokuz Eylül University

Graduate School Of Social Sciences

Department of Management Information Systems

Management Information Systems Program

In this study software test process, test management, test types, software defect management, test reporting and existing software test management applications's reporting dashboard had been researced in detail. It is aimed to develop a reporting system that will provide instant and updated reports, which detailed test case reporting with software test management applications and business intelligence solutions, to enable managers to make accurate and timely decisions making process.

With this test reporting system is aimed to create accurate and real-time continuous reporting by the people who conduct the test and who will follow the test, and to reduce the effort and costs spent in reporting.

Keywords: Software Test, Software Test Reporting, Software Test Process, Software Defect Reporting.

YAZILIM TESTİ SÜREÇ YÖNETİMİ VE RAPORLAMA SİSTEMİNİN GELİŞTİRİLMESİ

İÇİNDEKİLER

TEZ ONAY SAYFASI	ii
YEMİN SAYFASI	iii
ÖZET	iv
ABSTRACT	v
İÇİNDEKİLER	vi
KISALTMALAR	ix
TABLOLAR LİSTESİ	x
ŞEKİLLER LİSTESİ	xi
EKLER LİSTESİ	xiii
GİRİŞ	1

BİRİNCİ BÖLÜM
YAZILIM TESTİNE GENEL BAKIŞ VE TEST SÜREÇ YÖNETİMİ

1.1 YAZILIM TESTİ	3
1.2. YAZILIM TEST ÖNEMİ	4
1.2.1 Yazılım Hatası	5
1.3. YAZILIM TEST PRENSİPLERİ	9
1.4. YAZILIM TEST SÜREÇLERİ	11
1.4.1. Planlama ve Kontrol	11
1.4.2 Analiz ve Tasarım	14
1.4.3 Test Uygulaması ve Yürütme	14
1.4.4 Çıkış Kriterlerini Değerlendirme ve Raporlama	15
1.4.5 Test Kapanış Aşaması	16
1.5. YAZILIM TEST TİPLERİ	16
1.5.1 Fonksiyonel Test	17
1.5.2 Entegrasyon Testi	17

1.5.3 Birim Testi 18
1.5.4 Regresyon Testi 18
1.5.4.1 Regresyon Test Teknikleri 19
1.5.5 Performans Testi 20
1.5.6 Kullanıcı Kabul Testi (UAT) 21
1.6. YAZILIM HATA YÖNETİMİ 21
1.6.1 Yazılım Hataları Yaşam Döngüsü 22
1.6.2 Yazılım Hata Tipleri 26
1.6.2.1 Hata Şiddet Belirlenmesi 27
1.6.2.2 Hata Önceliğinin Belirlenmesi 28
1.7. YAZILIM HATA DURUM RAPORLAMASI 28
1.7.1 Hata Tipine Göre Dağılım 29
1.7.2 Hata Şiddetine Göre Dağılım Grafiği 29
1.7.3 Hata Önceliğine Göre Dağılım Grafiği 30
1.7.4 Hata Durumuna Göre Dağılım Grafiği 31
1.7.5 Test Senaryo İlerleme Grafiği 33
1.7.6 Hata ve Senaryo Sayısına Göre Analiz Raporları 34
1.8. MEVCUT YAZILIM TEST VE HATA YÖNETİM UYGULAMALARI 37
1.8.1 HP Application Lifecycle Management and HP Quality Center Enterprise 37
1.8.2 QA Complete 38
1.8.3 TestRail 38
1.8.4 TestLodge 38

İKİNCİ BÖLÜM

UYGULAMA

2.1 UYGULAMA DONANIM VE YAZILIM BİLEŞENLERİ 43
2.2 UYGULAMA EKRANLARI 45
2.2.1 Rapor Portal Giriş Ekranı 45
2.2.2 Rapor Portal Anasayfa 45
2.2.3 Proje Durum Raporu 46

2.2.4 Proje Seçimi Yapılır 47
2.2.5 Seçili Projeye Ait Oluşan Rapor 48
2.2.6 Proje Test Hata Grafikleri 50
2.2.7 Proje Kişi Bazlı Test Koşumu 53
2.2.8 Açık Durumda Bulunan Hatalar 55
2.2.9 Genel Test Raporu 55
2.2.10 Hata Durumları 62

SONUÇ 66
KAYNAKÇA 68
EKLER

KISALTMALAR

ALM	Application Lifecycle Management (Uygulama Yaşam Döngüsü Yönetimi)
API	Application programming interface (uygulama programlama arayüzü)
ISTQB	International Software Testing Qualifications Board
OCI	Oracle Call Interface
PHP	Hypertext Preprocessor (hiper metin önişlemcisi)
ss.	Sayfadan Sayfaya
vb.	ve benzeri
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language

TABLOLAR LİSTESİ

Tablo 1: Hata Tipleri Karakteristiği s. 27

ŞEKİLLER LİSTESİ

Şekil 1: ALM Defect Açma Ekranı s. 8
Şekil 2: Test Süreci s. 12
Şekil 3: Regresyon Test Tekniği s. 20
Şekil 4: Yazılım Hataları Yaşam Döngüsü s. 23
Şekil 5: Hata Tipine Göre Dağılım s. 30
Şekil 6: Hata Şiddetine Göre Dağılım s. 31
Şekil 7: Hata Durumuna Göre Dağılım s. 32
Şekil 8: Test Senaryo İlerleme Durumu s. 33
Şekil 9: HP ALM Arayüzü s. 37
Şekil 10: TestRail Arayüzü s. 39
Şekil 11: TestLodge Yeni Proje Yaratma Arayüzü s. 40
Şekil 12: TestLodge Yeni Test Planı Sayfası s. 40
Şekil 13: TestLodge yeni test gereksinimi sayfası s. 41
Şekil 14: TestLodge Test Durum Kontrol Sayfası s. 42
Şekil 15: Uygulama Akış Şeması s. 44
Şekil 16: Rapor Giriş Ekranı s. 45
Şekil 17: Rapor Portal Anasayfa s. 46
Şekil 18: Proje Durumları Menüsü s. 47
Şekil 19: Proje Seçim Ekranı s. 48
Şekil 20: Test Durum Raporu Ekranı 1 s. 49
Şekil 21: Test Durum Raporu Ekranı 2 s. 50
Şekil 22: Şiddet Bazlı Test Hata Raporu s. 51
Şekil 23: Statü ve Hata Tipine Göre Test Hatası s. 51
Şekil 24: Şiddet Bazlı Test Ortam Hata Raporu s. 52
Şekil 25: Statü ve Hata Tipine Göre Test Ortam Hatası s. 52
Şekil 26: Proje Kişi Bazlı Test Koşumu s. 53
Şekil 27: Açık Durumda Bulunan Hatalar s. 54
Şekil 28: Genel Test Rapor Menüsü s. 56
Şekil 29: Genel Test Raporu s. 57
Şekil 30: Ekip Günlük Test Koşumu s. 58

Şekil 31: Ekip Haftalık Test Koşumu s. 59
Şekil 32: Ekip Aylık Test Koşumu s. 60
Şekil 33: Test Hata Kabul Edilmeme Oranı s. 61
Şekil 34: Test Ortam Hata Kabul Edilmeme Oranı s. 61
Şekil 35: Test Verimliliği Grafiği s. 62
Şekil 36: Hata Durumları Menüsü s. 63
Şekil 37: Genel Test Hata Durum Raporu s. 64
Şekil 38: Statü Bazlı Test Hata Sayısı Dağılımı s. 65
Şekil 39: Aksiyon Alınması Gereken Hatalar s. 65

EKLER LİSTESİ

Ek 1: PHP OCI Örnek Kullanımı ek s. 1
Ek 2: Google Görselleştirme API Örnek Kullanım ek s. 3
Ek 3: Aylık Test Senaryo Sayısı ZingChart Örneği ek s. 8

GİRİŞ

Yazılım projelerinin bir parçası olan ve gün geçtikçe önemi artan yazılım testi, müşteriye kaliteli ürünler çıkarmayı hedefleyen, yazılım kalitesi hakkında bilgi sağlayan ve ürün piyasaya çıkmadan önce oluşabilecek riskler hakkında bilgi sunmayı hedefleyen bir yatırımdır.

Bir projede oluşabilecek tüm senaryoların test aşamasında gerçekleştirilmesi zaman ve maliyet açısında mümkün değildir. Yazılım testinde amaç, projenin planı içerisinde test sürecinin başarılı olarak sonlandırılması ve ürünün en az hata ile piyasaya çıkmasıdır.

Bu tez kapsamında yazılım testi hakkında genel bilgiler, test süreçleri, test tipleri, hata yönetimi ve yazılım hata durum raporlaması anlatılmıştır. Yazılım test süreçleri kapsamında testin başlangıcından bitişine kadar olan süreç detaylı olarak incelenmiştir. Yazılım test tipleri bölümünde fonksiyonel test, entegrasyon testi, birim testleri, regresyon test ve teknikleri, performans testi ve kabul testleri açıklanmıştır. Yazılımda bulunan hataların yaşam döngüleri ve hata tipleri hakkında bilgi sağlanıp hataların şiddet ve önceliklerinin nasıl belirleneceği belirtilmiştir.

Yazılım testlerinin takibi yazılım test ve hata yönetim uygulamaları ile yapılmaktadır. Mevcutta bulunan uygulamalar belirtilmiş ve özellikleri açıklanmıştır. Yazılım testlerinde hata durum ve test durum raporlamalarından hata tipi, hata şiddeti, hata önceliği, test senaryo ilerleme durumu ve test senaryolarına göre analiz raporlamaları hakkında bilgiler detaylıca açıklanmıştır. Uygulama bölümünde, yazılım test raporlama sisteminin teknik detayları, oluşturulacak raporların hangi şekilde proje paydaşlarına sunulacağı, raporların hangi görselliklerle oluşturulacağı gösterilmektedir. Uygulamada oluşturulan raporlar 3 ana başlıkta toplanmaktadır. Bunlar; proje bazında raporlama sistemi, test ekipleri bazında raporlama sistemi ve test ekipleri tarafından bulunan hataların analizinden oluşan raporlama sistemidir. Proje bazında raporlama sisteminde, test sürecindeki projelerde bulunan hata ve gerçekleştirilen test senaryolarına göre çıkarılan analizlerden elde edilen sayısal verilerin proje paydaşlarına sunulduğu raporlamalardır. Ayrıca projede test senaryosu gerçekleştiren testçilerin performanslarının izleneceği, aksiyon alınması gereken hataların görüntüleneceği raporlamalar bulunmaktadır. Projede oluşacak riskler ve

alınması gereken önlemler için proje ekibine bilgi sağlayacak raporlamalar üretilmektedir. Test ekiplerinin gerçekleştirdiği test senaryo sayısına göre yapılan analizlere göre performans değerlendirmesi, test verimliliği ve test kalitesi hakkında bilgi veren raporlamalar oluşturulmaktadır. Bu çalışmada ayrıca yazılım geliştirime süreci içerisinde bulunan ekiplerin koordinasyonunun sağlanması da hedeflenmiştir.

BİRİNCİ BÖLÜM
YAZILIM TESTİNE GENEL BAKIŞ VE TEST SÜREÇ YÖNETİMİ

1.1 YAZILIM TESTİ

Günümüz dünyasında hızla büyümekte olan bilişim sistemleri hayatımıza hızlıca yerleşmiş ve yaşamımızın her alanında yerini almış bulunmaktadır. Teknoloji firmaları bu ürünlerin pazarlanması ve piyasa içerisinde kalabilmesi için büyük bir rekabetin içerisinde bulunmaktadır. Firmaların piyasa içinde var olabilmek için sunduğu hizmetlerin müşteri isteklerini karşılayabilmesi ve müşterileri memnun etmesi gerekmektedir. Özellikle yazılım firmalarının ise bu başarının artması için ürün ve hizmetlerinin müşteriye sunulmadan önce test edilmesi önem taşımaktadır.

Kaliteli yazılımlar, kabul edilebilir seviyede hatasız, belirlenen beklentileri karşılayabilen yazılımlardır. Kaliteli yazılımlar için yazılım testi vazgeçilmez bir süreçtir. Yazılım testi, oluşturulacak ürünün piyasaya çıkmadan ya da müşteriye ulaşmadan önce oluşabilecek hataların görülmesine, ürünün yeterli olgunluğa ulaşıp ulaşamadığı bilgisine ve oluşabilecek riskleri önceden tespit edip müdahale edilmesine olanak sağlamaktadır.

Literatürde farklı yazılım test tanımları bulunmaktadır. Bunlardan bazıları;

- Yazılım testi bir yazılımın sonsuz sayıdaki çalışma alanından, sınırlı sayıda ve uygun şekilde seçilmiş senaryolar ile beklenen davranışlarını karşılamaya yönelik, dinamik olarak yapılan doğrulama faaliyetlerini kapsamaktadır (Bourque ve Fairley, 2014:82).
- Yazılım testi bir yazılımdaki gereksinimlerin karşılanıp karşılanmadığının gösterilmesi, bu testlere göre kalitesinin değerlendirilmesi ve gelişimine katkı sağlanması sürecidir (Friedman ve Voas, 1995:63).
- Genel olarak yazılım testi, gereksinimler doğrultusunda yazılımın kalitesinin ölçülmesi, yazılımdaki hata oranlarının gösterilmesini, yazılımdaki hataların en az seviyeye getirilmesini, yazılımdaki hataların erken bulunmasını ve bu sayede yazılım maliyetinin düşürülmesini, müşteri memnuniyetinin en üst seviyede tutulmasını hedefleyen süreçtir.

Yazılımda testi aşağıdaki maddelerin sağlanması için yapılmaktadır (Naik ve diğerleri, 2009:548-549):

- Yazılımın gereksinimleri karşılayabildiğini göstermek
- Yazılımın planlandığı gibi ilerlemesini sağlamak
- Ayrılan bütçe ile projenin ilerlemesini sağlamak
- Yazılımın kalitesini artırmak
- Yazılımın kabul edilebilir seviyede hatasız olmasını sağlamak
- Yazılımın sürdürülebilir olmasını sağlamak

1.2. YAZILIM TEST ÖNEMİ

İnsanoğlu her ne kadar ürettiği şeylerin hatasız ve her noktasının kusursuz olduğunu düşünerek en iyi ürünü ortaya çıkardığını düşünse bile, üretilen çıktı ve ürünlerin kontrol edilmesi ve minimum hata ile üretim yapılması gerekmektedir. Üretilen yazılımlar günlük hayatımızın hemen hemen her alanında yer alarak yaşamımızın ayrılmaz parçası olmuştur. Beklediğimiz gibi verim alamadığımız ve fonksiyonel olarak beklentimizi karşılamayan birçok yazılım prestij, zaman ve para açısından olumsuzluklara yol açabilir. Bu olumsuzluklardan kaçınmak ve en aza indirgemek için yazılım süreçlerinden olan test süreci önemli rol oynamaktadır.

Testin önemini geçmişte başarısızlıklarla sonuçlanmış olan birkaç yazılım projesini analiz ederek anlayabiliriz:

Denver Havaalanı Bagaj Otomasyon Sistemi: Denver uluslararası havaalanında, her yolcu için ortalama 30 dakika zaman kazancı olması için yazılımla yönetilen otomatik bagaj sistemi yapılmasına karar verilmiştir. Bu yazılım için 186 milyon dolar harcanarak 31 Ekim 1993'de sistemin açılması planlanmıştır. Ancak bagaj sisteminde ortaya çıkan yazılım hataları nedeniyle sistemin hizmete alınması gecikmeli olarak 28 Şubat 1995 tarihinde gerçekleşmiştir. Bu gecikmenin maliyeti günlük 1 milyon dolara yakın olmuştur ve gecikme nedeniyle yedek bir proje devreye sokulup ekstra maliyete sebep olmuştur. 2005 yılına kadar sürekli bakımlarla ayakta tutulan sistem, bakım maliyetinin çok fazla olmasından dolayı (aylık 1 milyon dolar) sistemin yerine başka sisteme geçilme kararı alınmıştır. Yazılımın başarısızlığı incelendiğinde gerçeğe yakın olmayan yetersiz simülasyonlar ile sistemin test

edilmesinden dolayı başarısız olduğu, zamanında bitirilemediği, sürdürülemediği ve maliyetli olduğu görülmüştür (Calleam Consulting, 2008:4-6).

Ariane 5: 4 Haziran 1996'da yapılan roket havalandıktan 40 saniye sonra 3700 metre yükseklikten düşmüş ve proje başarısızlıkla sonlanmıştır. Ariane 5 projesinin mühendisleri projenin başarısızlık sebeplerini araştırmaya başlamış ve daha sonra araştırmalar sonucu hatanın yazılımdaki hatadan kaynaklandığı anlaşılmıştır. Hatanın ondalıklı sayılardan kaynaklandığı, ondalıklı sayıyla ilgili kısımdaki kodda "olağandışı durum işleme" (exception handling) yapısının olmamasından kaynaklandığı belirtilmiştir. Başarısız proje sonucunda oluşturulan rapor içeriğinde, teknik olarak mümkün olduğu kadar gerçek sistemi simule edebilecek bir test ortamının hazırlanması, gerçekçi verilerle sistemin test edilmesi ve sistem testinin yapılması gerektiği belirtilmiştir. Ayrıca yapılacak olan testlerin kapsamının artırılmasına ve testten onay alınmadan hiçbir modülün devreye alınmamasına karar verilmiştir (Lions, 1996).

Mariner 1 Uzay Roketi: 1962 yılında yapılan uzay roketinde uçuşundan dakikalar sonrasında hata vermiş ve proje başarısızlıkla sonlanmıştır. Projenin başarısızlığı ve hatanın kaynağı araştırıldığında yazılımsal sorun olduğu tespit edilmiş ve bu hata uzay roketinin Atlantik Okyanusu'nda imha edilmesiyle son bulmuştur. Yazılımda yapılan bu hata 135 milyon dolarlık projenin başarısızlıkla sonuçlanmasına neden olmuştur (Johnson, 2012).

1.2.1 Yazılım Hatası

Yazılım testi belirlenen gereksinimlere göre hazırlanan test senaryoları ile gerçekleştirilir. Test sırasında bulunan belirlenen gereksinimleri karşılamayan ya da son kullanıcının beklemediği şekilde hata ile sonlanan senaryolara yazılım hatası (defect) denir.

Ron Patton'a göre yazılım hatası aşağıdaki 5 kuraldan en az birinden oluşmaktadır (Patton, 2001:44).

1. Yazılım, ürün özelliklerinde yapılması gerektiğini söylediği bir şey yapmaz.
2. Yazılım, ürün özelliklerinde yapmaması gereken bir şey yapar.

3. Yazılım, ürün özelliklerinde belirtilmeyen bir şey yapar.
4. Yazılım, ürün özelliklerinde belirtilmeyen ancak yapılması gereken bir şey yapmaz.
5. Yazılım yavaş çalışıyor veya anlaşılması ve kullanımı zor ise yazılım hatası olarak adlandırılabilir.

Yazılım hataları yazılımın kodunda, gereksinim dokümanlarında, kurulum dokümanlarında, tasarım dokümanlarında ya da yazılımın çalışma mantığında olabilir. Son kullanıcının istemediği sonuçları üreten her durum yazılımda 'yazılım hatası' olarak adlandırılabilir. Testçi tarafından bulunan hatalar çözüm ekiplerine detaylıca bildirir ve çözüm verildikten sonra gerekiyorsa tekrardan hata bulunan senaryosunun testini gerçekleştirir.

Bir yazılım hatasında olması gereken özellikler aşağıdaki gibidir;

- **Hata Numarası:** Bulunan tüm hataların eşsiz bir numarası bulunmalıdır.
- **Hata Özeti:** Bulunan hata hakkında detaylı bilgi verilmelidir. Verilen bu detay ile ilgili geliştirme ekipleri hata hakkında bilgi sahibi olmalı ve hatanın hangi senaryoda oluştuğunu görebilmelidir. Hata özetine ek olarak yapılan işlem hakkında log, ekran görüntüsü ya da video paylaşarak hatanın geliştiriciler tarafından hızlıca tespitine yardımcı olunabilir.
- **Hata Tarihi:** Hatanın açıldığı tarihi gösteren alan bulunmalıdır.
- **Raporlayan:** Hatanın kimin tarafından açıldığını gösteren alan bulunmalıdır.
- **Defect Durumu:** Defectin hangi süreçte olduğunu gösteren alan bulunmalıdır. Defectlerin statüleri; yeni, inceleniyor, kabul edilmedi, çözüm verildi, kapatıldı, ileri tarihe atıldı veya tekrar açıldı durumunda olabilir. Yeni bir defect açıldığında durumu yeni, geliştirmeci incelemeleri sonucunda ortada bir hata olmadığını yapılan testten kaynaklanan hata olduğunu belirtmek isterse statüsü kabul edilmedi, bulunan hataya bir çözüm verildi ise statüsü çözüm verildi, çözüm verilen bir hatanın tekrar testi sonucunda hatanın giderildiği görüldükten sonra statüsü kapatıldı, hatanın giderilmediği

görüldüğü durumda statüsü tekrar açıldı, hataya ileri bir tarihte ya da projede çözüm sağlanacaksa statüsü ileri tarihe atıldı olacaktır.

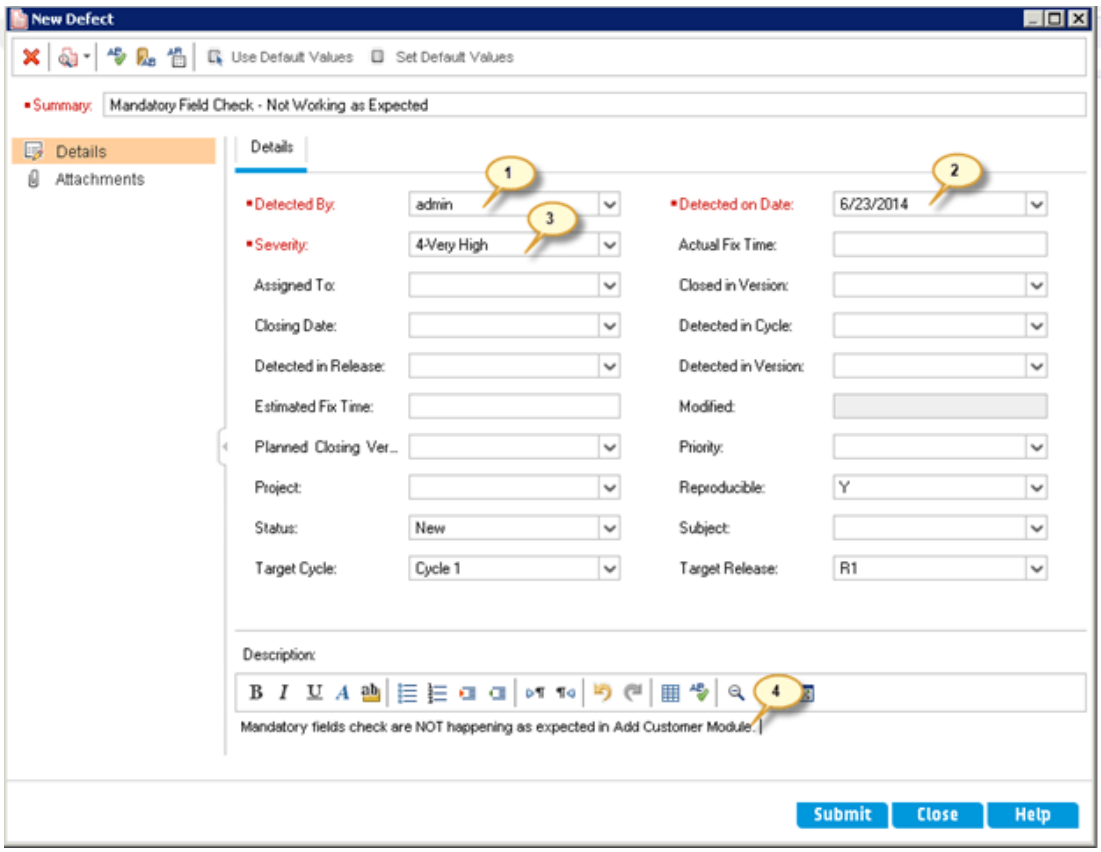
- **Çözüm Veren:** Hatanın giderilmesi için çözüm veren kişinin bilgilerinin bulunduğu alan.
- **Kapatılma Tarihi:** Hatanın kapatılma tarihini gösteren alandır.

- **Hata Şiddeti:** Hatanın şiddetinin belirlendiği alandır. Olması muhtemel hata türünün müşteriye, projeye, kuruluşlara ya da insanlara olan etkisinin önemini derecelendirmede kullanılır. ISTQB'ye göre hata şiddetleri aşağıdaki gibi sınıflandırılmıştır (Software Testing Fundamentals, 2018);
    - **Kritik Hata:** Kritik işlevselliği veya kritik verileri etkileyen hatalardır. Geçici bir çözümle ilerlenemeyen, projenin diğer safhalarını etkileyen ve projenin ilerlemesini etkileyen hatalardır. Örneğin; kurulum sırasında hata alınması ve kurulumun yapılamaması.
    - **Majör Hata:** Hatadan dolayı fonksiyonel gereksinimler karşılanamıyorsa ve alınan hatadan dolayı projede diğer fonksiyonlardan bazıları gerçekleştirilemiyorsa majör hata olarak sınıflandırılır.
    - **Küçük Hata:** Hatadan dolayı küçük işlevler veya kritik olmayan veriler etkileniyorsa küçük hata olarak derecelendirilir.
    - **Önemsiz Hata:** İşlevselliği veya verileri etkilemeyen hatalar önemsiz hata olarak sınıflandırılır. Örneğin yazım hataları ya da düzen uyumsuzlukları gibi problemlerdir.
- **Hata Önceliği:** Hatayı düzeltmenin önemini ve aciliyetini göstermek için kullanılır. ISQTB tarafından hata öncelikleri aşağıdaki gibi sınıflandırılmıştır (Software Testing Fundamentals, 2018):
    - **Acil:** Hatayla ilgili düzeltilmesinin öncelikli olarak yapılması gerektiğini gösterir.
    - **Yüksek:** Yeni iletilecek çözümlerin içinde yüksek öneme sahip hatalarında iletilmesi gerektiğini belirtir.

- **Orta:** Daha sonra iletilecek çözümlerle birlikte iletilebileceğini göstermektedir.
- **Düşük:** Hatanın çözülmesinin önemli olmadığını belirtmektedir.

Şekil 1'de örnek bir defect açma ekranı bulunmaktadır. Hata ile ilgili bulgular defect açma ekranında girildikten sonra ilgili kişiye yönlendirilmesi sağlanmış olur. Hata ile ilgili bilgilerin girişi yapıldıktan sonra hata ilk olarak "yeni" statüsünde bulunmaktadır.

Şekil 1: ALM Defect Açma Ekranı



Kaynak : Guru99, Defect Management Life Cycle in HP ALM (Quality Center) https://www.guru99.com/hp-alm-defect-management.html, (18.4.2019).

1.3. YAZILIM TEST PRENSİPLERİ

Yazılım testi son derece yaratıcı ve düşünsel olarak zor bir aşamadır. Bu zorlu aşamanın başarılı olarak gerçekleştirilmesi için ISTQB tarafından belirlenen yazılım test prensipleri doğrultusunda yazılım testi ilerletilmelidir. Yazılım test prensipleri 7 başlık halinde toplanmıştır (Chelliah ve Ross, 2012:17-19);

1. Test Hataları Gösterir

Yazılım testi sayesinde projede bulunan hataların ortaya çıkarılması hedeflenmektedir. Yazılım testi yapıldığında tüm hataların ortaya çıkarılmaması sistemde herhangi bir hata kalmadığı garantilenememektedir. Yazılım testi ile oluşabilecek hata sayılarında azalma ve sistemdeki hataların en az seviyeye indirgenmesi hedeflenmektedir.

2. Yazılımın Komple Test Edilmesi Mümkün Değildir

Yazılımda oluşabilecek tüm kombinasyonların hesaplanması ve uyarlanması testin gerçekleştirilmesi açısından zordur. Yazılım testlerinde tüm kombinasyon testlerinin gerçekleştirilmesi yerine, risk analizleri ve öncelikler verilerek, test aktivitesi gerçekleştirilmelidir.

3. Erken Test

Yazılımda karşılaşılacak hataların önceden tespit edilip erken çözüme ulaştırılması için yazılım testine olabildiğince erken başlanması gerekmektedir. Yazılım geliştirme süreçlerinden olan analiz ve tasarım aşamasında bulunan hataların çözüme ulaştırılması kolay ve daha maliyetsiz olacaktır.

4. Hatalar Belirli Yerlerde Yoğunlaşır

Yapılan araştırmalar, hataların büyük çoğunluğunun, küçük bir kısımda bulunduğunu göstermektedir. Bu duruma, Pareto prensibi denmektedir. Hataları %80'i ürünün %20'lik bir bölümünde bulunmaktadır. Bu kural dikkate alınarak yapılan testlerde, hataların büyük bir çoğunluğu daha kısa sürede bulunabilir.

5. Antibiyotik Direnci (Pesticide Paradox)

Sürekli aynı senaryolar ile yapılan testler sonucunda aynı hatalar yakalanacaktır, yeni oluşacak hataların aynı senaryolar ile yakalanması mümkün olmayacaktır. Pesticide Paradox prensibine göre test senaryoları sürekli olarak kontrol edilmeli ve yeni senaryolar eklenerek testler yapılmalıdır.

6. Test yaklaşımı projeye göre değişiklik gösterir

Belirlenecek olan test stratejileri, yapılacak olan testler projeden projeye farklılık gösterir. Bu nedenle, test yaklaşımı her proje için ayrı ele alınmalıdır ve farklı test stratejileri belirlenerek ilerlenmelidir. Örneğin banka yazılımları için belirlenecek olan test stratejileri ve yapılacak olan testler ile bir satış internet sitesi için test stratejileri ve yapılacak olan testler birbirinden tamamen farklıdır.

7. Hata bulunamadığı için ürünün hatasız olduğunu düşünmek

Testler başarıyla bitirildikten ve bulunan hatalar düzeltildikten sonra üründe hiç hata çıkmayacağı, ürünün müşteri ihtiyaçlarını karşılayacağı düşünülür. Ama ne yazık ki durum öyle değildir. Tüm testlerin yapılması mümkün olmadığı için, her zaman hata çıkma olasılığı vardır.

1.4. YAZILIM TEST SÜREÇLERİ

Geliştirilen yazılımların minimum seviyede hata içermesi ve gereksinimleri en iyi seviyede karşılayabilmesi için yazılım test eylemleri, yazılım geliştirme sürecinde en erken safhada yerini almalıdır. Yazılım dünyasındaki hata örnekleri incelendiğinde, çıkan hataların çoğunlukla analiz ve tasarım aşamasındaki hatalardan meydan geldiği görülmektedir. Analiz ve tasarım aşamasında bulunan hatalar projenin ilk safhalarında fark edilmediğinde, projenin sonraki fazlarında bu hataların düzeltilmesi için ekstra maliyet ve zaman harcanması gerekmektedir. Bu fazda bulunacak hatalar çok kritik olacağı için yazılımın komple değiştirilmesine ya da tekrardan projenin başlangıç safhası olan analiz ve tasarım aşamalarına dönülmesine sebep olacaktır. Yazılımlarda kaliteyi hedeflemek ve testin verimliliğini sağlamak için yazılım test süreçleri takip edilerek ilerlenmesi gerekmektedir.

Şekil 2'de gösterildiği gibi yazılım testlerinde temel süreçler aşağıdaki gibidir:

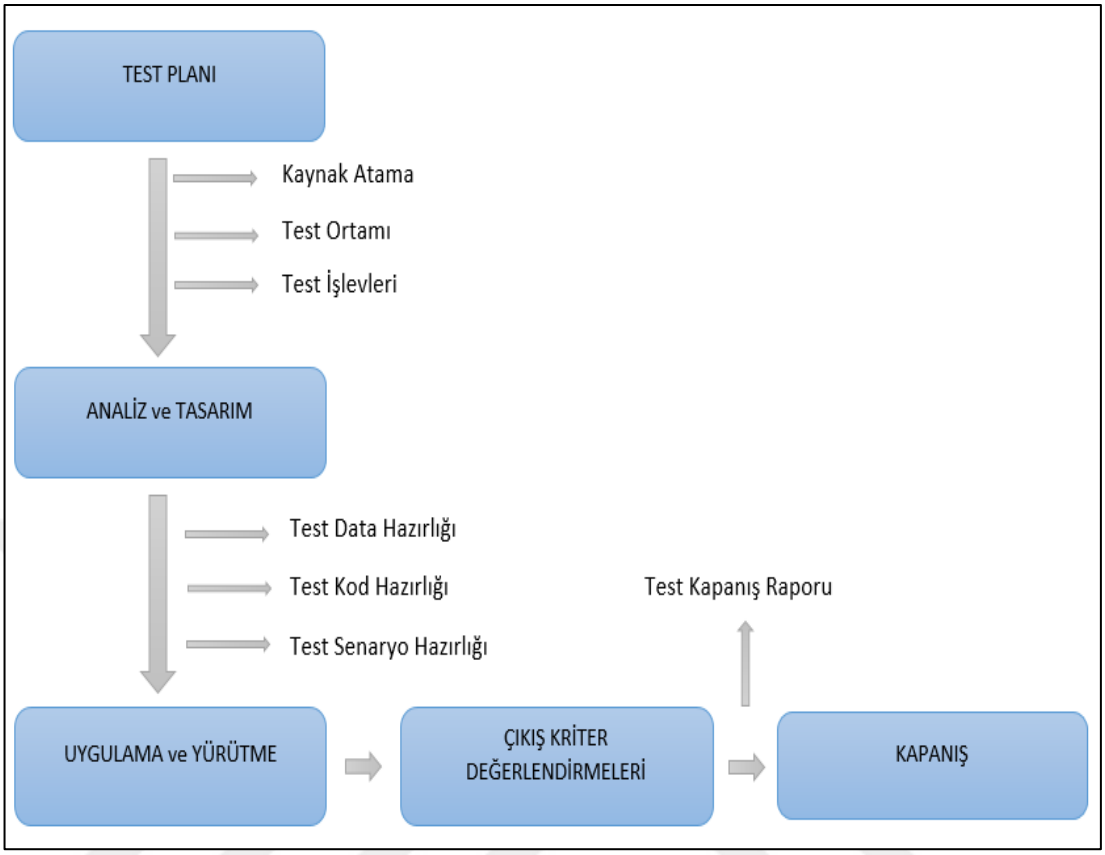
- Planlama ve Kontrol
- Analiz ve Tasarım
- Uygulama ve Yürütme
- Çıkış Kriterlerini Değerlendirme ve Raporlama
- Test Kapanış Faaliyetleri

1.4.1. Planlama ve Kontrol

Test planlaması sırasında, müşterilerin, paydaşların ve projenin amaçlarını ve hedeflerini, yapılacak testin kapsamını, testin stratejisini, hangi ortamlarda testin gerçekleştirileceği ve kaynakların belirlendiği fazdır. Planlama fazında ayrıca yazılım projeleri kapsamında geliştirilecek olan birim test planı, otomasyon test planı, veri hazırlık planı, yazılım test planı ve kullanıcı kabul test planı fazlarının test planı geliştirilebilir.

Test planlaması, aşağıdaki ana görevlerden oluşmaktadır (Graham ve diğerleri, 2008:24-25).

Şekil 2: Test Süreci



Kaynak : Yazar Tarafından Derlenmiştir.

- **Kapsam ve risk belirleme ve test hedefleri tanımlama:** Test etmek için hangi yazılımın, bileşenlerin ve sistemlerin kullanılacağı belirlenir. Ele alınması gereken iş, ürün, proje ve teknik riskler belirlenir. Yazılımın gereksinimleri karşılayacağını göstermek, sistemin amacına uygun olduğunu göstermek veya yazılımın niteliklerinin ölçmek için uygun test yöntemleri belirlenir.
- **Test yaklaşımı belirleme:** Yazılımın veya ürünün değişkenliğine bağlı olarak testinin nasıl yapılacağı, test yaparken kullanılacak tekniklerin belirlenmesi, yapılacak testin kapsamının belirlenmesi süreçleri gerçekleştirilir.
- **Test stratejisinin uygulanması:** Test stratejisinin doğruluğu ve tutarlılığı değerlendirilir ve testlerde bu stratejiye göre devam edildiğinin kontrolü yapılması gerekir.
- **Test kaynaklarının belirlenmesi:** Gerekli test kaynaklarının (testi yapacak insanların belirlenmesi, test yapılması için gerekli donanımların sağlanması)

belirlenmesidir. Gerekli olan yazılım ve donanımların kurulumlarının ve hazırlıklarının yapılması gerekir.

- **Test süreçlerinin planlanması:** Teste başlamadan önce tüm görev ve aktivitelerin planlanması gerekmektedir. Yapılan plan sayesinde test aktiviteleri takip edilebilir ve zamanında bitirilmesi için kontrol mekanizması haline gelmiş olacaktır. Aktiviteler önceden hazırlanan test plana göre uyarlanacağı için herhangi bir aktivitenin gecikmesi yada riskli bulunması durumunda önceden önlem alınabilecek ve test planının uyarlanmasına katkıda bulunulacaktır. Yürütme bölümünde oluşabilecek risklerden ve hatalardan dolayı yeniden planlamanın yapılması gerekebilir.
- **Çıkış kriterinin belirlenmesi:** Testlerin tamamlandığı bilgisinin verilmesi için belirlenen kriterlerdir. Testlerin tamamlanıp müşteriye ya da üretim ortamına çıkarılma kararı verilmesi için testlerin belirli bir seviyede tamamlanması gerekmektedir. Bu seviyenin belirlenmesinde testlerin tamamlanma yüzdesi, açıkta olan kritik hata sayıları, hangi görev ve kontrollerin tamamlanması gerektiği gibi etkenler belirlenmelidir.

Yapılan test planlama aktiviteleri her ne kadar başarılı ve hatasız olarak yapılmış olsa bile yapılan plan ve aktiviteler sürekli olarak kontrol edilmelidir. Yapılan planlama ve gerçek ilerleme karşılaştırılmalı ve planda herhangi bir değişiklik veya sapma olması durumunda proje yöneticisi ve müşteriye mevcut test durumu hakkında bilgi verilmelidir. Yapılan test durumları ve kontroller sayesinde projenin amaçlarını yerine getirmek için gerekli olan yerlerde önlem alınması kolaylaşacaktır.

Test kontrolleri aşağıdaki maddelere göre yapılmalıdır (Graham ve diğerleri, 2008:24):

- Test bulgularının analizi ve ölçümü

Testler sırasında bulunan hataların sayısı, türü ve öneminin bilinmesi. Tüm test senaryolarından kaçının tamamlandığı ve kaçının başarısız olduğunun izlenmesi gerekir.

- Testler hakkında bilgi verme

Proje yöneticisi, müşteri ve diğer paydaşlara proje durumu hakkında bilinçli kararlar verebilmelerine yardımcı olacak düzenli raporlar verilmelidir.

1.4.2 Analiz ve Tasarım

Test analizi ve tasarımı aşağıdaki temel adımlardan oluşmaktadır:

- Test temelini gözden geçirme; ürünün risk analizi, gereksinimleri, tasarım özellikleri ve arayüzlerinin test edilebilirliği açısından değerlendirilmesidir.
- Ürünün teknik özellikleri incelenir ve belirsizlikler tespit edilir.
- Test maddelerinin özelliklerine, davranışlarına, analizlerine göre test koşullarının tanımlanması ve önceliklendirilmesi.
- Test senaryolarının şablonu oluşturulması ve önceliklendirilmesi.
- Risk taşıyan test senaryolarının belirlenmesi ve önceliklendirilmesi.
- Test ortam kurulumunun tasarlanması
- Testte kullanılacak yardımcı yazılımların tanımlanması

1.4.3 Test Uygulaması ve Yürütme

Test uygulaması ve yürütme belirtilen senaryoları manuel olarak veya otomatik bir test aracı kullanarak çalıştırmayı içerir. Ürünün ele alınıp test çalışmalarının yapıldığı temel test sürecidir (Sharma,2018).

Test uygulaması ve yürütme aşamasında aşağıdaki temel adımlar yer almaktadır:

- Test senaryolarının geliştirilmesi ve önceliklendirilmesi
- Testte kullanılacak olan verilerin oluşturulması
- Otomasyonu yapılacak olan test senaryolarının oluşturulması
- Otomasyon test kodlarının oluşturulması
- Test prosedürlerini yazma ve önceliklendirme
- Test ortam kurulumu ve test ortamının doğru şekilde kurulduğunu doğrulama
- Önceliklendirilmeye uyulacak şekilde test prosedürlerinin çalıştırılması
- Test senaryolarını test ortamında gerçekleştirme

- Testler sırasında yapılan işlemlerin log ve ekran görüntülerinin alınması
- Gerçekleşen sonuçları beklenen sonuçlarla karşılaştırma
- Gerçekleşen ve beklenen sonuçlar arasında farklılıklar olduğunda, hatanın sebebi için detaylı log analizi ve hatanın sebebinin ne olduğunu analiz etme
- Testler sırasında bulunan hataların düzelttikten sonra tekrar aynı senaryonun gerçekleştirilmesi ve hatanın giderildiğinin görülmesi.
- Hatalara istinaden düzeltme yapıldıktan sonra etkilenecek modüllerde hata kontrolü yapılması

1.4.4 Çıkış Kriterlerini Değerlendirme ve Raporlama

Çıkış kriterleri, test planlamada aşamasında belirlenen çıkış kriterlerinin değerlendirildiği ve kontrol edildiği aşamadır. Yeterli testin yapılabildiğini gözlemleyebilmek için çıkış kriter değerlendirmesi bütün test seviyelerinde gerçekleştirilmelidir. Çıkış kriterlerinin değerlendirilmesinde gerçekleştirilen test senaryo sayısı, bulunan hata sayısı, hataların şiddetleri ve çözümlenmemiş olan açık hataların dereceleri gibi parametreler kullanılır.

Çıkış kriterlerinin değerlendirilmesi ve raporlama aşaması aşağıdaki temel adımlardan oluşmaktadır (Graham ve diğerleri, 2008:28):

- Test uyarlama aşamasında gerçekleştirilen test kayıtları çıkış kriterlerine göre değerlendirilir. Hangi testlerin yapıldığı, hangi hataların kaldırıldığı ve son durumda testlerin hangi durumda olduğu görülür.
- Belirtilen çıkış kriterlerine göre daha fazla teste gerek olup olmadığı kararı alınır. Planlanan testlerde beklenen tamamlanma oranına ulaşılamadığı durumda daha fazla test yapılması kararı alınabilir.
- Çıkış kriterlerinde tüm kritik test senaryolarının tamamlanması, kritik hataların düzeltilmiş olması ve testlerinin tekrar yapılmış olması gerekmektedir.

- Yazılımda test sürecinin sonuna gelindiğinde paydaşlar ve ürün sahiplerinin projenin durumu hakkında bilgilendirilmesi için test özet raporu hazırlanır. Test özet raporunda yazılım hakkında test sürecinde karşılaşılan tüm bulguların özet olarak sunulması gerekmektedir. Test raporlaması ile projede risk görülen maddeler ortaya koyulur ve bu riskler karşısında verilecek kararların verilmesine yardımcı olur.

1.4.5 Test Kapanış Aşaması

Test kapanış aşaması aşağıdaki maddelerden oluşmaktadır:

- Plan aşamasında belirlenen tüm testlerin tamamlanmasının kontrolü gerçekleştirilir ve bilinen tüm hataların çözüme ulaştırıldığı kontrolü yapılır.
- Test aktiviteleri tamamlandıktan sonra testler sırasında kullanılan komutlar, test ortamları ve test verileri daha sonra tekrar kullanılacak şekilde arşivlenir.
- Yazılımın başarılı veya başarısız olduğunu gösteren raporlar hazırlanır.
- Testler sırasında karşılaşılan sorunları ve testlerin nasıl ilerlediğini değerlendirecek şekilde test değerlendirme dokumanı hazırlanır. Hazırlanan dokuman bu projeyle ilgili ya da benzer projeyle ilgili test süreçlerine yol gösterici olacaktır. Hazırlanan değerlendirme dokümanında çıkan kritik hatalar hakkında bilgi verilir, projenin test efor takvimine göre ilerlenebildiği, test sürecinde iyileştirme gereksinimleri gibi maddeler yer almalıdır (Try QA, 2018).

1.5. YAZILIM TEST TİPLERİ

Yazılımın özelliğine göre farklı test teknikleri kullanılabilmektedir. Genel olarak yazılım test tipleri aşağıdaki gibidir (Watkins ve Mills, 2010:35-43).

1.5.1 Fonksiyonel Test

Sistemin işlevselliğinin, gereksinimlerinin doğru şekilde çalışıp çalışmadığının görüldüğü test tipidir. Sistemin alacağı girdilere göre beklenen çıktıların kontrol edildiği test durumlarıdır. Fonksiyonel testlerde sistemin teknik altyapısı ile ilgilenilmez, sadece istenilen işlevin gerçekleşip gerçekleşmediği kontrolü yapılır.

Fonksiyonel teste bir web sayfasında kullanıcı kayıt olma bölümünde telefon numarası giriş alanının testi örnek olarak verilebilir. Test senaryomuzda telefon numarası alanına 10 karakterlik sayısal değer girişi yapılması isteniyor ve sayısal değerler dışında ya da fazla karakter girişinde sistemin hata vermesini bekleniyor olsun. Bu alana 10 karakterli sayısal değer girişi yapıldığında herhangi bir hata vermediğinin görülmesi, 10 karakterin altında yada üstünde giriş yapıldığında anlamlı hata verdiğinin görülmesi,10 karakterli sayısal değerler dışında hata veriyor olmasının görülmesi fonksiyonel test örneğimizdir.

Fonksiyonel testlerin amacı aşağıdaki gibidir;

- Sistemin gereksinimleri karşılayıp karşılamadığının belirlenmesi ve sistemin hangi oranda doğru çalıştığının görülmesi
- Sistemin kullanıcı/müşteri tarafından kolaylıkla kullanılabildiğinin görülmesi
- Sistemin hatasız ve güvenilir olarak çalışmasının sağlanması

1.5.2 Entegrasyon Testi

Fonksiyonel testlere başlanılmadan önce sistemin test edilebilirliğini ve diğer modüllerle uyumlu şekilde çalıştığını görmek için yapılan testtir. Oluşturulan yazılım ürünleri birçok modülden oluşabilir ve bu modüller bir araya geldiğinde uyumlu çalışıp çalışmadığı, modüller arası etkileşimden kaynaklı çıkacak hataların analizlerinin yapılması ve çözümlenmesinin yapıldığı test adımıdır. Bu test tipinde birbiriyle bağımlılığı bulunan veri alışverişi yapan modüllerden test senaryoları oluşturulur ve bu senaryolar doğrultusunda entegrasyonlarının testi gerçekleştirilir.

1.5.3 Birim Testi

Birim testi yazılım geliştiriciler tarafından yapılan testtir. Birim testin amacı yazılımın içinde bulunan küçük parçaların kendi içinde test edilmesidir (Software Testing Fundamentals, 2018). Örneğin bir kod içerisinde yazılmış birden çok fonksiyon ve metotlar bulunsun, bu fonksiyon ve metotların verilen girdilere göre doğru çalıştığını gözlemlemektir.

1.5.4 Regresyon Testi

Fonksiyonel testler sırasında bulunan hatalara istinaden düzeltme paketleri alınır ve bu paketler ile hata ile karşılaşılan senaryolara ait testler tekrarlanır (retest). Tekrar yapılan bu test işleminde sadece hata ile karşılaşılan senaryo ile ilgilenilir, burada gözden kaçırılan nokta ise verilen yeni paketin diğer test senaryolarına etkileridir. Verilen paket daha önce başarılı olarak sonuçlandırdığımız senaryolardan bazılarını etkilemiş olabilir veya hâlihazırda çalışan senaryoların bozulmasına sebep olabilir. Bu tarz hataların gözden kaçmaması için fonksiyonel testler sonrası yapılan testlere regresyon testi denir (Graham ve Diğerleri, 2008:59). Regresyon testlerinde etkilenecek modül ve sistemlerin senaryoların testleri gerçekleştirilir. Ayrıca ana fonksiyonel senaryolar hata düzeltmeleri sonrasında etkilenmediğinin görülmesi için regresyon testlerine eklenmelidir.

Görüldüğü üzere regresyon testleri her projenin sonunda yer almalı ve genellikle aynı senaryoların testleri gerçekleştirilmektedir. Başarılı regresyon testleri için regresyon test kümesi sürekli olarak genişletilmeli ve hata çıkan modüllere daha çok ağırlık verilmelidir.

Otomasyon testleri sürekli aynı senaryoların gerçekleştirilmesini sağlamakta ve sistemde herhangi bir değişiklik ya da hata olduğunda bu hatalar otomasyon testleri sayesinde görülebilmektedir. Regresyon test kümeleri otomasyon testleri için en ideal test senaryolarıdır.

Regresyon testi aşağıdaki maddeler sonrasında yapılmalıdır;

- Gereksinimlerde değişiklik olduğunda ve bu değişikliklere göre kodda yapılan değişiklik durumlarında

- Yazılıma yeni özellikler eklendiğinde
- Testler sırasında bulunan hatalar sonrası verilen çözümler sonrasında
- Performans hatalarına yönelik çözümler sonrasında

Chen L. Ve diğerlerine göre Regresyon testleri için test senaryoları arasında bir bağımlılık analizi çıkarılmalı ve bu bağımlılığa göre senaryolar önceliklendirilmelidir. Senaryoların önceliklendirilmesi bir grafik üzerinde yapılır ve hataların giderilmesi sonrasında en çok etkilenen senaryolardan en az etkilenen senaryolara göre ağırlıkları belirlenmelidir (Chen ve diğerleri, 2010:24-25).

1.5.4.1 Regresyon Test Teknikleri

Regresyon testi Şekil 3 de belirtildiği gibi aşağıdaki tekniklere göre yapılmalıdır (Guru99, 2018).

- Tüm Test Senaryolarının Tekrarlanması

Bu yönteme göre tüm test senaryoları tekrardan kontrol edilmeli ve gerçekleştirilmelidir. Bu yöntem regresyon test teknikleri içerisinde en maliyetli olan yöntemdir. Yazılımda fonksiyonel testler sırasında yapılan değişiklikler büyük değişiklikler değilse bu yöntemin kullanılması zaman ve maliyet açısından kullanışsız olacaktır.

- Regresyon Test Seçimi ile İlerleme

Bu yönteme göre tüm testlerin tekrarlanması yerine bazı test senaryolarının seçimi ile ilerlenir. Bu seçimin yapılması sırasında eski test senaryolarından ve uygulanabilirliği kolay olan test senaryo seçimine dikkat edilmelidir.

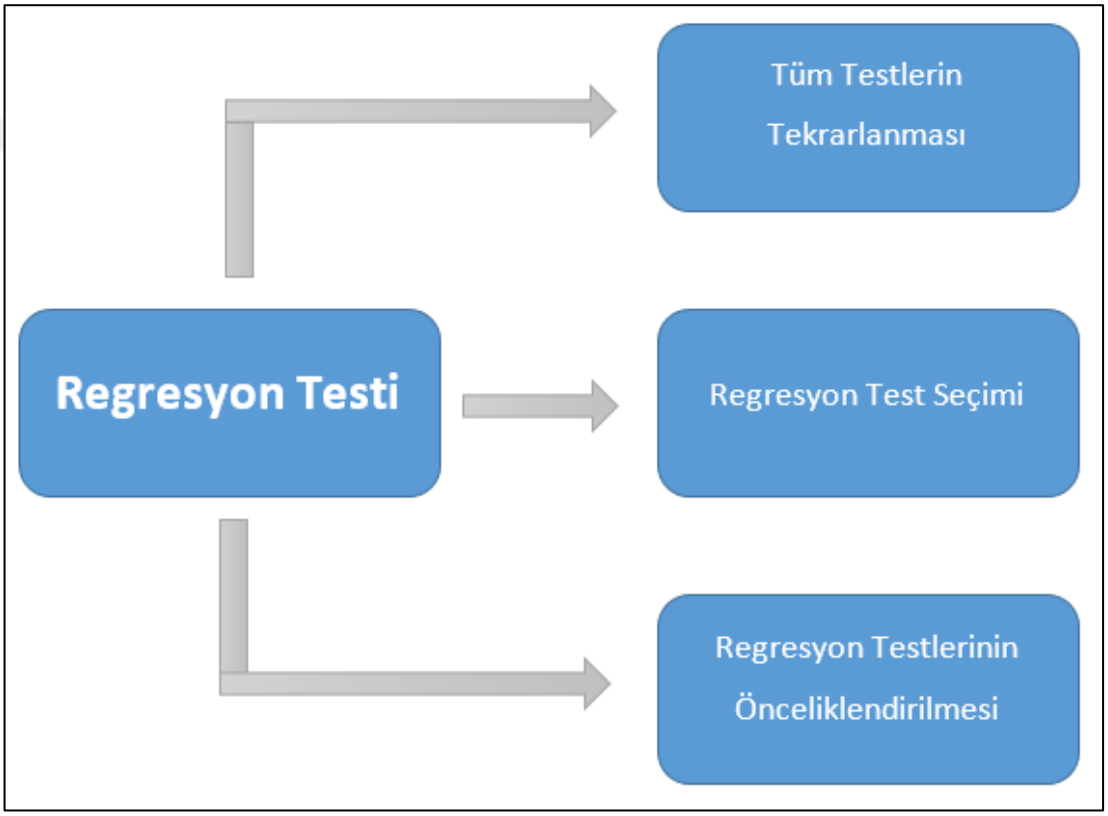
- Test Senaryoları Önceliklendirilmesi

Fonksiyonel gereksinimler göz önünde bulundurularak en çok kullanılacak ana senaryoların seçimi ile test senaryolarının önceliklendirilmesiyle yapılır. Bu yöntemle yapılan testlerde zaman kaybı önlenerek ilerlenmiş olacaktır.

Rajal ve Sharma'ya göre test sürecinin son zamanlarında bulunan hata düzeltmelerinin etkisi diğer hatalara göre daha fazla olmakta ve bu test senaryolarında daha dikkatli olunmalıdır. Regresyon test senaryo seçiminde aşağıdaki kriterlerin önemi belirtilmektedir (Rajal ve Sharma, 2015:7-13):

- Senaryo seçimlerinde daha çok hata içeren senaryoların seçimi yapılmalı
- Kullanıcılar tarafından en çok kullanılan senaryolar seçilmeli
- En çok değişiklik içeren senaryolar seçilmeli
- Entegrasyon test senaryoları seçilmeli
- Uygulaması kolay senaryolar seçilmeli
- Hataya kolayca düşürülecek senaryolar seçilmeli

Şekil 3: Regresyon Test Tekniği



Kaynak: Jaspreet Singh Rajal ve Shivani Sharma, "A Review on Various Techniques for Regression Testing and Test Case Prioritization", International Journal of Computer Applications (0975 – 8887) Cilt:116,Sayı: 16, Nisan 2015, ss. 10.

1.5.5 Performans Testi

Performans testi çıkış kriterlerinde değerlendirilmesi gereken test tipidir. Tüm fonksiyonel testler istenilen başarıda tamamlanmış olsa bile performans testinde başarısız olursa çıkış kriterinden geçememiş olacaktır.

Performans testi belirlenen kriterlere göre sistemin performans ölçümünün yapılmasıdır. Örneğin bir internet sitesi, siteye kaydolan müşterilere ilk aktivasyon maili atıyor olsun ve aktivasyonlar belirli dönemlerde tepe noktaya ulaşıyor olsun. Performans gereksinimi olarak ise sistemin saniyede 50 kişiye mail gönderebiliyor olması gereksin. Performans testi sayesinde kriterin sağlanıp sağlanmadığı testi gerçekleştirilir. Performans testleri için hazırlanan yazılımlar ile yazılım test performans testleri gerçekleştirilir. Aynı zamanda performans testleri için test uygulayıcıları tarafından kod geliştirilebilir.

1.5.6 Kullanıcı Kabul Testi (UAT)

Üretilen ürün ya da programın son kullanıcılar ya da müşteriler tarafından test edilme sürecidir. Test sürecinde son olarak yapılan kullanıcı kabul testidir ve bu süreçte bulunamayan hataların telafisi çok zor ve maliyeti çok yüksel olacaktır.

Kullanıcı kabul testleri öncesinde yazılımcılar tarafından birim testlerin tamamlanmış olması gerekir ve testçiler tarafından tüm test senaryolarının tamamlanmış olması gerekmektedir.

Bu aşamada projenin son hali görülür ve proje başındaki gereksinim ve kriterler arasındaki uyumun gözlenmesine yardımcı olunur. Kullanıcılara yazılımla etkileşimde bulunma, özelliklerin gözden geçirilmesi, iletişim kurulması imkanının sağlandığı test aşamasıdır (Usersnap, 2019).

1.6. YAZILIM HATA YÖNETİMİ

Yazılım hata yönetimi, yazılım geliştirme sürecinin her aşamasında oluşabilecek hataların bulunması ve çözüme ulaştırılması sürecidir. Test sürecinde hatalar bulunarak daha kaliteli ürünler ortaya çıkarmak hedeflenmektedir. Projelerde bulunan hataların doğru bir şekilde yönetimi yapılırsa çözüme ulaştırılmaları daha kolay olacaktır. Hataların yönlendirilmesi ve yönetilmesi için piyasada çok sayıda bulunan yazılım hata yönetim araçları kullanılabilir. Test ekipleri çıkan hataları adresledikten sonra çözümüyle ilgili süreçleri takip etmelidir, bu takibi ise sürekli raporlamalar ile gerçekleştirmelidir. Düzenli raporlamalar sayesinde çözüm süresi

geciken hataları ve kritik hataların projeyi ne kadar etkilediğini görebilmektedir. Raporlama ile test süreci takip edilebilir ve proje ekibine sayısal olarak zengin bir raporlama sunabilir. Yazılım hata yönetimi ve raporlamaları aşağıdaki başlıklar takip edilerek yapılabilir.

1.6.1 Yazılım Hataları Yaşam Döngüsü

Yazılımda oluşabilecek hatanın bulunmasından çözümlenmesine kadar geçen sürece hata yaşam döngüsü denilir. Yazılım hata yaşam döngüsü genellikle Şekil 4'te bulunan adımlardan oluşmaktadır, bu adımlar organizasyonlara ve hata takibi için kullanılacak olan yazılımlara göre farklılık gösterebilmektedir.

Yeni (New)

Bulunan hatanın sisteme ilk girildiği durum 'yeni' olarak adlandırılır. Bu aşamada bulunan hatayla ilgili tüm detaylara yer verilir. Hatayla ilgili detaylar, ekran görüntüleri, hatanın oluştuğu ortamın özellikleri ve hatanın oluşturulmasında kullanılan data tipleri gibi özelliklerin belirtilmesi gerekmektedir.

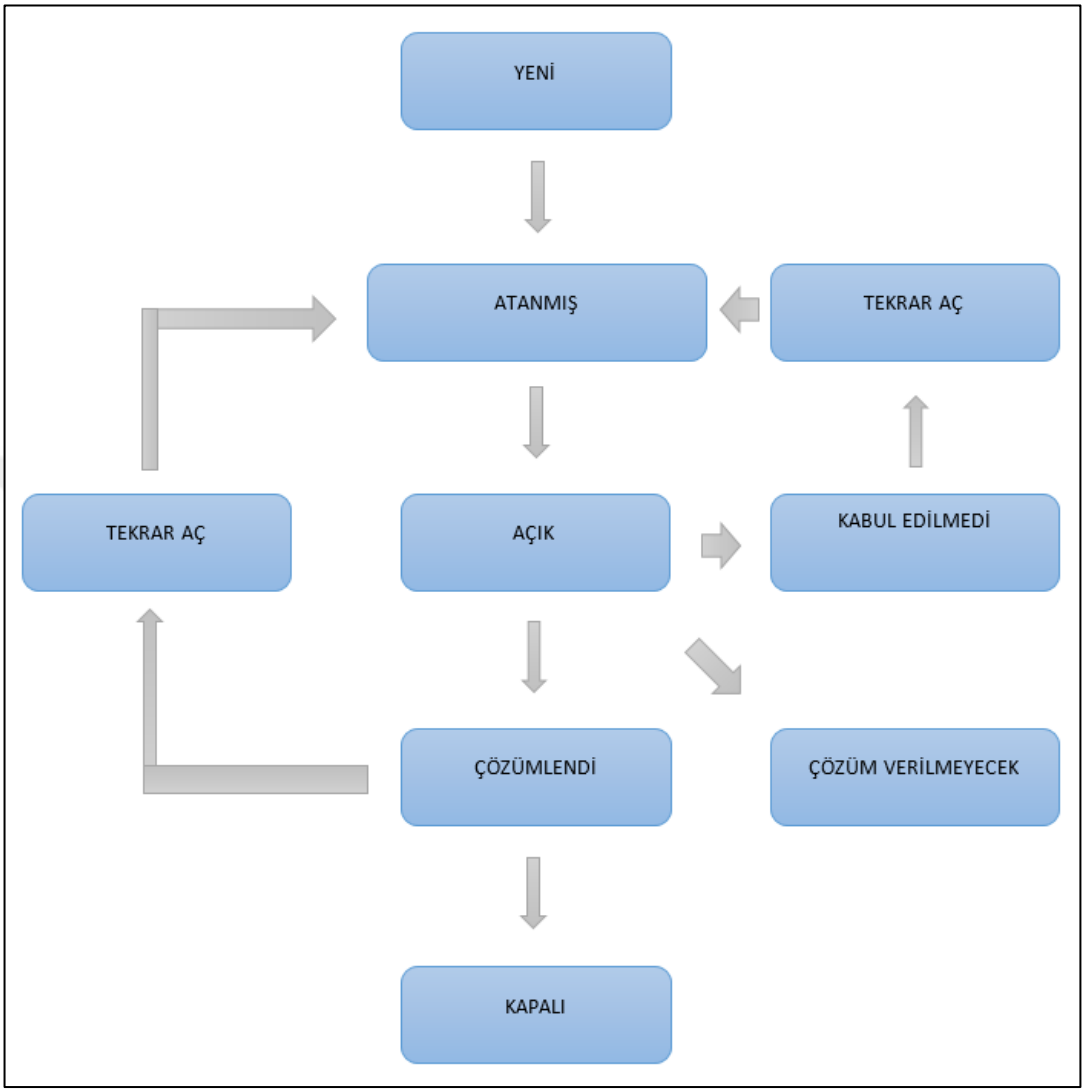
Atanmış (Assigned)

Bulunan ve yeni statüsünde olan hata, test lideri tarafından değerlendirilir ve hatanın detayları incelenir. İnceleme sonucunda sorunun gerçekten hata olduğu anlaşılırsa geliştirme ekiplerine atanır ve statüsü 'Atanmış' durumu geçer.

Açık (Open)

Aksiyonu geliştirme ekipleri üzerinde bulunan hatanın analizine ve çözümüne çalışıldığını göstermektedir. Hatanın çözümüyle uğraşıldığı bu aşamada hatanın önceliği ve şiddetine göre çözüm süresi göz önünde bulundurulmaktadır. Bu statüden sonra geliştirmeci hatanın statüsünü 'Çözümlendi', 'Kabul Edilmedi' ya da 'Çözüm Verilmeyecek' statüsüne çekerek ilerleyebilir.

Şekil 4: Yazılım Hataları Yaşam Döngüsü



Kaynak: Yazar Tarafından Derlenmiştir.

Çözümlendi (Fixed)

'Açık' statüsünde bulunan hatalara çözüm verildikten sonra statüsü 'Çözümlendi' olarak ilerletilir ve hatayı açan testçiye hatanın çözümünün iletildiği aşamadır. Geliştirmeci üzerinde bulunan aksiyon bu aşamada artık test uygulayıcısına geçmiştir ve testçi tarafından incelendikten sonra ilgili senaryosunun tekrar test edilmesi sürecine girilmiş olunur.

Kabul Edilmedi (Rejected)

Geliştirme ekiplerine iletilen ve 'Açık' statüsünde bulunan hatalar, geliştirmecinin incelemesi sonucu hata olmadığı kanısına varılırsa tekrar test ekibine gönderilir ve hatayı kabul etmemiş olur. Hatanın kabul edilmeme statüsü sadece geliştirmecinin hatayı kabul etmemesinden dolayı olmayabilir, eğer açılan hata içerisinde yeterli kadar log, ekran görüntüsü, hatanın gerçekleştiği ortam bilgisi, hatanın oluşturulduğu veri bilgisi bulunmaz ya da açıklamalarda eksiklik varsa test ekibinin detaylı olarak anlatılması istenildiğinden tekrar testçiye yönlendirilmesi için de kullanılır. Bazı test yönetim uygulamalarında hata içerisinde bulunan bilgi eksikliklerinin giderilmesi için "Kabul Edilmedi" statüsü yerinde "Bilgi İsteği" statüsüde bulunabilmektedir.

Tekrar Aç (Reopen)

Geliştirme ekibi tarafından 'Kabul Edilmedi' statüsüne getirilen hatalar test ekibi tarafından incelenir. Kabul etmeme sebebi açılan hatanın gerçekten bir hata olmadığını belirtmesidir.

- Kabul etmeme sebebi hata olmadığını belirtmesi durumunda, test ekibi tarafından hata tekrar detaylıca incelenir ve hatanın oluşup oluşmadığını tekrar gözlemler. Eğer hata oluşuyorsa farklı log ve verilerle tekrardan geliştirmeciye iletilebilir ve hatanın statüsü 'Tekrar Aç' statüsüne getirilmiş olur.

- Kabul etmeme sebebi hata olmadığını belirtmesi durumunda test ekibinin incelemeleri sonucunda geliştirmecinin haklı olduğu görülürse (test ortam farklılıklarından oluşacak test hatası, test veri hatası, yapılan testin hatası vb.) hata artık 'Tekrar Aç' statüsüne gönderilmez.

- Kabul etmeme sebeplerinden birisi de hatanın yeterince açıklayıcı olmadığı ya da log ve eklerin yeterli görülmediğini yeni ve ek bilgiler eklendiğini belirtmek için olabilir. Bu durumda ortada test ekibi hataya ait yeni log ve verileri tekrar oluşturur geliştirmeciler tarafından istenen ek bilgiler varsa onlar sağlandıktan sonra hatanın içerisine eklenir ve 'Tekrar Aç' statüsüyle geliştirmecilere atanır.

Çözüm Verilmeyecek (Deferred)

Test ekipleri tarafından bulunan ve geliştirme ekiplerine yönlendirilen hatalar incelendikten sonra hata olduğu görülür ve bu hata düşük şiddetli ise başka projelerde ya da daha sonraki sürümlerde çözümlenmesi kararıyla hatanın statüsü 'Çözüm Verilmeyecek' olarak güncellenir (Hambling ve diğerleri, 2010:234). Hatanın 'Çözüm Verilmeyecek' statüsüne getirilmesinin birden fazla sebebi olabilir. Test süresinin zamanında tamamlanabilmesi ve en az hata ile üretim ortamına çıkarılması için diğer hatalarla da ilgilenilmesi gerekir, eğer bulunan hatanın çözülmesi çok fazla zaman alacak ve proje takvimini engelleyecekse ileriki sürümlerde çözüm verilmesi için ötelenebilir. İlgili hatanın projeye etkisi az ise ve bu sürümde çözüm verilmesi kritik değilse de çözüm için öteleme yapılabilir. Ayrıca bulunan hatanın çözüm ekipleri tarafından incelenmesi sonucunda hata olup olmadığı kararına varamaması durumunda daha sonra detaylı analizi yapılması içinde 'Çözüm Verilmeyecek' statüsü kullanılır. Hataların 'Çözüm Verilmeyecek' statüsüne geçirilmesi sadece geliştirme ekibinin kararıyla olmayabilir, bu statüye geçirilmesi için proje yöneticisi veya analistlerin onayı da gerekebilir.

Kabul Edilmedi ve Kapatıldı (Rejected&Closed)

Test ekipleri tarafından bulunan hatalar geliştirmecinin incelemesi sonucu hata olmadığı kanısına varılırsa ve test ekibi tarafından tekrar incelenmesi sonucunda hata olmadığı görülürse 'Kabul Edilmedi ve Kapatıldı' statüsüne getirilir. 'Kabul Edilmedi ve Kapatıldı' statüsünde bulunan hataların yaşam döngüsü sonuçlanmış olur. Bu statüde bulunan hata sayısı test ekibinin test verimliliğinde düşüklüğe neden olmaktadır. Bu sayının artışı oluşuyorsa test ekibine proje hakkında eğitim ya da test ortamlarında düzeltme aksiyonları alınabilir. Test verimliliği test süreci boyunca sürekli takip edilmesi gereken bir değerlendirmedir. Verimlilik oranının düşük olması projenin zamanında tamamlanmasına engel olur ya da kalitesiz ürünlerin ortaya çıkmasına sebep olur. Hata olmayan bir senaryo üzerine çalışıldığından dolayı proje test sürecinde yaşanan zaman kaybından dolayı gerçekte hata olan senaryoların bulunamamasına neden olabilir.

Kapalı (Closed)

Geliştirme ekibi tarafından çözümlendi statüsünde olarak iletilen hataların, geliştirmenin ortama kurulumu yapıldıktan sonra tekrar testi gerçekleştirildikten sonra hatanın giderildiği görülürse testçi tarafından hata 'Kapalı' statüsüne getirilir. Kapalı statüsü hatalar için son statüdür ve kapatıldıktan sonra hata ile ilgili işlemlerin bitirildiği gösterilir.

Çözüm Verilmeyecek ve Kapatıldı (Deferred&Closed)

Çözüm verilmeyecek olan hatalara daha sonra çözüm verileceğini belirtmiştik. Hataya ait gerekli analiz ve inceleme sağlandıktan sonra 'Çözüm Verilmeyecek' statüsünde bulunan hatalar 'Çözüm Verilmeyecek ve Kapatıldı' statüsüne getirilir. Hatanın bu statüye getirilmesi için ayrıca bir talep açılır ya da başka bir proje kapsamına alınır.

1.6.2 Yazılım Hata Tipleri

Yazılım testindeki asıl amaç üründe bulunan hataların maksimum seviyede ortaya çıkarılmasıdır. Çıkan hatalar belirli kriterlere göre sınıflandırılmalıdır. Hata tipleri sektöre göre değişiklik gösterebilir fakat yazılımda genel olarak aşağıdaki gibi hata tipleri bulunmaktadır.

- Dokuman Hatası
- Kod Hatası
- Analiz Hatası
- Tasarım Hatası
- Ortam hatası
- Diğer

2009 yılında, Chen ve Huang tarafından yapılan bir e-posta anketine göre, çoğu yazılım projelerinde hataların ve tiplerin karakteristiği incelenmiştir ve Tablo 1'de özetlenmiştir (Chen ve Huang, 2009:987:990).

Tablo 1: Hata Tipleri Karakteristiği

Yüksek Şiddetli Problem Faktörleri		
#	**Yazılım Geliştirme Hata Faktörleri**	**Problem Boyutu**
1	Kaynak kod yorum eksikliği	Programlama kalitesi
2	Güvenilir olmayan dokümantasyon	Dokuman kalitesi
3	Değişikliklerin yeterince dokümantasyon edilmemesi	Dokuman kalitesi
4	İzlenebilirlik eksikliği	Dokuman kalitesi
5	Standart eksikliği	Programlama kalitesi
6	Bütünlük tutarsızlığı	Dokuman kalitesi
7	Sürekli değişen gereksinimleri	Sistem gereksinimi
8	Proje ekibinde sürekli değişiklik	İnsan kaynakları
9	Yanlış teknik kullanılması	Programlama kalitesi
10	Yazılım kalite gereksinimlerinin dikkate alınmaması	Sistem gereksinimi

Kaynak: Jie-Cherng Chen, Sun-Jen-Huang ," An empirical analysis of the impact of software development problem factors on software maintainability ",Journal of Systems and Software, Cilt:82, Sayı: 6, Haziran 2009, ss. 981-992.

Chen ve Huang'a göre, hataların büyük çoğunluğu analiz sırasında hatalı analizlerin yapılması, gereksinimlerin yanlış olarak aktarılması ya da net olmayan analizlerin çıkarılmasıdır. Hataların yarısı belirsiz ve hatalı gereksinimlerden dolayı oluşmaktadır (Mogyorodi, 2001:286-296).

1.6.2.1 Hata Şiddet Belirlenmesi

Hata şiddeti, bulunan hatanın sistemin çalışmasına etkisine göre belirlenir. Sistemin fonksiyonel olarak çalışmasına etkisi bulunmayan hatalar, gereksinimleri karşılamayan hatalara göre daha az şiddetli olarak sınıflandırılırlar.

Hata şiddeti, hatanın ne kadar kötü olduğunu gösterir ve bu hatanın ürüne olan etkisini yansıtır (Patton, 2001:289).

Hata şiddetleri genel olarak 4 tipte sınıflandırılır. Bunlar; kritik hata, majör hata, orta hata ve düşük hatadır.

Kritik seviyedeki hatadan kozmetik hatalara doğru hatanın şiddetinin azaldığı belirtilmektedir. Kritik seviyedeki hata sistemin çalışmasına engel olan, gereksinimlerden bir ya da birkaçının gerçekleşmesini engelleyen hatalar olarak örneklendirilebilir. Düşük hatalar ise herhangi bir fonksiyonel etkisi olmayan yazı tip ve renklerinde olan bozukluklar olarak değerlendirilebilir.

1.6.2.2 Hata Önceliğinin Belirlenmesi

Hata önceliği, bir hatanın ne kadar önemli ve acil olduğunu gösteren parametredir. Önceliğin belirlenmesinde proje takvimi de dikkate alınır, hatanın çözümlenme süresi ve tekrar testinin gerçekleştirilme süresi göz önünde bulundurularak seviye belirlenir. Hata seviyesi yüksek olan kusurlar öncelikli olarak düzeltilmesi gereken hatalardır.

Hata önceliği, hatayı düzeltmenin ne kadar önemli olduğunu ve ne zaman düzeltilmesi gerektiğini gösteren değişkendir (Patton, 2001:289).

Hata şiddetleri genel olarak 4 tipte sınıflandırılır. Bunlar; kritik hata, majör hata, orta hata ve düşük hatadır.

1.7. YAZILIM HATA DURUM RAPORLAMASI

Test süresi boyunca bulunan hatalar yazılımın kalitesi hakkında bilgi sahibi verir ve ürünün müşteriye sunulup sunulmamasına, alınması gereken önlemler için önceden bilgi vermesine, çıkabilecek maliyet ve risklerin ortaya çıkarılmasına yardımcı olmaktadır. Yazılımın kalitesi hakkında en kritik bilgiyi test sürecinde bulunan hatalar göstermektedir. Bu bilgilerin yöneticilere sunulması için bulunan hatalardan analiz ve raporlamaların yapılması gerekmektedir. Geliştirme ekipleri, proje paydaşları ve yöneticilere çıkartılabilecek raporlamalar aşağıdaki gibidir.

- Hata tipine göre dağılımı grafiği
- Hata şiddetine göre dağılım grafiği
- Hata önceliğine göre dağılım grafiği

- Hata durumuna göre dağılım grafiği
- Hata sayısına göre verimlilik durumu

1.7.1 Hata Tipine Göre Dağılım

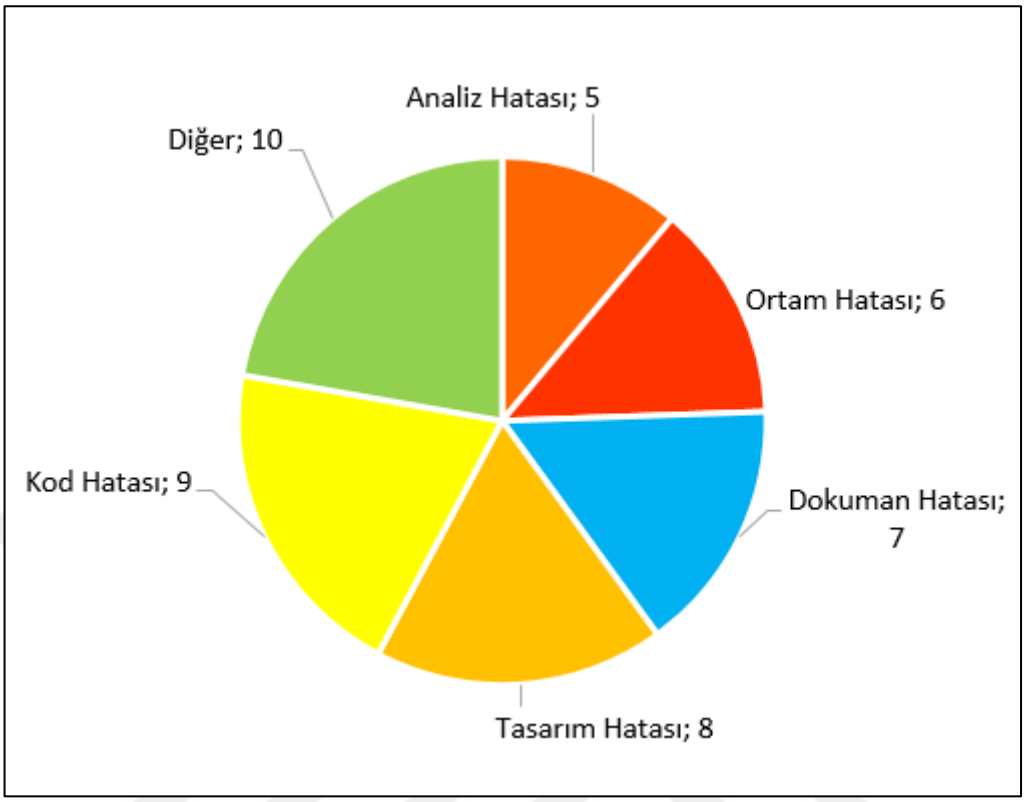
Testlerde bulunan hataları dokuman hatası, kod hatası, analiz hatası, tasarım hatası, ortam hatası ve diğer hatalar olarak sınıflandırdık. Bulunan hataların sınıfına göre hataların ana sebebinin nereden kaynaklandığını daha çok hangi ekiplerin hangi noktalarda yoğunlaşması gerektiğinin çıkarılacağı raporlama tipi "Hata Tipine Göre Dağılım" grafiği olacaktır. Şekil 5'te hataların hata tipine göre dağılımı gösterilmiştir. Bu dağılım grafiklerine göre hataların en çok hangi tipte olduğunu hangi bölüme daha çok odaklanılması gerektiği görülebilmektedir.

- Dokuman Hatası
- Kod Hatası
- Analiz Hatası
- Tasarım Hatası
- Ortam hatası
- Diğer

1.7.2 Hata Şiddetine Göre Dağılım Grafiği

Test durum raporlarında kullanılması gereken ve hataların önem derecelerini yönetici ve ilgili düzeltme ekibine gösteren dağılım grafiği şiddet dağılım grafiğidir. Hataların şiddeti arttıkça öncelikli düzeltilmesi büyük önem kazanmakta ve düzelene kadar da diğer gereksinim ve fonksiyonelitelerin etkilendiği gösterilmektedir. Bu raporlama sayesinde belli sayıda kritik ya da majör sayıda bulunan hataların çözümü geciktiğinde ya da proje başından itibaren bulunan hataların şiddeti kritik ya da majör ise; projenin durdurulması ya da alınacak başka kararlar için yol gösterici rolünde olacaktır. Hata şiddetine göre dağılım gösterimi Şekil 6'da gösterildiği gibi pasta grafik yöntemiyle ya da bir tablo biçiminde raporlanabilir.

Şekil 5: Hata Tipine Göre Dağılım

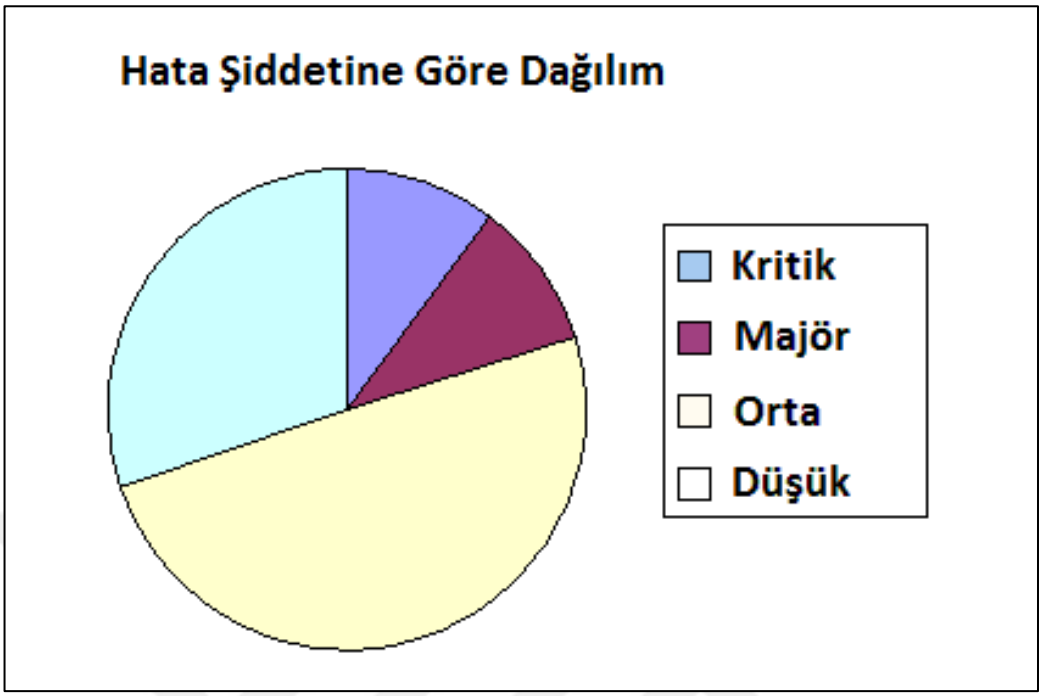


Kaynak: Yazar Tarafından Derlenmiştir.

1.7.3 Hata Önceliğine Göre Dağılım Grafiği

Hata önceliğine göre verilen hata dağılım grafikleri önceliklendirilmesi gereken hataları çözüm ekiplerine gösterir ve bu öncelik sırasına göre hata çözümlerinin alınması hedeflenmiş olur. Hata öncelik grafiğine göre; büyük oranda açık statüde bulunan hatalar varsa, proje planında değişiklik yapılması için ya da riskleri belirtmek için bu grafik raporlamalar arasına eklenmelidir. Hata çözümlerinin öncelikle çözüm alınması gereken hataları gösterir ve çözüm ekiplerine yol gösterici rolünü alır. Hata önceliğine göre dağılım grafikleri pasta grafik yöntemiyle ya da bir tablo biçiminde raporlanabilir.

Şekil 6: Hata Şiddetine Göre Dağılım



Kaynak: Test Evaluation Summary for the Architectural Prototype https://sceweb.uhcl.edu/helm/RUP_course_example/courseregistrationproject/artifacts/test/results/test_report_arch.htm, (29.04.2019).

1.7.4 Hata Durumuna Göre Dağılım Grafiği

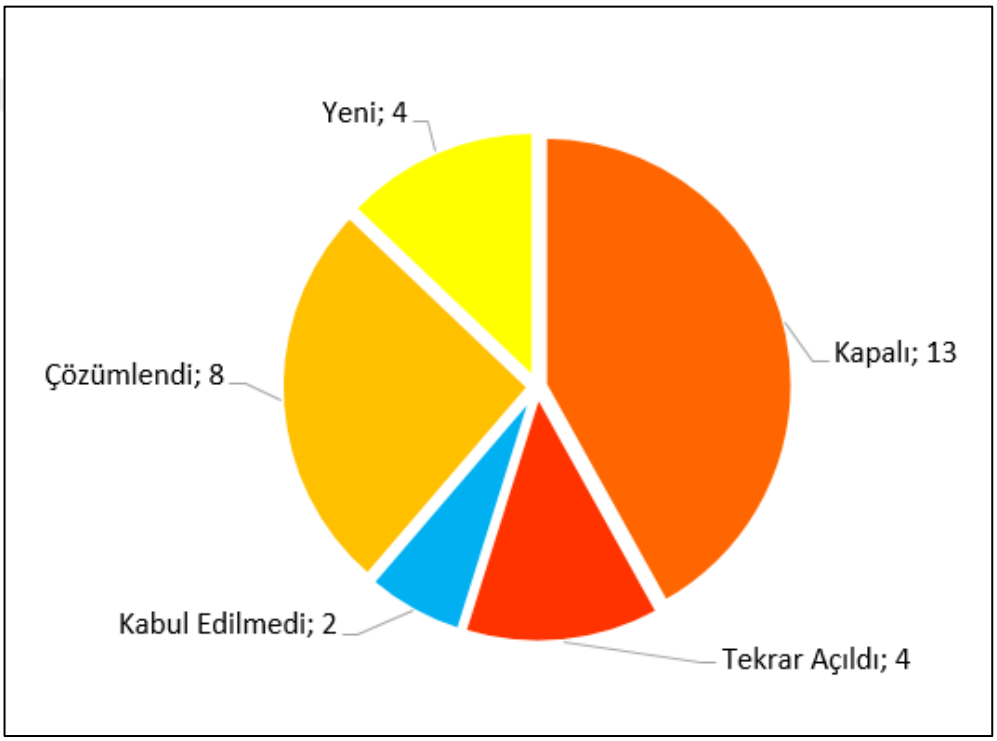
Yazılım hata durumları yeni, atanmış, açık, çözümlendi, kabul edilmedi, kabul edilmedi ve kapatıldı, tekrar aç, çözüm verilmeyecek ve kapalı statüsünde olabilmektedir. Yazılımda çıkabilecek hatalar bu yaşam döngülerini gerçekleştirmektedir. Bu statülere göre çıkan hatalar sınıflandırılmalı ve raporlarda mutlaka sunulmalıdır. Statü raporları yöneticilere, çözüm ekiplerine ve test ekiplerine proje hakkında detaylı bilgi sağlayabilmektedir. Örneğin projede açık ya da yeni statüsünde bulunan hata oranının fazla olması proje planında risk oluşacağını çok fazla hatanın biriktiğini göstermektedir. Düzenli raporlamalar yapılarak hataların çözümleri sürekli takip edilmeli ve açık ya da yeni statüsünde bulunan hata oranlarının birikmesi engellenmiş olacaktır.

Düzenli raporlamalar sayesinde test ekibi kendi kalitesini ve alması gereken aksiyonları görüyor olacaktır. Örneğin proje hata dağılım raporlarında kabul edilmedi

ve kapatıldı ya da çözümlendi statüsündeki oranın artışı, test ekibi üzerinde işlerin biriktiği ve aksiyon almaları gerektiğini göstermektedir.

Ayrıca ekiplerin kalitelerinin göstergesi için de kullanılabilecek bir rapor olarak da kullanılabilir. Açılan hatalarda çok fazla kabul edilmedi ve kapatıldı statüsü oranındaki artış, test ekibine aksiyon olarak yansıyacaktır. Test hata raporundaki bu artış test ekibinin hataları açmadan tekrar tekrar kontrol etmeleri gerektiğini göstermektedir.

Şekil 7: Hata Durumuna Göre Dağılım



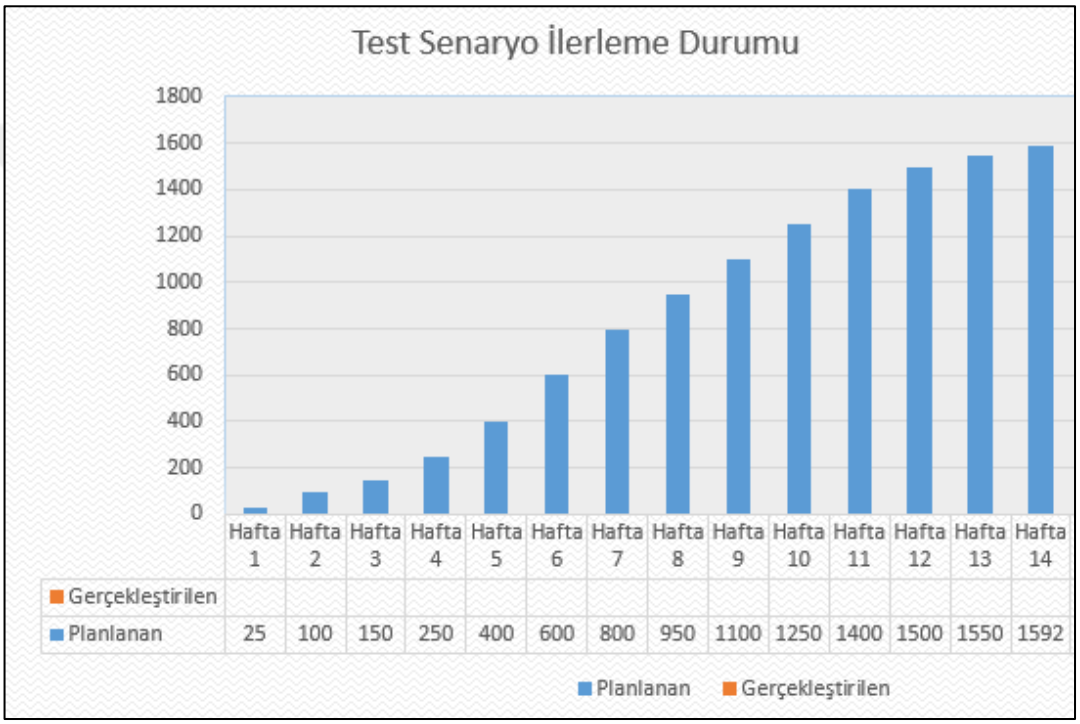
Kaynak: Yazar Tarafından Derlenmiştir.

Şekil 7'de bulunan hata durumuna göre dağılım grafiğinde açıkta bekleyen hataların test ve geliştirme ekipleri üzerinde olduğu görülmektedir. Test ekibinde 8 adet aksiyon alınması gereken hata bulunmakta ve projede gecikme yaşanmaması için çözümlenen bu hataların tekrardan kontrolleri ve testlerinin yapılması gerekmektedir. Ayrıca yeni statüsünde ve tekrar açıldı statüsünde bulunan toplamda 8 adet hata için de geliştirme ekiplerinin hatalar için çözüm sağlaması beklenmektedir. Proje planının etkilenmemesi ve risklerin önceden görülebilmesi için açık statüde bulunan hata oranının azaltılması gerekmektedir.

1.7.5 Test Senaryo İlerleme Grafiği

Proje ekibine test sürecinin tamamlanma oranı ve plana uygunluk durumu Şekil 8'de bulunan "Test Senaryo İlerleme Grafiği" ile gösterilir. Toplam test senaryo sayısına göre test senaryo ilerleme grafiği oluşturulur ve belirli periyotlar ile proje ekibine sunulan raporlarda paylaşılır.

Şekil 8: Test Senaryo İlerleme Durumu



Kaynak: Kshirasagar Naik ve Priyadarshi Tripathy, Software Testing and Quality Assurance: Theory and Practice, 2008 John Wiley & Sons, ss. 417.

Tabloda planlanan test senaryo sayısının gerçekleştirilen test senaryo sayısına göre ilerlemesi gösterilmektedir. Tabloya göre proje testinde toplam 1592 adet gerçekleştirilmesi gereken test senaryosu bulunmaktadır. Bu senaryoların 14 haftada tamamlanacak şekilde planlamasının yapıldığı görülmektedir. Plana göre ilk hafta gerçekleştirilecek senaryo sayısı 25, ikinci hafta gerçekleştirilecek senaryo sayısı 75 olarak belirlenmiştir. Test senaryolarının uygulanmasına başlanıldığında haftalık olarak tabloda gerçekleştirilen senaryo sayısı alanı güncellenecektir. Test ilerleme grafiğine göre proje ekibi test ilerleme durumu hakkında bilgi sahibi olacaktır. Bu

tabloya göre; gerçekleştirilen senaryo sayısı planlanan senaryo sayısından az ise, test/proje yöneticisi projede gecikme olduğunu fark edecek ve projede oluşacak riskler hakkında önceden bilgilendirilmiş olacaktır (Naik ve Tripathy, 2008:415-419).

1.7.6 Hata ve Senaryo Sayısına Göre Analiz Raporları

Hata sayılarına çeşitli analizler çıkarılır ve bu analizlerin yorumlanmasıyla test verimliliği, proje kalitesi gibi önemli sonuçları ortaya çıkarılabilir. Tüm projelerin test aşamasında aşağıdaki raporların çıkarılması gerekmektedir.

Test Verimlilik Durumu

Testler sırasında hataların bulunması ve kaçırılmaması test verimliliğini göstermektedir. Test verimliliği ölçümünü aşağıdaki formül ile yapılır (Lazic ve Mastorakis, 2008:599-619).

$$Test\ Verimliliği = \frac{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı}{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı + Canlı\ Sistemde\ Bulunan\ Hata\ Sayısı} \times 100$$

Proje sonunda ortaya çıkan ürünün canlı ortamda devreye alınması sonrası oluşacak hata analizlerine göre projenin test verimliliğini hesaplamak mümkün olacaktır. Bu hesaplamada ürünün üretim ortamına alım sonrası müşteri/kullanıcı tarafından bulunan hata sayısı kullanılmaktadır. Bulunan hatalar proje test sürecinde testçiler tarafından bulunamamış hatalı senaryoları ifade etmektedir (Jones, 2008:177-179).

Kullanılan test verimliliği formülünde, testler sırasında bulunan hata içerisinde mükerrer olmayan ve sadece kabul edilmiş hataların olması gerekmektedir. Mükerrer, kabul edilmeyen hatalar bu sayıya dâhil edilmezken "çözüm verilmeyecek" statüsünde bulunan hataların eşitliğe dâhil edilmesi gerekmektedir. Canlı sistemde müşteri tarafından bulunan hataların ise aynı şekilde mükerrer ve kabul edilmeyen hata sayıları çıkarılarak hesaplanmalıdır (Black, 2002:164-166).

Çözümlenen Hata Oranı

Çözümlenen hata oranı çözümlenmiş hata sayısının toplam hata sayısına oranına göre aşağıdaki formül ile hesaplanmaktadır.

$$Çözümlenen\ Hata\ Oranı\ = \frac{Çözümlenmiş\ Hata\ Sayısı}{Toplam\ Hata\ Sayısı} \times 100$$

Başarısız Test Senaryo Oranı

Başarısız test senaryo testler sırasında başarısız olan test senaryo sayısının toplam gerçekleştirilen test senaryo oranına göre aşağıdaki formül ile hesaplanmaktadır.

$$Başarısız\ Test\ Senaryo\ Oranı\ = \frac{Başarısız\ Test\ Senaryo\ Sayısı}{Gerçekleştirilen\ Test\ Senaryo\ Sayısı} \times 100$$

Başarısız test senaryo oranının düşük olması, test ekibine iletilen kodun ne kadar düzgün olduğunu gösterir. Bu oranın yüksek olması kodun birim test eksikliğini ya da kalitesiz bir kod iletildiğinin göstergesidir. Bu tarz bilgilerle, proje ekibi kaliteyi artırmak adına bazı önlemler alabilir. Alınan önlemlerden sonra bu oranın takibine devam edilir ve ileride yapılacak projeler için yol gösterici metotlar geliştirilir (Lazic ve Mastoralis, 2008:603-604).

Kabul Edilmeyen Hata Oranı

Kabul edilmeyen hata oranı göre aşağıdaki formül ile hesaplanmaktadır.

$$Kabul\ Edilmeyen\ Hata\ Oranı\ = \frac{Kabul\ Edilmeyen\ Hata\ Sayısı}{Toplam\ Hata\ Sayısı} \times 100$$

Çözümlenmeyecek Hata Oranı

$$\text{Çözümlenmeyecek Hata Oranı} = \frac{\text{Çözümlenmeyecek Hata Sayısı}}{\text{Toplam Hata Sayısı}} \times 100$$

Hata Çözümleri İçin Harcanan Ortalama Süre

$$\text{Hata Çözüm Süresi} = \frac{\text{Çözümlendi Statüsüne Kadar Geçen Süre}}{\text{Toplam Hata Sayısı}} \times 100$$

Test Sayısına Göre Hata Oranı

$$\text{Hata Oranı} = \frac{\text{Kabul Edilmeyen Hata Hariç Toplam Hata Sayısı}}{\text{Toplam Test Senaryo Sayısı}} \times 100$$

Tekrar Açılan Hata Oranı

$$\text{Tekrar Açılan Hata Oranı} = \frac{\text{Tekrar Açılan Hata Sayısı}}{\text{Toplam Hata Sayısı}} \times 100$$

Regresyon Test Hata Durumu

Fonksiyonel testler sırasında bulunan hatalar ile regresyon testleri sırasında bulunan hataların oranlaması ile bulunur. Bu oran fonksiyonel testlerin verimliliğini göstermektedir. Aşağıdaki formül ile regresyon hata durum oranına ulaşılır.

$$\text{Regresyon Hata Durumu} = \frac{\text{Regresyon Süresinde Bulunan Hata Sayısı}}{\text{Toplam Bulunan Hata Sayısı}} \times 100$$

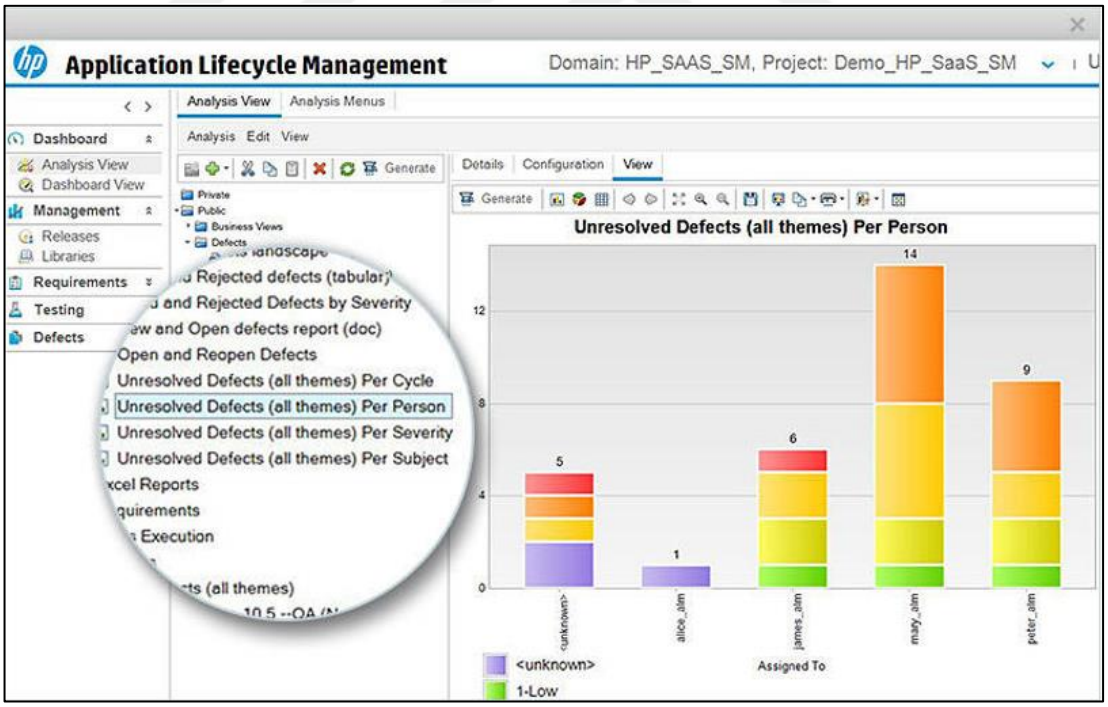
1.8. MEVCUT YAZILIM TEST VE HATA YÖNETİM UYGULAMALARI

1.8.1 HP Application Lifecycle Management and HP Quality Center Enterprise

HP ALM (Uygulama Yaşam Döngüsü Yönetimi), kuruluşların uygulama yaşam döngüsünü proje planlamadan, gereksinim toplamasına yardımcı olan ve test yönetiminin sağlanmasına yardımcı olan web tabanlı bir araçtır. Daha önceden Kalite Merkezi (Quality Center) olarak Mercury interactive tarafından geliştirilmiştir ve lisanslı olarak kullandırılmaktadır. Şekil 9'da HP ALM arayüzü ve örnek bir defect raporu bulunmaktadır.

Artık HP tarafından, yazılım geliştirme yaşam döngüsünün çeşitli aşamalarını destekleyen Uygulama Yaşam Döngüsü Yönetim Aracı (veya) ALM olarak geliştirilmiştir (Guru99, 2018).

Şekil 9: HP ALM Arayüzü



Kaynak: Software Testing Tools Guide, HP ALM http://www.testingtoolsguide.net/tools/hp-alm/,(15.10.2018).

1.8.2 QA Complete

QAComplete, test yönetim araçları içerisinde kullanımı en kolay test senaryosu yönetim aracı olmak, mevcut süreçlerinize uyum sağlamak ve tüm testlerinizi rapor ederken tüm manuel ve otomatik testlerinizi kolaylıkla yönetmenizi sağlamak için temelden oluşturulmuştur. Tek bir merkezi konumda sonuçlanır. QAComplete, testleri yönetmek için daha az zaman harcanmasına ve daha çok teste odaklanılmasına izin verir.

QAComplete, yazılım test süreçlerinde görünürlük ve güven kazanmak için dünyanın her köşesinde ve her büyük sektördeki dikey ve büyük tüm yazılım ekipleri tarafından güvenilmektedir (SmartBear, 2018).

1.8.3 TestRail

TestRail, yazılım test eforlarının yönetilmesine ve izlenmesine olanak sağlamaktadır. Ayrıca, Kalite Güvence departmanlarındaki işlerin düzenlenmesine yardımcı olmaktadır. Sezgisel web tabanlı kullanıcı arabirimi, test senaryo oluşturulmasına, test çalıştırmalarının yönetilmesine ve test sürecini koordine etmeyi kolaylaştırır. Şekil 10'da TestRail arayüz görüntüsü bulunmaktadır.

Kontrol panelleri ve etkinlik raporlarıyla bağımsız testlerin, kilometre taşlarının ve projelerin durumunu kolayca izlenebilir ve takip edilebilir. Test ilerlemesi hakkında gerçek zamanlı bilgiler verebilir ve kişiselleştirilmiş yapılacaklar listesi, filtre ve e-posta bildirimleri ile verimliliği artırır (Testrail, 2018).

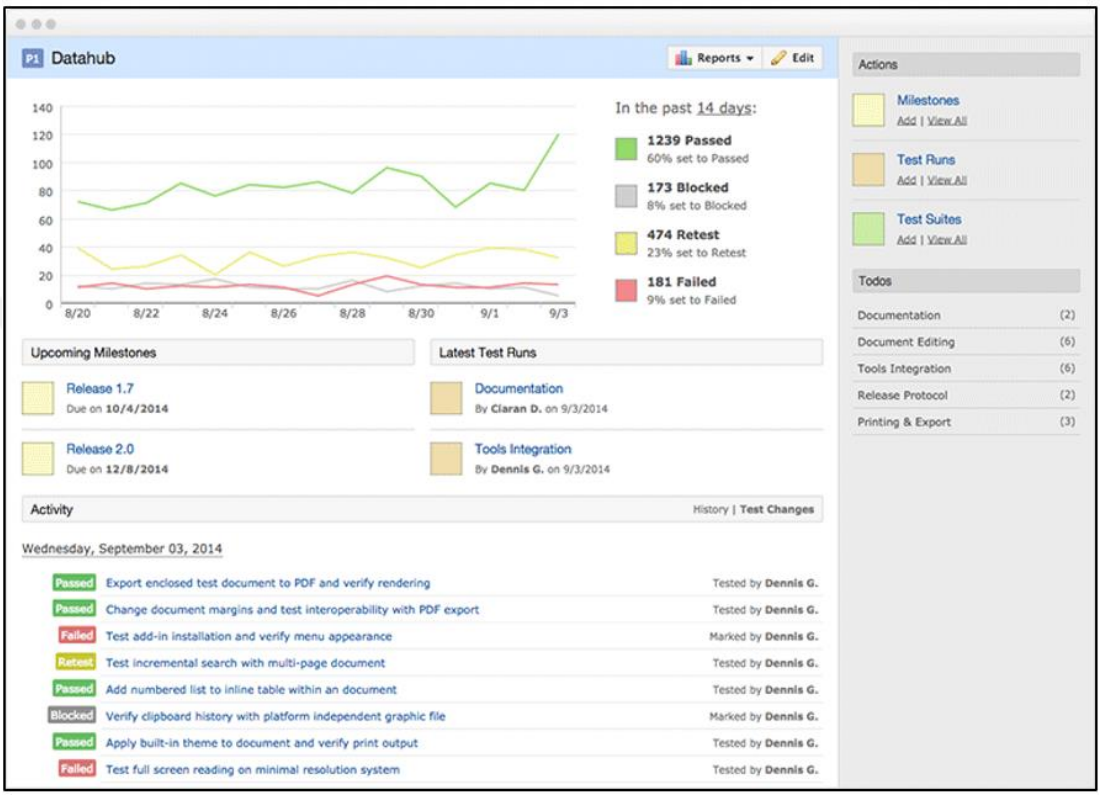
1.8.4 TestLodge

TestLodge, test planlarının ve test senaryo yazılmasını yönetilmesini sağlayan, test koşumlarının gerçekleştirilmesine izin veren çevrimiçi bir test yönetim aracıdır. TestLodge, testlerinizin ilerlemesini organize etmek, ortak çalışmak ve izlemek için kolay bir yol sağlayarak test sürecini geliştirir (TestLodge, 2018).

TestLodge içerisinde yer alan projeler alanı, tüm test verilerinizin yaşadığı yerlerdir (test planları, test vakaları, raporlar, vb.). TestLodge hesabı birden fazla

projeye sahip olma özelliğine sahiptir. Üzerinde çalışılan her bir proje için ayrı ayrı proje oluşturma imkanı sağlamaktadır. Şekil 11'de TestLodge da yeni proje oluşturma ekranı bulunmaktadır.

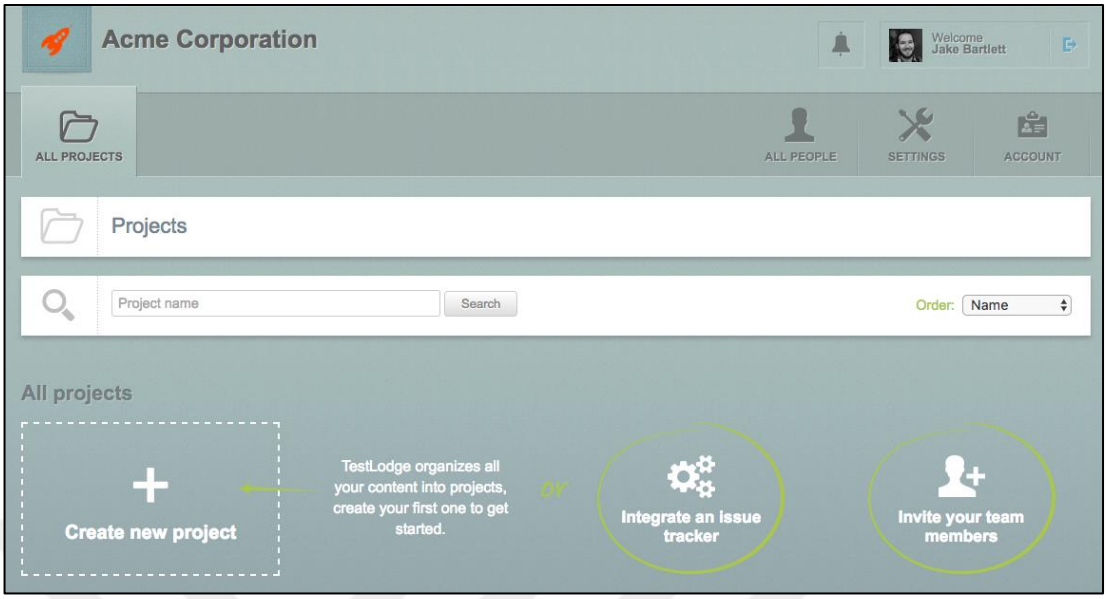
Şekil 10: TestRail Arayüzü



Kaynak: TestRail, Modern test management for your team http://www.gurock.com/testrail/,(15.10.2018).

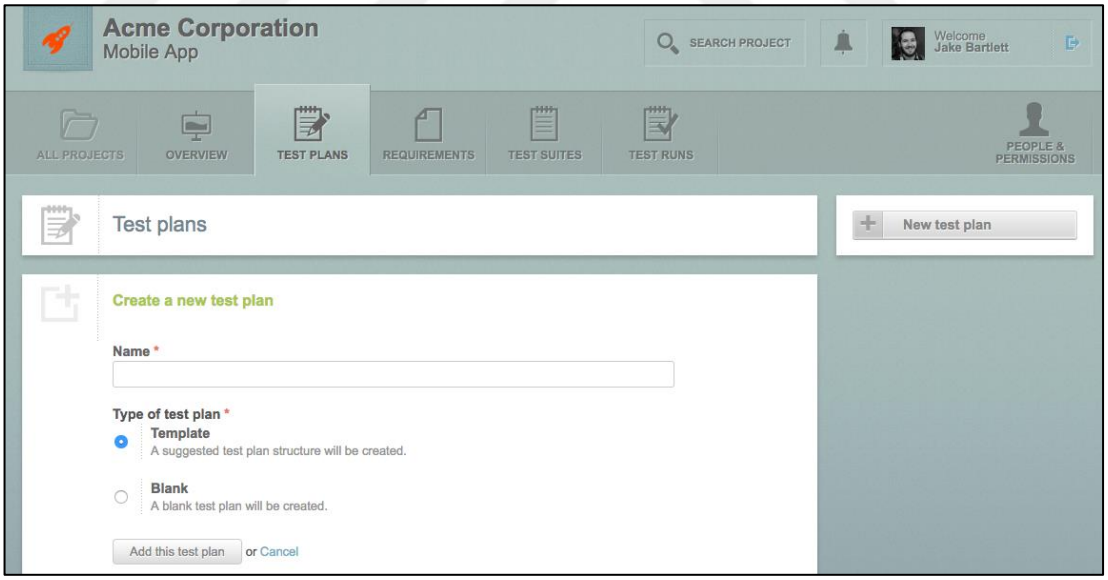
Test planı zaman çizelgeleri, bağımlılıklar, kaynaklar, başarılı / başarısız test senaryolar hakkında ayrıntılar sağlar. Bir "proje planı"' na benzer şekilde, test projesinin etrafında aynı sayfada tutulmasına yardımcı olur. TestLodge, şablon kullanarak test planı oluşturulmasını veya sıfırdan test planı oluşturmasını sağlayabilir. Şablon, test planına eklenmesi istenebilecek şeyler için fikirler ve ilham kaynağı sağlar. Şekil 12'de test plan sayfası gösterilmektedir.

Şekil 11: TestLodge Yeni Proje Yaratma Arayüzü



Kaynak : TestLodge, Getting Started With TestLodge https://help.testlodge.com/hc/en-us/articles/236206547-Getting-Started-With-TestLodge (26.10.2018).

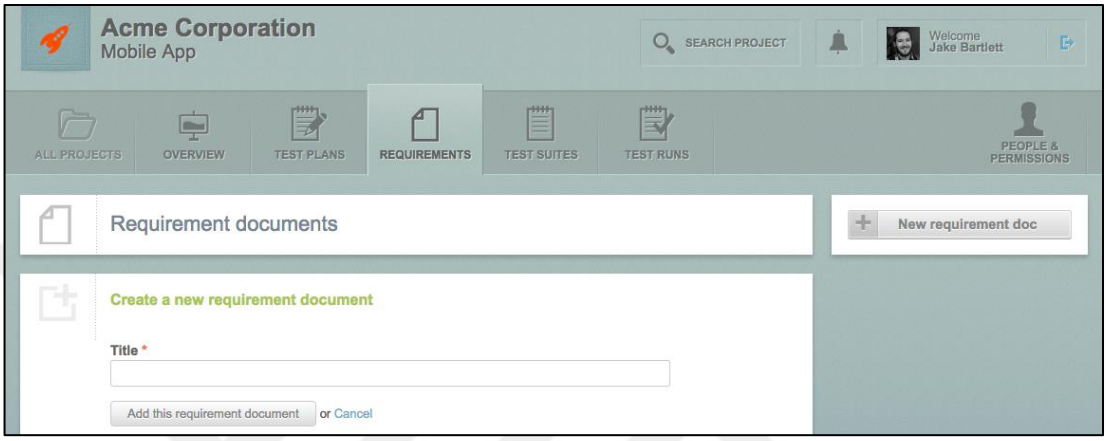
Şekil 12: TestLodge Yeni Test Planı Sayfası



Kaynak : TestLodge, Getting Started With TestLodge https://help.testlodge.com/hc/en-us/articles/236206547-Getting-Started-With-TestLodge (26.10.2018).

Gereksinimlerin ve kullanıcı hikayelerinin saklaması için gereksinim belgesi kullanılabilir. Daha sonra test durumları ile bağlantı kurularak gerçekleştirilen senaryolar ile gereksinimlerin tam olarak karşılandığının doğrulaması yapılabilir. Şekil 13'de TestLodge'da gereksinim sayfası gösterilmektedir.

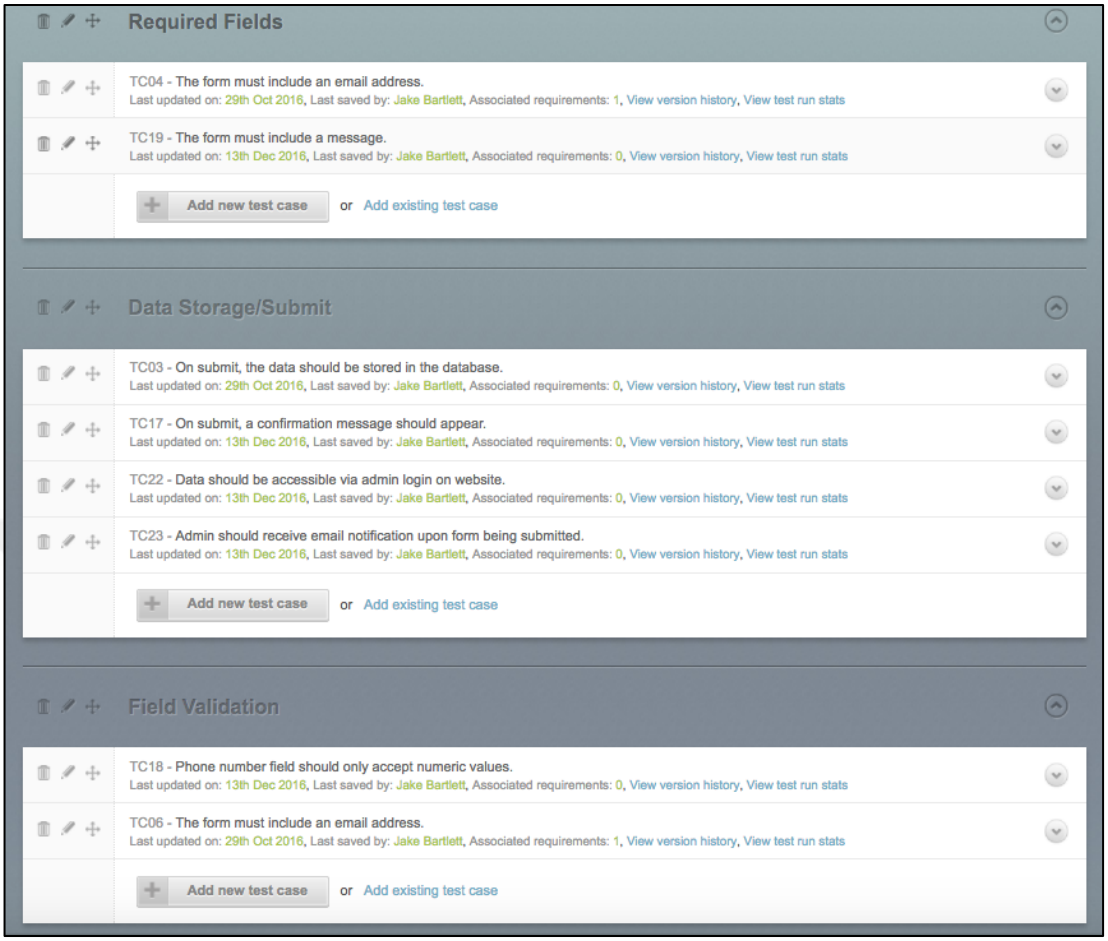
Şekil 13: TestLodge yeni test gereksinimi sayfası



Kaynak : TestLodge, Getting Started With TestLodge https://help.testlodge.com/hc/en-us/articles/236206547-Getting-Started-With-TestLodge (26.10.2018).

Test senaryo belgeleri, test için en önemli belgelerdir. Bunlar, testçiye belirli bir fonksiyonelitenin nasıl test edileceğini adım adım anlatan olay dizisidir. Test senaryoları, test durumlarının saklandığı "klasör" olarak düşünülebilir. Başka bir deyişle, test senaryoları test durumlarının bir toplamıdır.

Şekil 14: TestLodge Test Durum Kontrol Sayfası



Kaynak : TestLodge, Getting Started With TestLodge https://help.testlodge.com/hc/en-us/articles/236206547-Getting-Started-With-TestLodge (26.10.2018).

İKİNCİ BÖLÜM
UYGULAMA

2.1 UYGULAMA DONANIM VE YAZILIM BİLEŞENLERİ

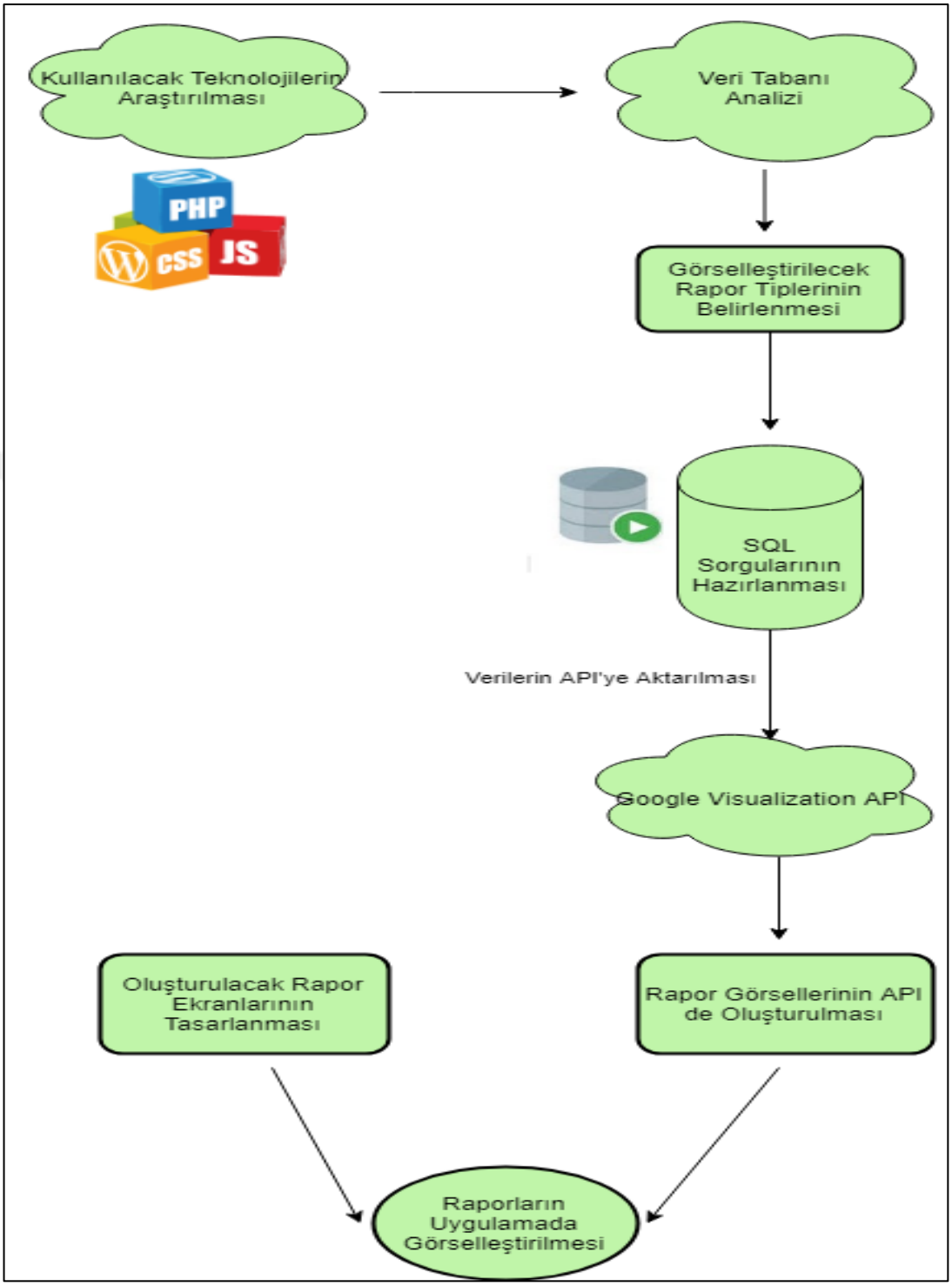
Tez kapsamında HP ALM uygulamasından test raporlama grafiklerinin ve test durum raporlarının oluşturulacağı bir rapor modülü geliştirilmiştir. Veriler Oracle veritabanında saklanmakta ve verilerin işlenmesiyle elde edilen görseller PHP ile geliştirilmiştir. Veritabanı ve PHP arasında bağlantı Oracle Call Interface (OCI) teknolojisi kullanılarak yapılmıştır. "OCI, özel veya paketlenmiş uygulamalar için kapsamlı, yüksek performanslı, Oracle Veritabanına özgü yerel C dili arayüzüdür (Oracle, 2019)". Kullanılan OCI teknolojisine ait kod Ek 1'de gösterilmektedir.

Raporların görselleştirilmesinde JavaScript yazılım dili, Google Görselleştirme API (Google Visualization API) ve ZingChart API kullanılmıştır. Google Görselleştirme API'si, yapılandırılmış verilere erişimi ve web sayfalarında JavaScript kullanarak bu verilerin görselleştirilmesini sağlar. (Programmable Web, 2019) Ek 2'de Google Görselleştirme API'ye ait kod parçası eklenmiştir. Bazı görsellerin daha anlaşılır olması ve Google Görselleştirme API'de olmayan bazı özelliklerden dolayı Zingchart API de kullanılmıştır. Zingchart API sayesinde oluşan grafikler üzerinde düzenlemeler ve görsellerde değişiklik yapılabilmektedir. Zingchart API'nin kullanımına ait kod parçacıkları Ek 3'de bulunmaktadır.

Geliştirmede açık kaynak kod şeması kullanılmıştır. Menü ekranları olarak GitHub'da bulunan "pro-sidebar-template" kullanılarak geliştirmeler yapılmıştır. Açık kaynak kodlu şema içerisinde Java Script, CSS ve HTML kodları bulunmaktadır.

Yapılan geliştirmede yazılım projelerinde en çok kullanılan ve oluşturulması uzun süren raporların anlık olarak ulaşılabileceği test ve defect rapor sayfaları oluşturulmuştur. Uygulamaya giriş yapıldıktan sonra istenilen projelerin test senaryo, defect durumları, defect detayları ve proje hakkında detaylı test durum raporları oluşturulabilmektedir. Proje bazlı raporlar dışında ekip bazında genel test durum raporları oluşturulabilmektedir. İstenilen ekip ya da kişilerin buldukları hatalara ait grafikler ve gerçekleştirilen test senaryo sayılarına ait raporlamalara yapılan geliştirme sayesinde anlık ve kolaylıkla ulaşılabilmesi hedeflenmiştir.

Şekil 15: Uygulama Akış Şeması



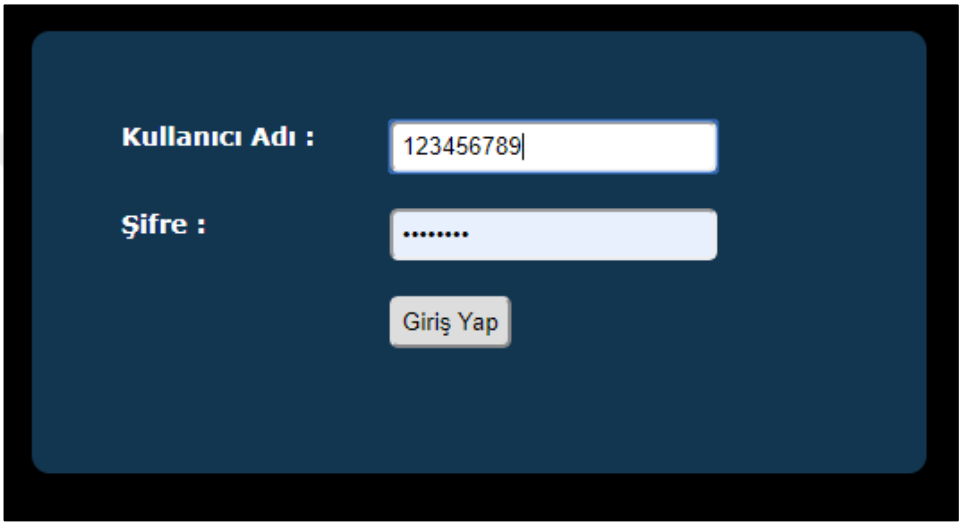
Kaynak: Yazar Tarafından Derlenmiştir.

2.2 UYGULAMA EKRANLARI

2.2.1 Rapor Portal Giriş Ekranı

Portal giriş ekranı Şekil 16'da olduğu gibi "Kullanıcı Adı" ve "Şifre" alanlarından oluşmaktadır. Kullanıcı adı ve şifre alanları veri tabanı üzerinden kontrol edilerek kullanıcı girişini sağlamaktadır.

Şekil 16: Rapor Giriş Ekranı



Kaynak: Yazar Tarafından Derlenmiştir.

2.2.2 Rapor Portal Anasayfa

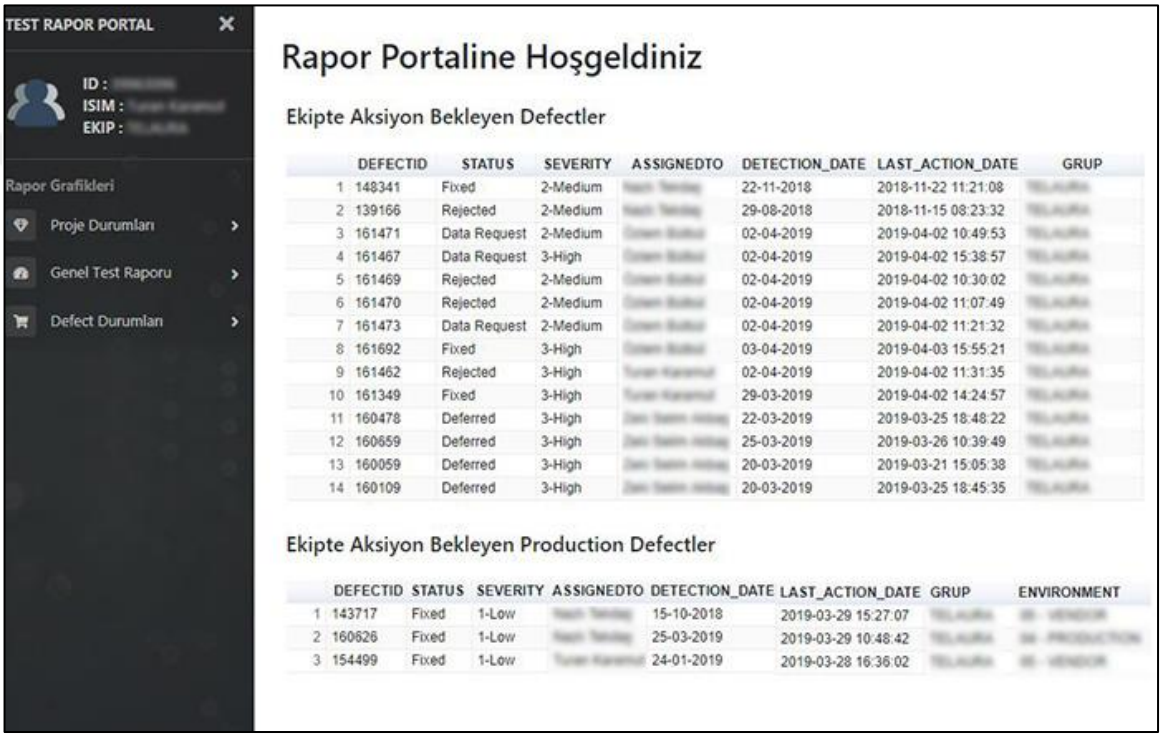
Kullanıcı portal ekranına giriş yaptıktan sonra Şekil 17'de ki ekran ana ekran olarak karşısına çıkmaktadır. Sisteme giriş yapan kullanıcının grubunda bulunan kişilere ait aksiyon bekleyen hataları göstermektedir. Hatalar test sırasında bulunan hataları ve canlı sistemdeki hataları göstermektedir. Sisteme giriş yapan kullanıcının id, isim ve ekip bilgisi veritabanından çekilerek ekranda gösterilmektedir. Aksiyon alınması beklenen hataların hata numarası, hata statüsü, hata şiddeti, aksiyonun kimde beklediği, hatanın açılma tarihi, hata ile ilgili son yapılan işlem tarihi ve hatanın hangi ekipte bulunduğu bilgisi gösterilmektedir.

Şekil 17'nin sol tarafında bulunan menüde "Proje Durumları" , "Genel Test Raporu", ve "Defect Durumları" seçenekleri bulunmaktadır. "Proje Durumları"

menüsü altında projeye ait test senaryo durum ve test hata durum raporlarının detayları bulunmaktadır.

"Genel Test Raporu" menüsü altında toplam gerçekleştirilen test senaryo sayısına ait grafikleri göstermektedir. Test ekipleri bazında test sayıları ve kişi bazında gerçekleştirilen test senaryo sayı grafiklerini göstermektedir. "Defect Durumları" menüsü altında açılan hatalara ait detaylı analiz raporlarını göstermektedir.

Şekil 17: Rapor Portal Anasayfa



Kaynak : Yazar Tarafından Derlenmiştir.

2.2.3 Proje Durum Raporu

Şekil 18'de "Proje Durumları" menüsünde seçili projeyi takip eden proje yönetici, test ekipleri, geliştirme ekipleri, test yöneticileri gibi ilgili kişilere sunulacak raporlamalar sunulmaktadır. Proje Durumları menüsünde oluşturulacak raporlamalar sayesinde ilgili kişiler projenin durumu hakkında bilgi sahibi olur ve alınacak aksiyonlar hakkında güncel bilgilere sahip olunacaktır. Projede oluşacak riskler ve alınması gereken önlemler için proje ekibine bilgi sağlayacak raporlamalar üretilmektedir.

Şekil 18: Proje Durumları Menüsü

Kaynak: Yazar Tarafından Derlenmiştir.

2.2.4 Proje Seçimi Yapılır

Proje Durumları menüsü altında ilk olarak hangi projeye ait rapor oluşturulması istenildiği sorulmaktadır. Seçilecek projeler HP ALM (Application Life Cycle Management) veri tabanından çekilerek kullanıcıya açılır liste halinde sunulmaktadır.

Şekil 19'da proje seçimi ekranında açılan kutu (combobox) içerisinde proje isimleri bulunmaktadır. Proje seçimi yapıldıktan sonra "Proje Seç" butonuna tıklandığında proje raporu oluşturulduğu görülür.

Şekil 19: Proje Seçim Ekranı

Kaynak: Yazar Tarafından Derlenmiştir.

2.2.5 Seçili Projeye Ait Oluşan Rapor

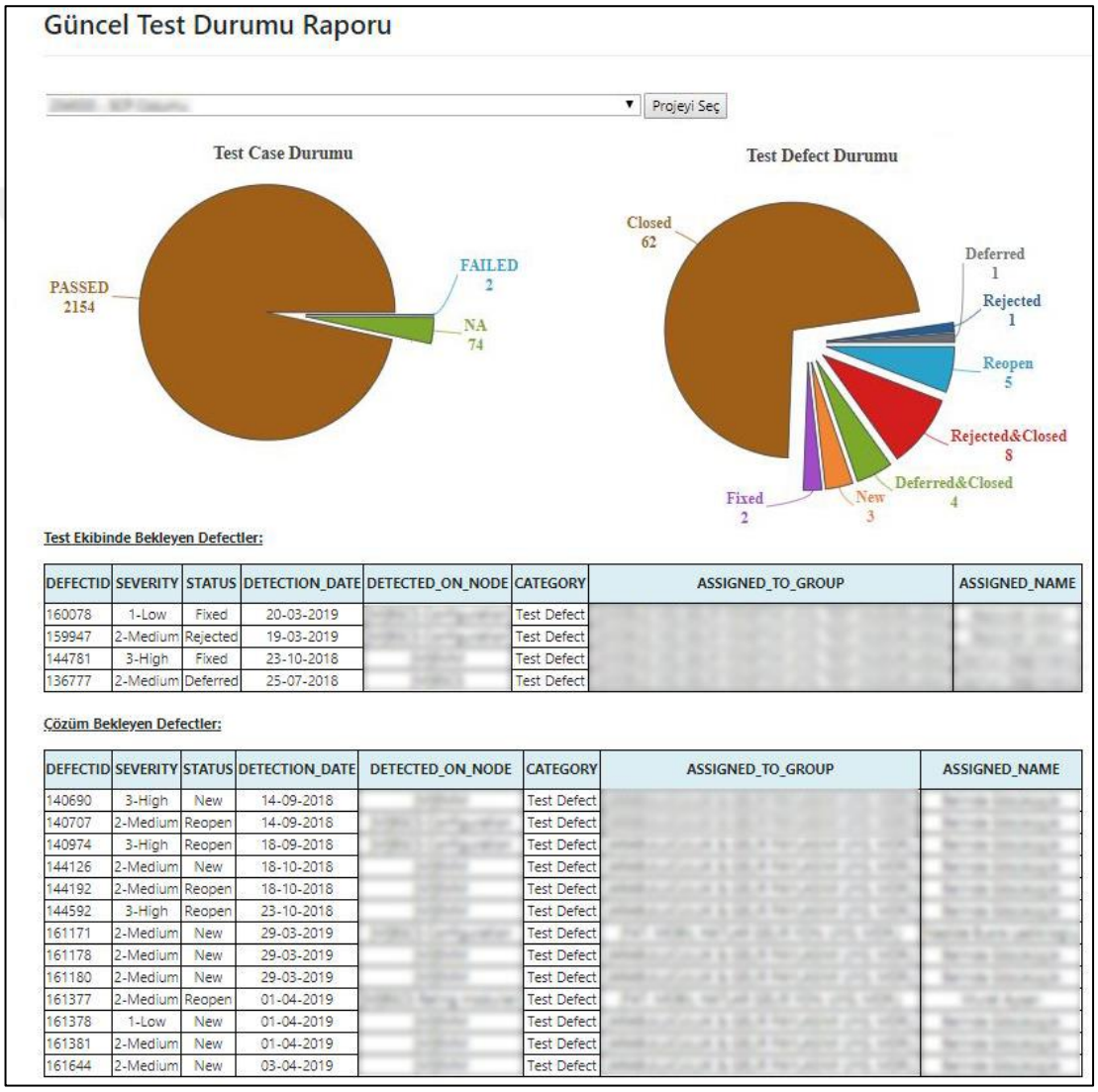
Proje seçimi yapıldıktan sonra projeye ait "Test Senaryo Durumu" ve "Test Hata Durumu" grafikleri oluşturulmaktadır. Şekil 20'de "Test Senaryo Durumu (Test Case Durumu)" grafiğinde test senaryolarının statü bazında güncel durumları gösterilmektedir. Şekil 20'de "Test Hata Durumu (Test Defect Durumu)" grafiğinde testler sırasında bulunan hatalara ait statü bazında hata durumları gösterilmektedir. Bu grafiklerin altında ise aksiyon bekleyen hatalara ait detaylar bulunmaktadır. Aksiyon bekleyen hatalar test ekibinde bekleyen ve çözüm ekibinde bekleyen hatalar olarak iki ayrı tablo halinde oluşturulmaktadır. Bu tablolarda hataların numaraları, şiddetleri, statüleri, hata açılma zamanı, hata kategorisi ve hatanın kimde beklediği gösterilmektedir.

Testler sırasında bulunan hataların çözülmesi ve test sürecinin zamanında tamamlanabilmesi için açılan hataların çözüm süreleri takip edilmektedir. Şirketler tarafından hataların çözümü için çalışılmış belirli çözüm süreleri bulunmaktadır. Hataların bu çözüm süreleri içerisinde yapıldığını gösteren grafik Şekil 21'de "Test Defect Çözüm Süreleri" tablosunda oluşturulmaktadır. Bulunan hatalar şiddetlerine göre gruplandırılmakta ve çözüm sürelerine uygun olarak çözülmeyen maddeler tabloda kırmızı renk ile gösterilmektedir. Çözüm süresine uygun olarak aksiyon alınan maddeler ise tabloda yeşil renk ile gösterilmektedir.

Test raporunda en önemli parametreler projede bulunan hatalardır. Bulunan hatalara ait hataların şiddetleri ve statüleri bazında sayılarını gösteren tablo Şekil 21'de "Defect Durumu" içerisinde oluşturulmakta ve toplam açılan hata sayısı görülmektedir.

Proje test senaryoları grafik dışında tablo olarak da Şekil 21'de "Fonksiyonel Test Case Durumu" tablosunda oluşturulmaktadır. Projeyle ilgili grupların hangisinde kaç tane hata beklediğini gösteren "Açık Defect özet Tablosu" oluşturulmaktadır. Proje isim bilgisi, kişi bilgisi, grup bilgisi ve hatanın bulunduğu birim bilgisi şekillerde gizli olarak gösterilmektedir.

Şekil 20: Test Durum Raporu Ekranı 1



Test Ekibinde Bekleyen Defectler:

DEFECTID	SEVERITY	STATUS	DETECTION_DATE	DETECTED_ON_NODE	CATEGORY	ASSIGNED_TO_GROUP	ASSIGNED_NAME
160078	1-Low	Fixed	20-03-2019		Test Defect		
159947	2-Medium	Rejected	19-03-2019		Test Defect		
144781	3-High	Fixed	23-10-2018		Test Defect		
136777	2-Medium	Deferred	25-07-2018		Test Defect		

Çözüm Bekleyen Defectler:

DEFECTID	SEVERITY	STATUS	DETECTION_DATE	DETECTED_ON_NODE	CATEGORY	ASSIGNED_TO_GROUP	ASSIGNED_NAME
140690	3-High	New	14-09-2018		Test Defect		
140707	2-Medium	Reopen	14-09-2018		Test Defect		
140974	3-High	Reopen	18-09-2018		Test Defect		
144126	2-Medium	New	18-10-2018		Test Defect		
144192	2-Medium	Reopen	18-10-2018		Test Defect		
144592	3-High	Reopen	23-10-2018		Test Defect		
161171	2-Medium	New	29-03-2019		Test Defect		
161178	2-Medium	New	29-03-2019		Test Defect		
161180	2-Medium	New	29-03-2019		Test Defect		
161377	2-Medium	Reopen	01-04-2019		Test Defect		
161378	1-Low	New	01-04-2019		Test Defect		
161381	2-Medium	New	01-04-2019		Test Defect		
161644	2-Medium	New	03-04-2019		Test Defect		

Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 21'de "Açık Defect Özet Tablosu" bölümünde bulunan grup bilgileri gizlenerek gösterilmiştir.

Şekil 21: Test Durum Raporu Ekranı 2

Test Defect Çözüm Süreleri:

Severity	Toplam Defect	Çözüm Süresi (gün)	Olması Gereken (gün)
1-Low	5	15.27	8
2-Medium	39	20.8	4
3-High	42	24.25	2

Defect Durumu :

	Closed	Fixed	New	Open	Data Req.	Rejected	Reopen	Rejected & Closed	Deferred	<total>
1-Low	4	1	0	0	0	0	0	0	0	5
2-Medium	29	0	2	0	0	1	3	3	1	39
3-High	29	1	1	0	0	0	2	5	0	42
Total	62	2	3	0	0	1	5	8	1	86

Fonksiyonel Test Case Durumu:

Failed	Blocked	N/A	No Run	Not Completed	Passed	<total>
2	0	74	0	0	2154	2230

Açik Defect Özet Tablosu :

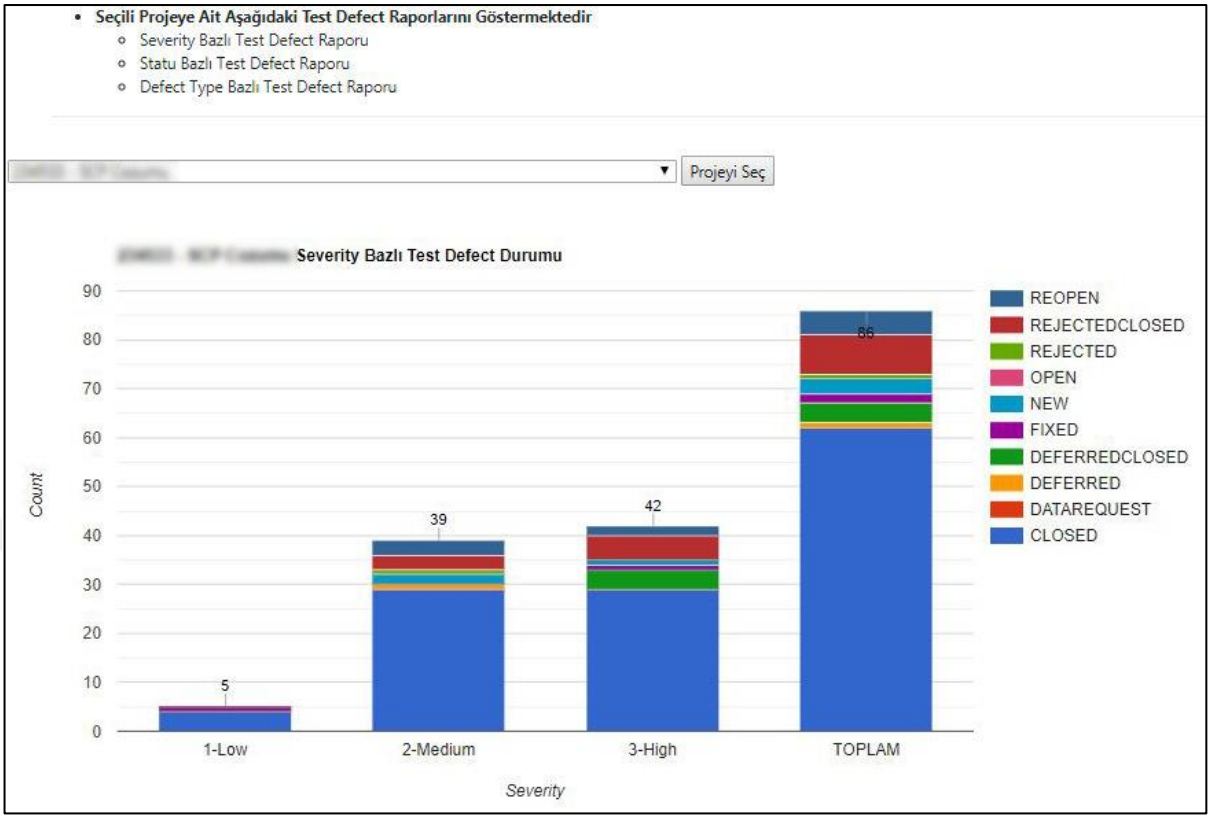
	1-Low	2-Medium	3-High	4-Urgent	<total>
[redacted]	0	2	0	0	2
[redacted]	1	7	3	0	11
[redacted]	0	1	0	0	1

Kaynak : Yazar Tarafından Derlenmiştir.

2.2.6 Proje Test Hata Grafikleri

"Proje Durumları" menüsü altında bulunan "Proje Test Hataları" sekmesinde seçili proje için bulunan hatalara ait grafikler oluşturulmaktadır. Hataların şiddetlerine göre gruplanmış hali, statüleri ve hata sayıları gösterilmektedir. Şekil 22'de seçili bir projeye ait şiddet ve statü bazında tüm test hata durum grafiği bulunmaktadır. Yüksek şiddette açılan hata sayısının fazla ya da az olmasına göre projenin durumu hakkında yöneticilere bilgi sağlanması amaçlanmıştır. Ayrıca bu grafikte açılan hataların kabul edilmeme oranları da görülebilmekte ve kabul edilmeyen hata sayılarının oranlarına göre test ekibi kalitesi hakkında bilgi sağlaması hedeflenmektedir.

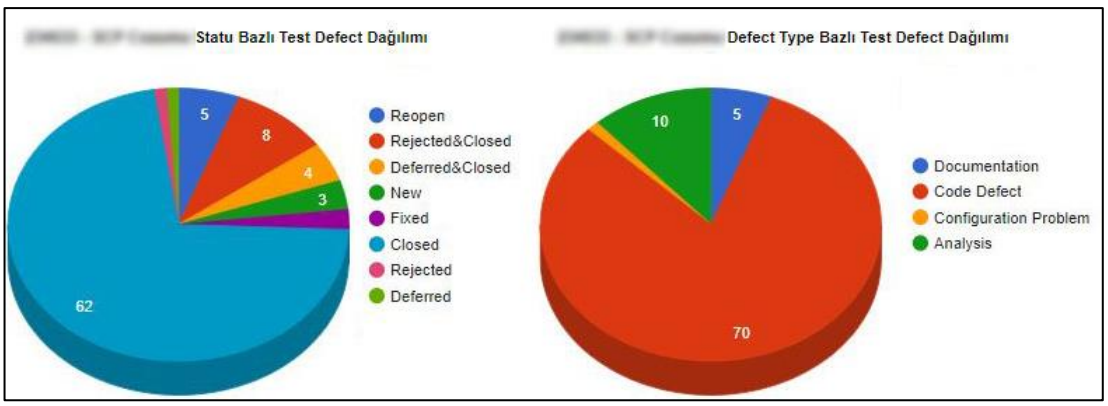
Şekil 22: Şiddet Bazlı Test Hata Raporu



Kaynak: Yazar Tarafından Derlenmiştir.

Şekil 23'de testte bulunan hataların statüsüne göre sayısal dağılım grafikleri ve hata tipine göre dağılım grafikleri oluşturulmuştur. Hata tipine göre dağılım grafiğinde projede kod, analiz, doküman vb. alanlarından hangisinde çok hata olduğu ve projenin hangi yönüne daha çok ağırlık verilmesi gerektiği sunulmaktadır.

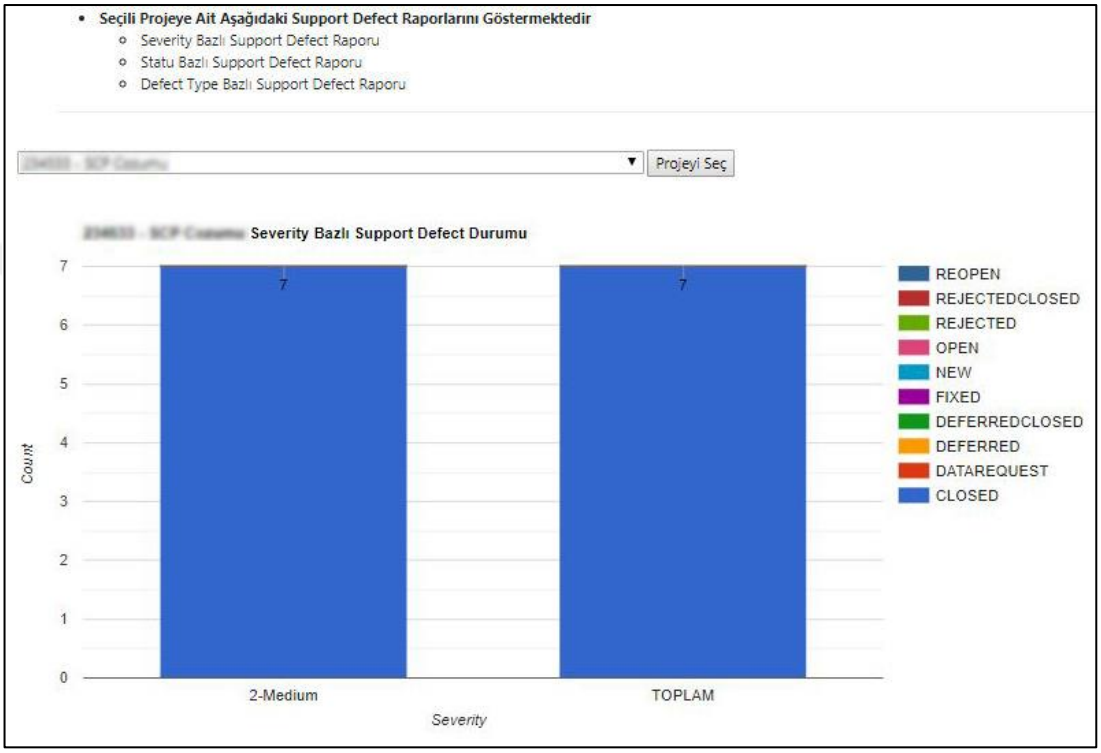
Şekil 23: Statü ve Hata Tipine Göre Test Hatası



Kaynak: Yazar Tarafından Derlenmiştir.

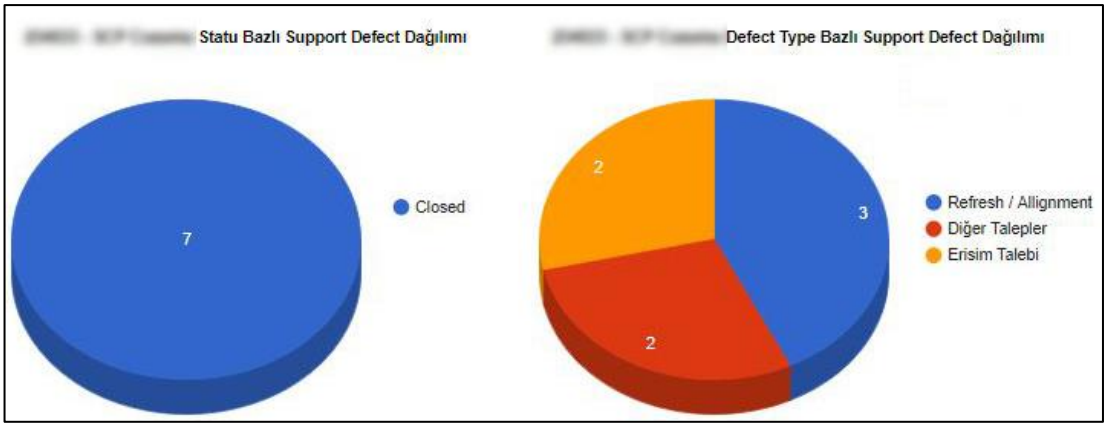
Test destek ekibine açılan hatalara ait raporlamalar Şekil 18'de bulunan "Proje Support Defectleri" sekmesi altında oluşturulmaktadır. Açılan support hatalarına ait şiddet, statü ve hata tipi bazında raporlamalar oluşturulmaktadır. Şekil 24'de seçili projeye ait açılan hataların şiddet bazında gruplanmış hali görülmektedir.

Şekil 24: Şiddet Bazlı Test Ortam Hata Raporu



Kaynak: Yazar Tarafından Derlenmiştir.

Şekil 25: Statü ve Hata Tipine Göre Test Ortam Hatası



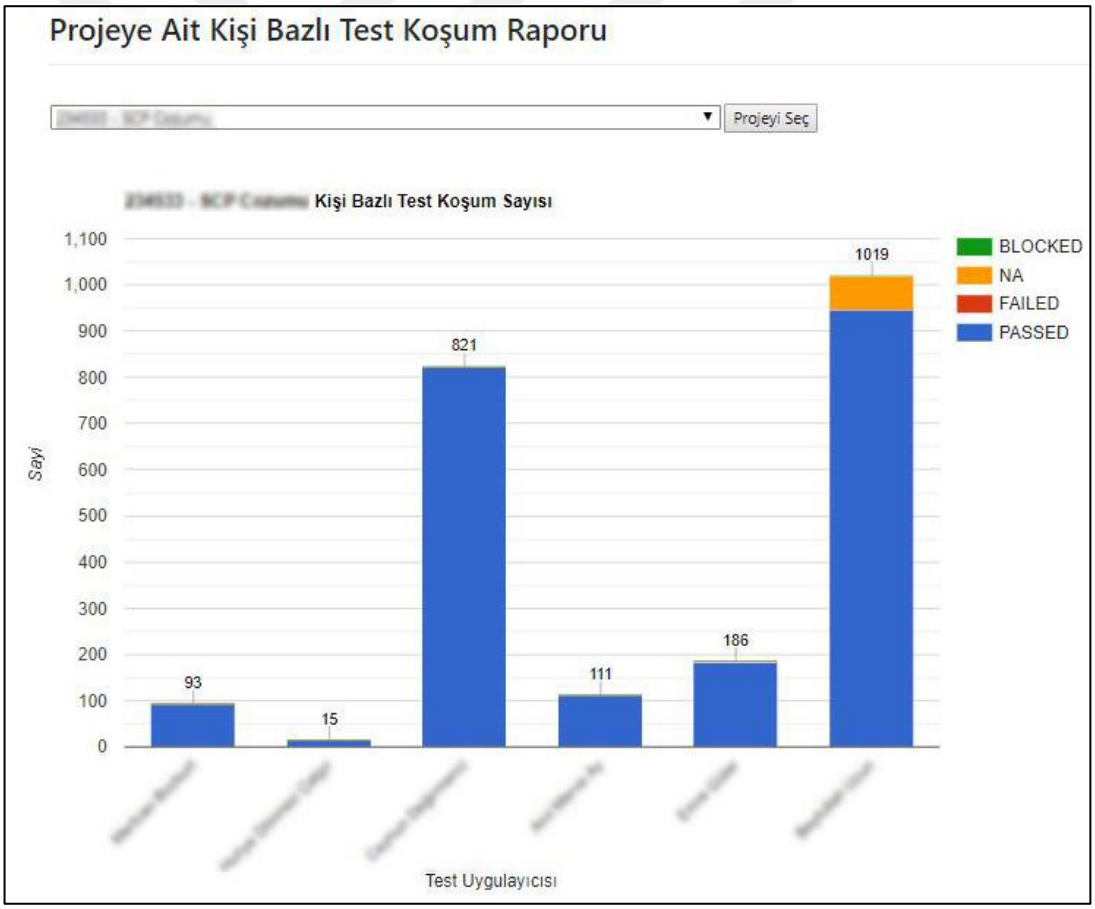
Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 25'de açılan support hatalarının statüsüne göre sayısal dağılım grafikleri ve hata tipine göre dağılım grafikleri oluşturulmuştur.

2.2.7 Proje Kişi Bazlı Test Koşumu

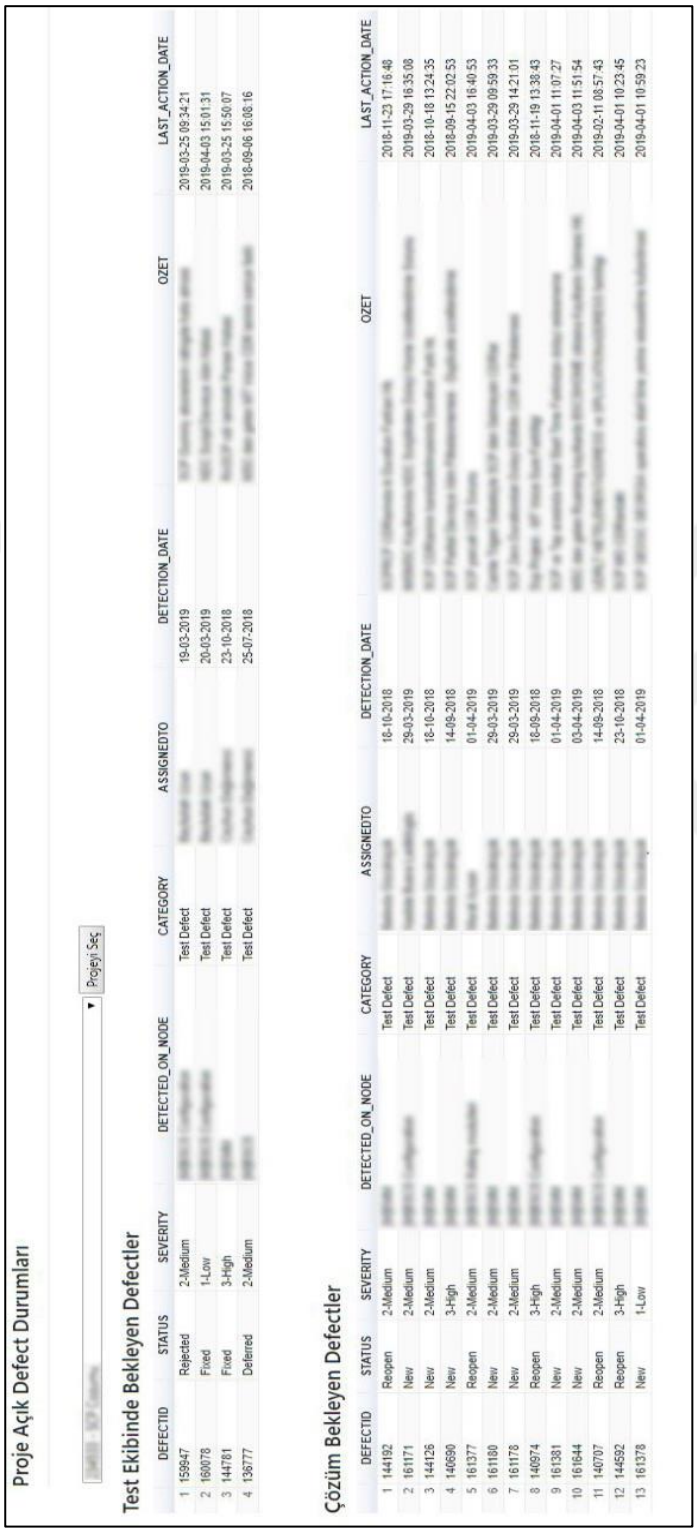
Şekil 26'da test senaryolarını gerçekleştiren test uygulayıcılarının proje kapsamında kaç adet test gerçekleştirdiğini gösteren grafik oluşturulmaktadır. Bu grafikte kişi bazında gerçekleştirilen test senaryolarının statüleri dikkate alınarak oluşturulmaktadır. Bu rapordaki amaç ise test uygulayıcılarının projede gerçekleştirdiği performans takibinin yapılmasıdır.

Şekil 26: Proje Kişi Bazlı Test Koşumu



Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 27: Açık Durumda Bulunan Hatalar



Proje Açık Defect Durumları

Projeyi Seç

Test Ekibinde Bekleyen Defectler

	DEFECTID	STATUS	SEVERITY	DETECTED_ON_NODE	CATEGORY	ASSIGNEDTO	DETECTION_DATE	OZET	LAST_ACTION_DATE
1	159947	Rejected	2-Medium		Test Defect		19-03-2019		2019-03-25 09:34:21
2	160078	Fixed	1-Low		Test Defect		20-03-2019		2019-04-03 15:01:31
3	144761	Fixed	3-High		Test Defect		23-10-2018		2019-03-25 15:50:07
4	136777	Deferred	2-Medium		Test Defect		25-07-2018		2018-09-06 16:08:16

Çözüm Bekleyen Defectler

	DEFECTID	STATUS	SEVERITY	DETECTED_ON_NODE	CATEGORY	ASSIGNEDTO	DETECTION_DATE	OZET	LAST_ACTION_DATE
1	144192	Reopen	2-Medium		Test Defect		18-10-2018		2018-11-23 17:16:48
2	161171	New	2-Medium		Test Defect		29-03-2019		2019-03-29 16:35:08
3	144126	New	2-Medium		Test Defect		18-10-2018		2018-10-18 13:24:35
4	140690	New	3-High		Test Defect		14-09-2018		2018-09-15 22:02:53
5	161377	Reopen	2-Medium		Test Defect		01-04-2019		2019-04-03 16:40:53
6	161180	New	2-Medium		Test Defect		29-03-2019		2019-03-29 09:59:33
7	161178	New	2-Medium		Test Defect		29-03-2019		2019-03-29 14:21:01
8	140974	Reopen	3-High		Test Defect		18-09-2018		2018-11-19 13:38:43
9	161381	New	2-Medium		Test Defect		01-04-2019		2019-04-01 11:07:27
10	161644	New	2-Medium		Test Defect		03-04-2019		2019-04-03 11:51:54
11	140707	Reopen	2-Medium		Test Defect		14-09-2018		2019-02-11 08:57:43
12	144592	Reopen	3-High		Test Defect		23-10-2018		2019-04-01 10:23:45
13	161378	New	1-Low		Test Defect		01-04-2019		2019-04-01 10:59:23

Kaynak: Yazar Tarafından Derlenmiştir.

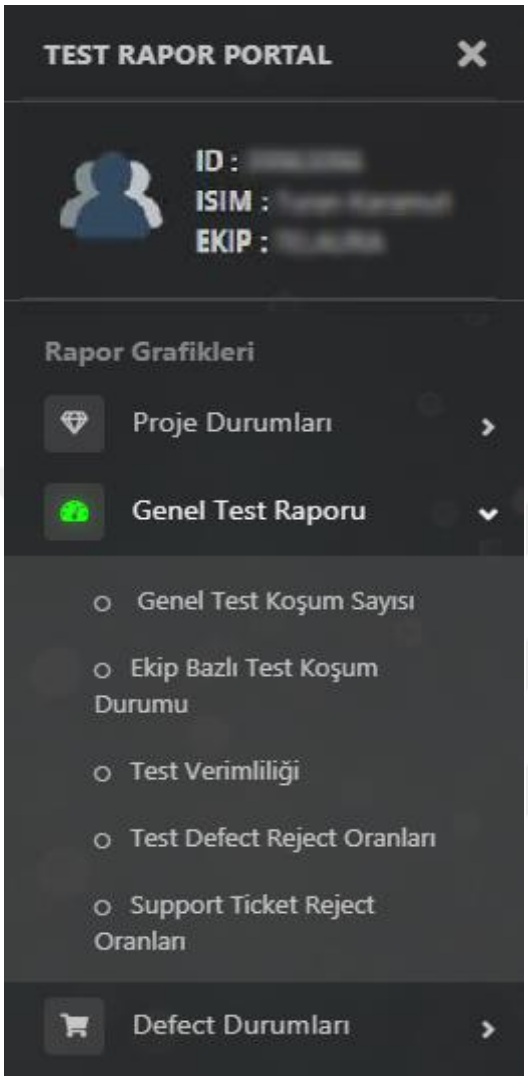
2.2.8 Açık Durumda Bulunan Hatalar

Şekil 18'de "Proje Durumları" menüsü altında projeye ait açık defetlerin gösterildiği menü bulunmaktadır. Bu menüden seçilecek olan projeyle ilgili hem test ekipleri hem de geliştirme ekiplerinde bekleyen hatalar detaylarıyla birlikte gösterilmektedir. Bu rapor sayesinde projenin ilgili kişileri gerekli aksiyonları zamanında görebilecektir. Projelerin test süreçlerinde takibi en çok gerekli olan hata durum raporları olduğu için anlık olarak değişiklerin görüntülenebileceği ve iş takibi açısından sık kullanılacak raporlama türlerinden biridir. Şekil 27'de projede açık durumda bulunan hatalara ait detaylar gösterilmektedir. Raporda hata numarası, hata statüsü, hata şiddeti, hatanın açılma tarihi, hatanın özeti ve hata ile ilgili yapılan son işlem tarihi bulunmaktadır.

2.2.9 Genel Test Raporu

Şekil 28'de "Genel Test Raporu" menüsü altında "Genel Test Koşum Sayısı", "Ekip Bazlı Test Koşum Durumu", "Test Verimliliği", "Test Defect Reject (Kabul Edilmeme) Oranları" ve "Support Defect Reject Oranları" alt menüleri bulunmaktadır.

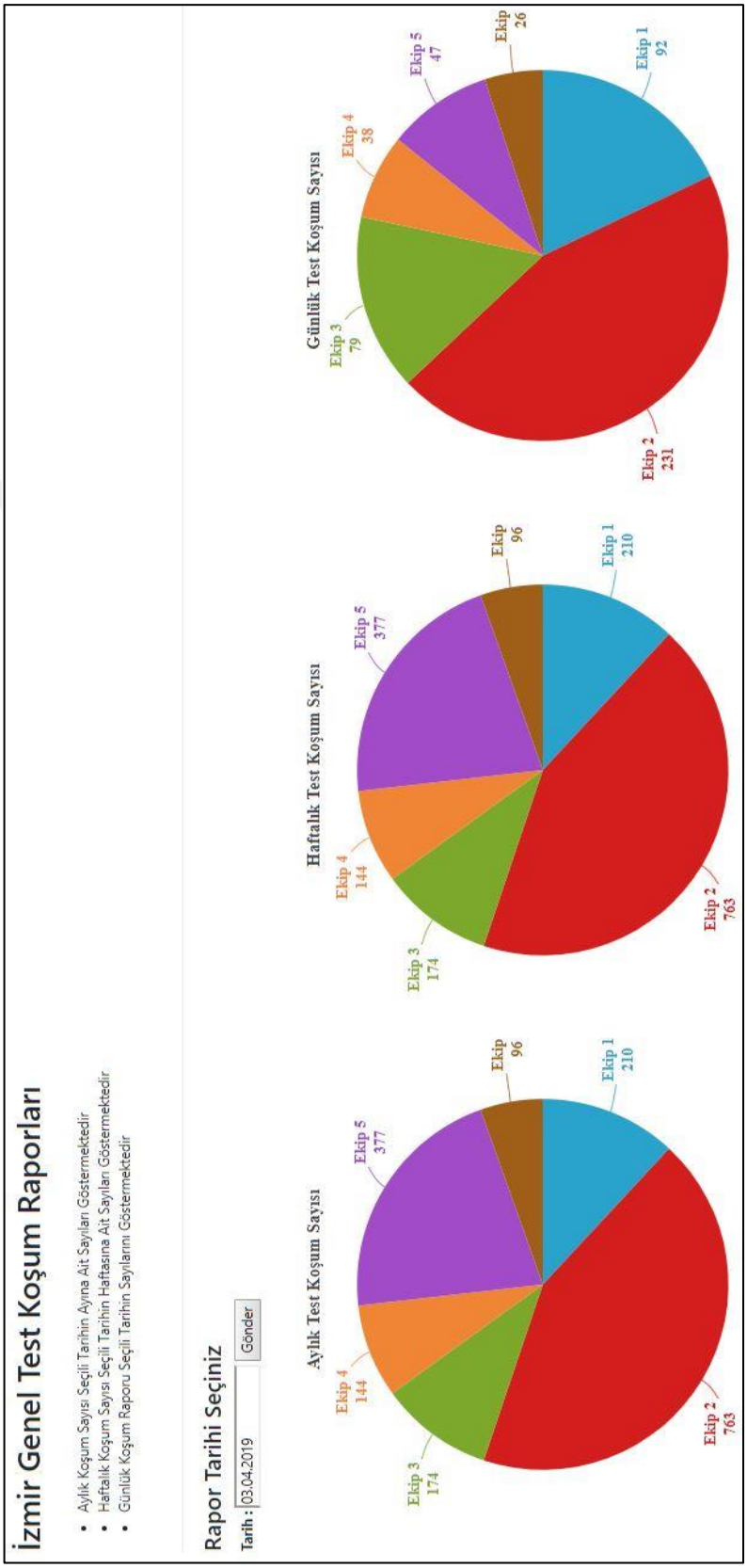
Şekil 28: Genel Test Rapor Menüsü



Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 29'da genel test raporu menüsü altında aylık, haftalık ve günlük test senaryo gerçekleştirme sayılarını ekip bazlı olarak sunan grafik bulunmaktadır. Bu grafik sayesinde test ekiplerinin performans durumu izlenebilmektedir. Genel test raporunda günlük grafik seçili tarihin bulunduğu güne ait test senaryo gerçekleştirme sayılarını gösterir. Haftalık grafik ise seçili tarihin bulunduğu haftaya ait grafiği gösterir. Başlangıç tarihi olarak pazartesi günü bitiş tarihi olarak pazar günü alınarak rapor oluşturulmaktadır. Aylık grafik ise seçili tarihin bulunduğu aya ait grafiği gösterir. Başlangıç tarihi olarak seçili ayın 1. günü, bitiş tarihi olarak ise ayın son günü alınarak grafik oluşturulmaktadır.

Şekil 29: Genel Test Raporu

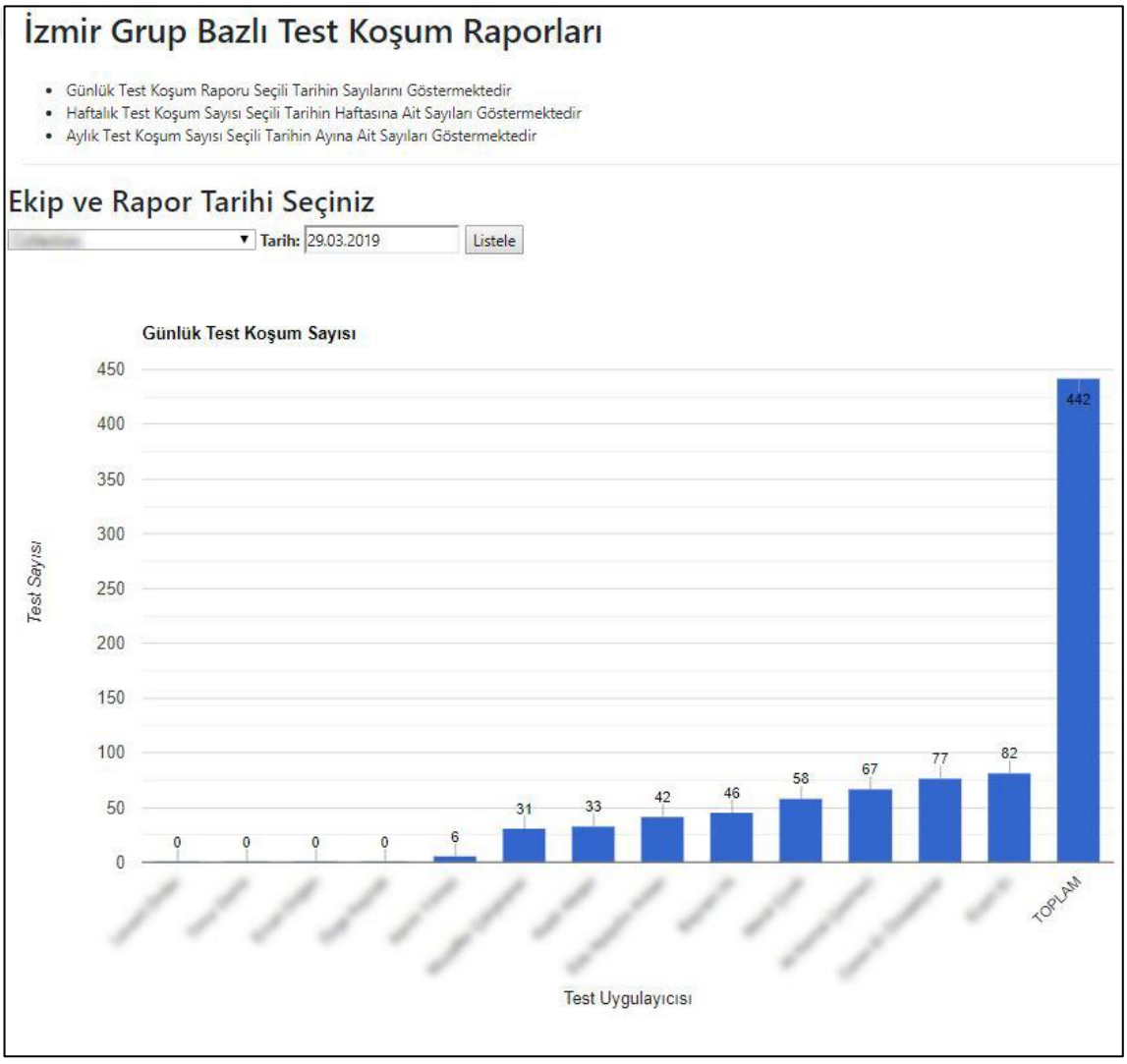


Kaynak: Yazar Tarafından Derlenmiştir.

Ekiplerde test senaryo gerçekleştirme sayılarının detayları kişi bazında da görüntülenebilmektedir. Rapor oluşturulmadan önce ekip ve tarih bilgisi seçimi yapılması gerekmektedir. Ekip bilgileri veritabından çekilmekte ve açılan kutu içerisinde listelenmektedir. Seçilecek olan ekibe ait günlük, haftalık ve aylık test koşum raporu oluşturulmaktadır.

Kişi bazında günlük test senaryo gerçekleştirme sayıları Şekil 30'da bulunan grafik ile gösterilmektedir. Günlük olarak yapılan test sayılarının takibi kolaylıkla yapılabilmekte ve varsa hedeflerin gerçekleştirme durumu izlenebilmektedir.

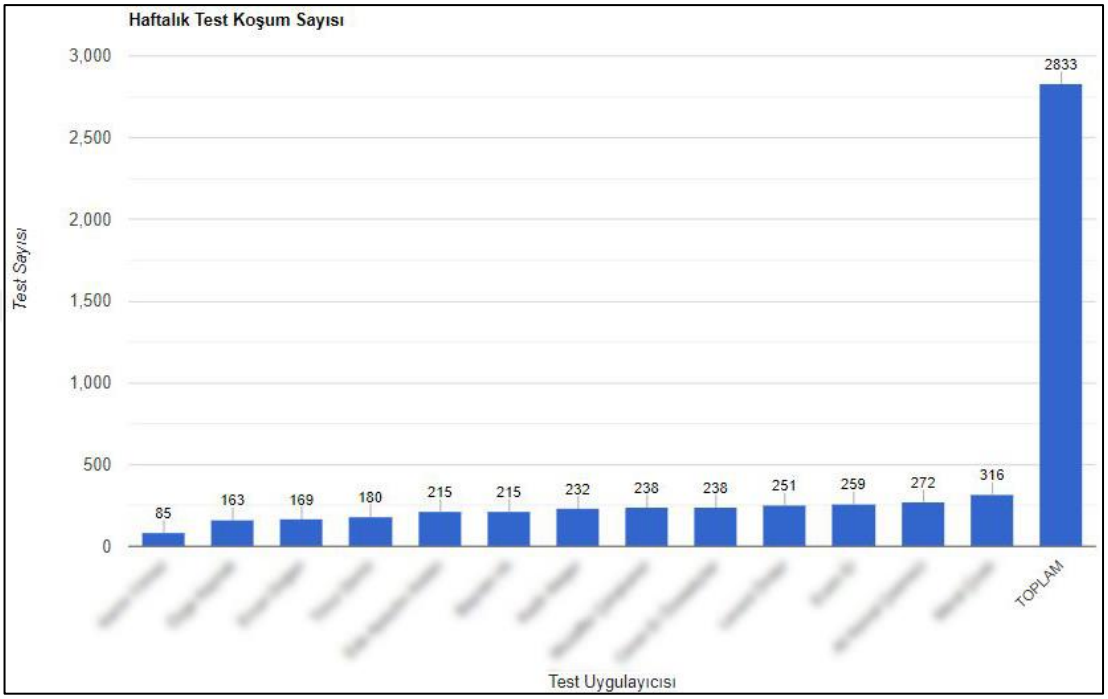
Şekil 30: Ekip Günlük Test Koşumu



Kaynak: Yazar Tarafından Derlenmiştir.

Aynı sayfa içerisinde seçili günün haftasına ait test senaryo gerçekleştirme sayıları Şekil 31'de gösterilmektedir. Haftalık performans takibinin yapılması bu grafik ile gerçekleştirilebilmektedir.

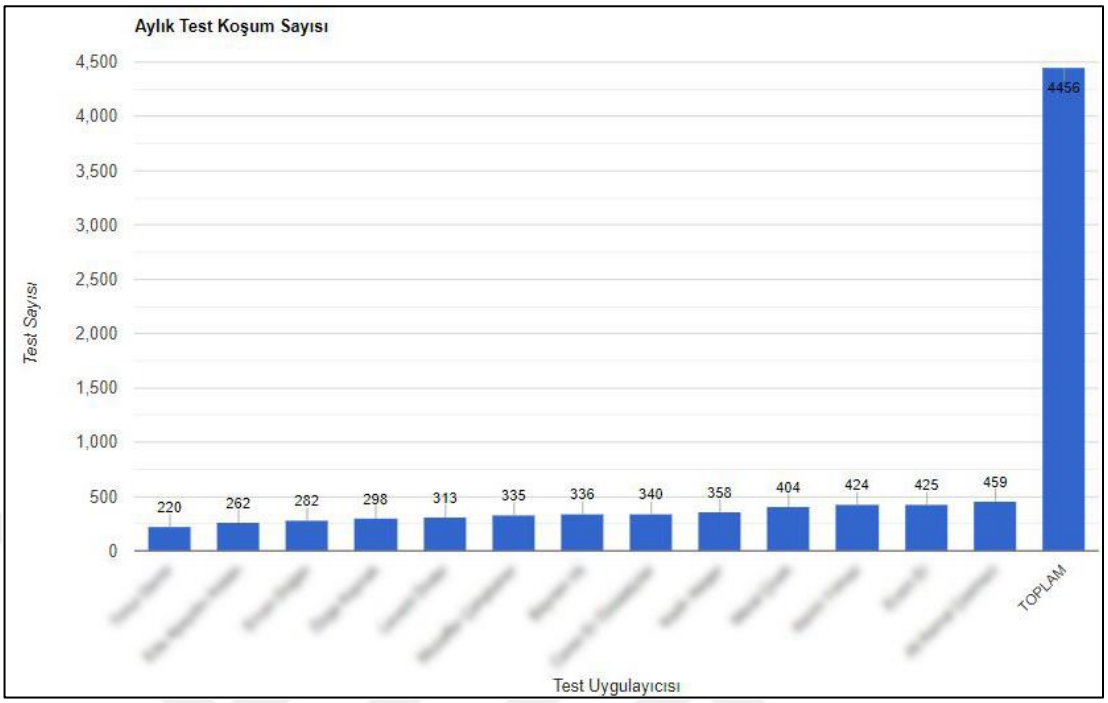
Şekil 31: Ekip Haftalık Test Koşumu



Kaynak : Yazar Tarafından Derlenmiştir.

Günlük ve haftalık test senaryo sayılarının dışında aylık test senaryo gerçekleştirme grafiği de rapor olarak oluşturulmaktadır. Şekil 32'de bulunan aylık raporda seçili tarihin bulunduğu aya ait kişi bazlı toplam test koşum sayısı gösterilir. Başlangıç tarihi olarak seçili ayın 1. günü, bitiş tarihi olarak ise ayın son günü alınarak grafik oluşturulmaktadır.

Şekil 32: Ekip Aylık Test Koşumu



Kaynak : Yazar Tarafından Derlenmiştir.

Genel test raporu menüsü altında ayrıca test ekipleri tarafından açılan defectlerin analiz raporları da oluşturulmaktadır. Bu analizler test hataları ve ortam hataları olarak değerlendirilmektedir.

Şekil 33 ve Şekil 34'de test ekipleri tarafından bulunan test ve ortam hatalarının kabul edilmeme oranları gösterilmektedir. Kabul edilmeme oran hesabı; kabul edilmeyen hataların toplam açılan hataya bölümünden çıkan oran sonucudur. Bu oran sayesinde test ekiplerinin açtığı hataların ne kadarının gerçekten hata olarak kabul edildiği görülmekte ve test ekibinin hata açma kalitesinin ölçümü yapılmaktadır. Yüksek oranda kabul edilmeyen hata bulunmasında o aya ait maliyetlerin arttığı görülebilir. Kabul edilmeyen her hatada, test ekiplerinin ve çözüm ekiplerinin boşa harcadığı efor görülmektedir. Çıkan rapor sonucuna göre yöneticiler tarafından, kabul edilmeyen hata oranını düşürecek aksiyon alınması gerekecek ve maliyet, zaman tasarrufu yapılmış olacaktır. Grafikte mavi ile gösterilen çubuklar test ekibi tarafından o ay içerisinde bulunan ve hata olarak kabul edilen hata sayısını göstermektedir. Turuncu ile gösterilen çubuklar ise o ay içerisinde kabul edilmeyen hata sayılarını göstermektedir.

Şekil 33: Test Hata Kabul Edilmeme Oranı

Kaynak : Yazar Tarafından Derlenmiştir.

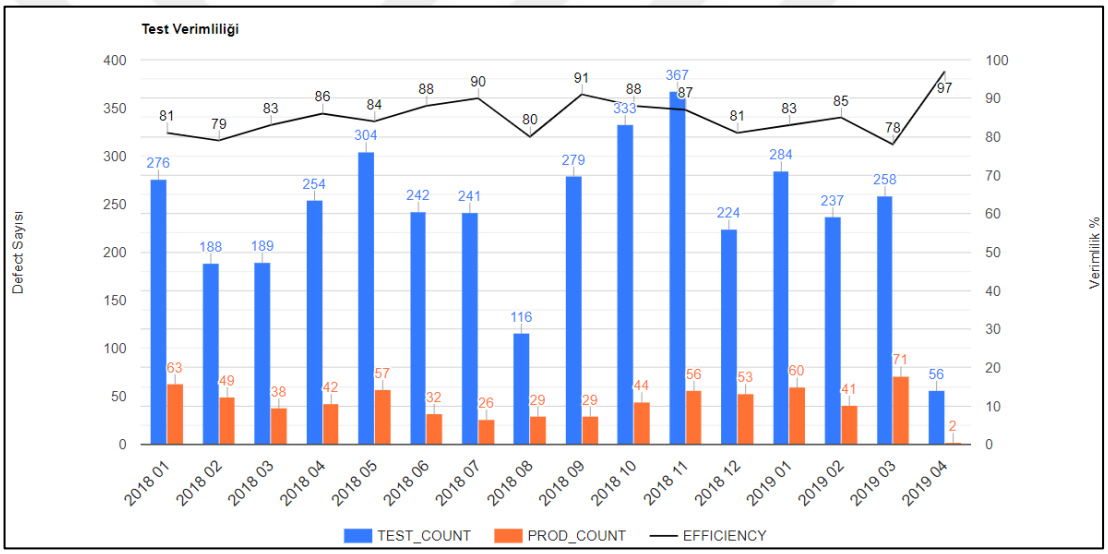
Şekil 34: Test Ortam Hata Kabul Edilmeme Oranı

Kaynak : Yazar Tarafından Derlenmiştir.

Proje sonunda ortaya çıkan ürün müşteriye sunulduktan sonra oluşan hataların fazla olması test verimliliğini düşürmektedir. Şekil 35'de portal tarafından oluşturulan test verimliliği grafiği bulunmaktadır. Test verimliliği aşağıdaki formüle göre hesaplanmıştır. Grafikte mavi ile gösterilen çubuklar test ekibi tarafından o ay içerisinde bulunan test hatalarını göstermektedir. Turuncu çubuklar ise piyasada bulunan ürün içerisinde bulunan hata sayılarını göstermektedir.

$$Test\ Verimliliği = \frac{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı}{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı + Canlı\ Sistemde\ Bulunan\ Hata\ Sayısı} \times 100$$

Şekil 35: Test Verimliliği Grafiği

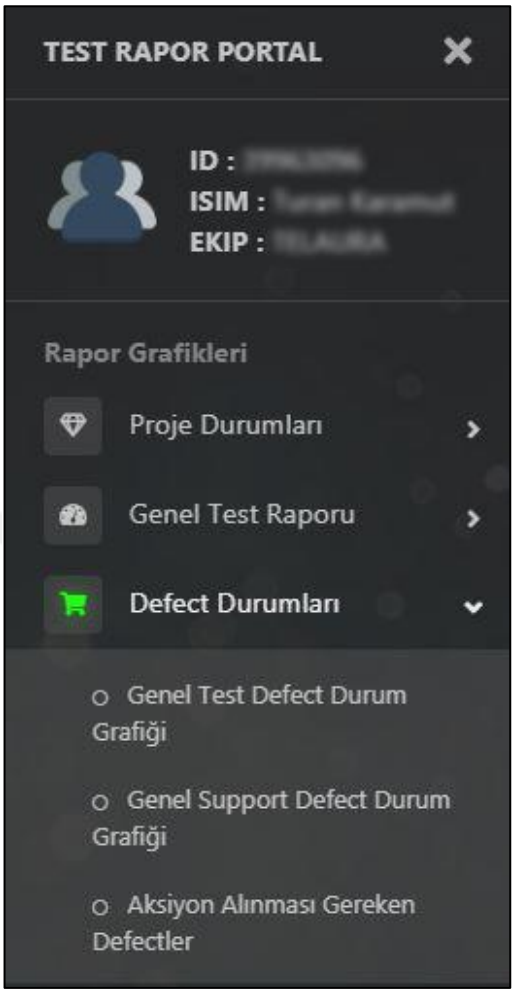


Kaynak: Yazar Tarafından Derlenmiştir.

2.2.10 Hata Durumları

Hata durumları menüsü altında 3 alt menü bulunmaktadır. Bunlar; "Genel Test Defect Durum Grafiği", "Genel Test Ortam Defect Durum Grafiği" ve "Aksiyon Alınması Gereken Hatalar" alt menüleridir. Şekil 36'da menünün görüntüsü bulunmaktadır.

Şekil 36: Hata Durumları Menüsü

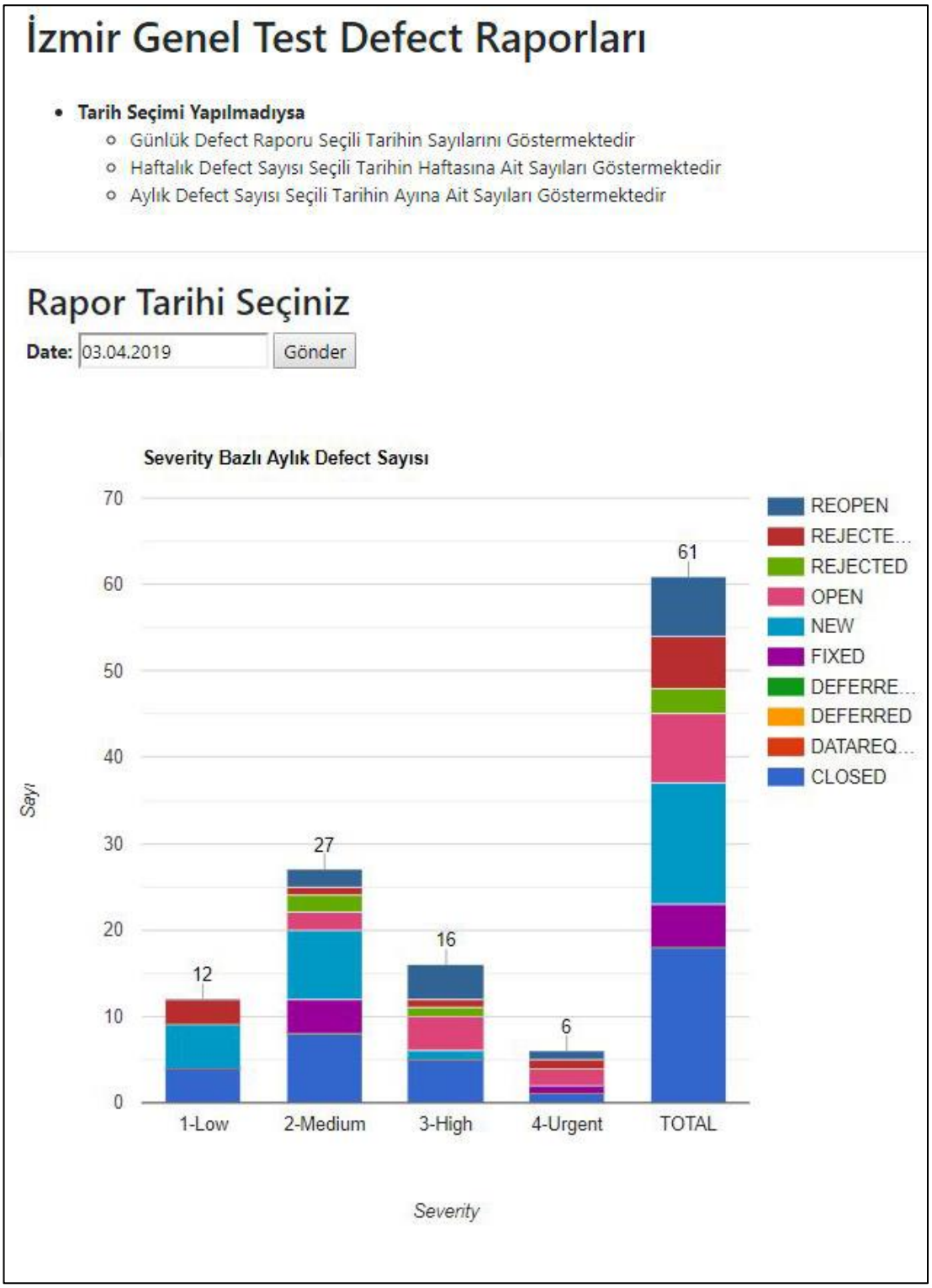


Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 37'de seçili tarihe ait tüm test ekipleri tarafından bulunan hataların aylık, haftalık ve günlük olarak analizleri oluşturulmaktadır. Günlük gösterilen raporda seçili tarihin gününe ait rapor, haftalık raporda seçili tarihin haftasına ait rapor ve aylık raporda Şekil 37'de görüldüğü gibi seçili tarihin ayına ait şiddet ve statü bazında gruplanmış olarak hata sayıları gösterilmektedir. Şekil 38'de açılan hataların statülerine göre grafiği bulunmaktadır. Bu grafikler ile yazılımın test aşamasında kalitesi izlenebilmektedir.

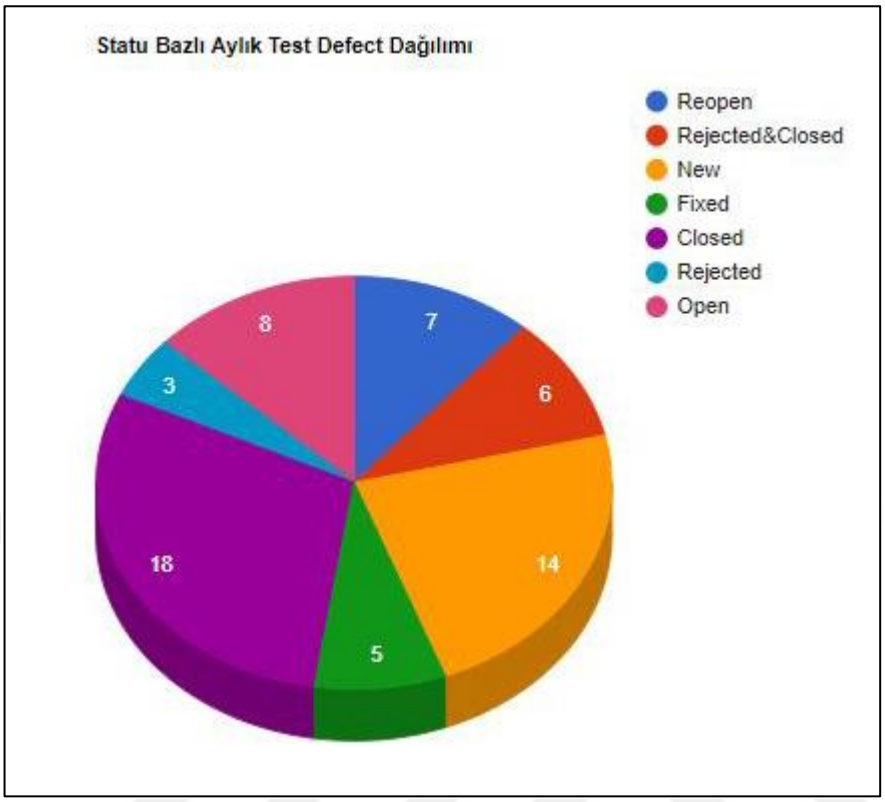
Tüm ekiplerin üzerinde bulunan hataların raporu ise Şekil 39'da oluşturulmaktadır. Aksiyon alınması gereken hatalar listesinin düzenli olarak takibi ile ekipler üzerinde bulunan hataların tekrar testlerinin gerçekleştirilmesi ve ekipler üzerinde bekleme süresi azaltılmış olacaktır.

Şekil 37: Genel Test Hata Durum Raporu



Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 38: Statü Bazlı Test Hata Sayısı Dağılımı



Kaynak : Yazar Tarafından Derlenmiştir.

Şekil 39: Aksiyon Alınması Gereken Hatalar

İzmir Test Ekibi Aksiyon Alınması Gereken Defectler

- **Son 3 Gündür Aksiyon Alınmamış Defectleri Göstermektedir.**

	DEFECTID	STATUS	SEVERITY	ASSIGNEDTO	DETECTION_DATE	LAST_ACTION_DATE	GRUP
1	158022	Fixed	1-Low	[redacted]	27/02/2019	2019-03-26 09:57:54	[redacted]
2	159947	Rejected	2-Medium	[redacted]	19/03/2019	2019-03-25 09:34:21	[redacted]
3	144781	Fixed	3-High	[redacted]	23/10/2018	2019-03-25 15:50:07	[redacted]
4	159763	Fixed	1-Low	[redacted]	18/03/2019	2019-03-25 13:34:54	[redacted]
5	136777	Deferred	2-Medium	[redacted]	25/07/2018	2018-09-06 16:08:16	[redacted]
6	159754	Fixed	1-Low	[redacted]	18/03/2019	2019-03-25 13:26:40	[redacted]
7	161156	Fixed	1-Low	[redacted]	29/03/2019	2019-03-29 16:42:53	[redacted]
8	160802	Deferred	2-Medium	[redacted]	26/03/2019	2019-03-28 15:38:18	[redacted]
9	160605	Deferred	2-Medium	[redacted]	25/03/2019	2019-03-28 15:38:54	[redacted]
10	159229	Fixed	2-Medium	[redacted]	12/03/2019	2019-03-29 09:22:27	[redacted]
11	160635	Fixed	3-High	[redacted]	25/03/2019	2019-03-27 10:01:40	[redacted]
12	161279	Fixed	1-Low	[redacted]	29/03/2019	2019-03-29 14:09:35	[redacted]
13	160371	Fixed	2-Medium	[redacted]	22/03/2019	2019-03-22 14:01:00	[redacted]
14	161037	Rejected	2-Medium	[redacted]	28/03/2019	2019-03-28 11:24:19	[redacted]
15	148341	Fixed	2-Medium	[redacted]	22/11/2018	2018-11-22 11:21:08	[redacted]
16	139166	Rejected	2-Medium	[redacted]	29/08/2018	2018-11-15 08:23:32	[redacted]
17	160659	Deferred	3-High	[redacted]	25/03/2019	2019-03-26 10:39:49	[redacted]
18	160109	Deferred	3-High	[redacted]	20/03/2019	2019-03-25 18:45:35	[redacted]
19	160478	Deferred	3-High	[redacted]	22/03/2019	2019-03-25 18:48:22	[redacted]
20	160059	Deferred	3-High	[redacted]	20/03/2019	2019-03-21 15:05:38	[redacted]

Kaynak : Yazar Tarafından Derlenmiştir.

SONUÇ

Yaşamamızın her alanında yazılıma olan ihtiyaç sürekli artmakta ve bu ihtiyaca paralel olarak yazılım proje sayıları da artmaktadır. Kaliteli yazılımların oluşturulabilmesi için projelerin iyi bir test sürecinden geçirilmesi gerekmektedir. Test sürecinden geçirilmeyen projelerin oluşturacağı risk ve sorunlar yüzünden piyasa içerinde kaybolacağı görülmekte ve şirketler açısından saygınlık, para ve zaman kaybına sebep olacaktır. Projelerin iyi bir test sürecinden geçirilmesi ve en az hata ile müşterilere sunulması için proje takvimi içerisinde oluşacak risklere ve alınması gerekecek önlemlere yol gösterecek zengin raporlamaların olması gerekmektedir.

Test süreçleri ve testlerde oluşacak hata yönetimleri için piyasa içerisinde bulunan uygulamaların yeterli grafiksel arayüzleri bulunmamakta ve olan grafiksel arayüzlerin oluşturulması teknik bilgi ve uzmanlık gerektirmektedir. Ayrıca test sürecinde kullanılacak olan raporların oluşturulmasında büyük zaman kayıpları yaşanmakta ve bilişim sektörü içerisinde bulunan teknik ekipler dışındakiler tarafından raporların oluşturulması ve anlaşılması zor olmaktadır.

Geliştirilen uygulama ile bütün proje ekipleri ve yöneticiler tarafından kolaylıkla erişilebilir ve anlaşılabilir raporlamaların oluşturulması sağlanmıştır. Bu uygulama ile karar vericilerin zaman kaybetmeden anlık ve gerçek zamanlı olarak raporlara ulaşılması sağlanmaktadır. Yazılım sürecinde yazılımın kalitesinin görüldüğü en önemli aşama olan yazılım testinin çıktısı yazılımın durumu hakkında bilgi sağlayan raporlardır. Yazılım test sürecinin en iyi şekilde görüntülenebilir hale gelmesi ve yazılımın kalitesi hakkında bilgi sağlayan raporlamalar için zaman kaybı yaşanmadan sadece testin kalitesi için çalışan testçilere daha fazla zaman ayırmak için raporlama sistemi oluşturulmuştur.

Bu tez çalışmasında bilişim problemleri karşısında, bilişim sistemleri kullanılarak yazılım test süreçleri ve raporlama tipleri analizi yapılmış ve problem çözümü için karar gereksinimleri araştırılıp, geliştirilen uygulamanın yazılım testleri karar verme süreçlerinde etkili rol alması sağlanmıştır.

Tez çalışmasının devamı olarak aşağıdaki maddelerin yapılmasıyla çalışmanın daha yararlı bir hale gelebileceğine inanılmaktadır.

a. Yazılım test raporlama sistemi uygulamasının şirketlere sağlayacağı yararın, çalışanların performanslarındaki artışın ve proje kalitelerindeki artışın analiz edilmesi.
b. Geliştirilen uygulamanın hali hazırda var olan ve yeni geliştirilecek yazılım test süreç uygulamalarının hepsiyle uyumlu çalışacak bir raporlama sistemine dönüştürülmesi.
c. Çalışma kapsamında oluşturulan raporlamaların rol bazında gruplandırılması ve hangi seviyedeki yöneticilere sunulması gerekeceğinin analizinin yapılması.
d. Çalışan performans değerlendirmesinin yapılacağı, kişi bazında yıllık veya dönemlik olarak sayısal verilerle çalışan performans değerlendirmesinin yapılacağı bir uygulama geliştirilmesi.

KAYNAKÇA

Bourque, P. ve Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.

Chen, J. C. ve Huang, S. J. (2009). An Empirical Analysis of the Impact of Software Development Problem Factors on Software Maintainability. *Journal of Systems and Software*, *82*(6): 981-992.

Chen, L., Wang, Z., Xu, L., Lu, H. ve Xu, B. (2010, June). Test case prioritization for web service regression testing. In *2010 Fifth IEEE International Symposium on Service Oriented System Engineering* (pp. 173-178). IEEE.

Chelliah, D. ve Ross, D. (2012). *What is Software Testing?*, (2012,ss.17, 18, 19. What is Software Testing?: ISTQB Foundation Companion and Study Guide

Denver Airport Baggage Handling System Case Study. (2008). Calleam Consulting. https://www5.in.tum.de/persons/huckle/DIABaggage.pdf, (30.04.2018)

Friedman, M. A. ve Voas, J. M. (1995). *Software Assessment: Reliability, Safety, Testability*. New York: John Wiley & Sons, Inc.

Graham, D., Van Veenendaal, E. ve Evans, I. (2008). *Foundations of software testing: ISTQB certification*. Cengage Learning EMEA.

GURU99, (2018). *Introduction to HP ALM(Quality Center)*, https://www.guru99.com/hp-alm-introduction.html, (03.09.2018)

GURU99, (2018). *What is Regression Testing?*. http://www.guru99.com/regression-testing.html, (28.06.2018)

Hambling, B., Morgan, P., Samaroo, A., Thompson, G. ve Williams, P. (2008). *Software Testing: An ISEB Foundation, Revised*. British Computer Society.

Hambling, B., Thompson, G., Samaroo, A., Williams, P. ve Morgan, P. (2010). *Software testing: an ISTQB-ISEB foundation guide*. British Informatics Society Limited.

Johnson, P. (2012). *Mariner 1's $135 Million Software Bug.* https://www.itworld.com/article/2717299/mariner-1-s--135-million-software-bug.html, (06.05.2018)

Jones, C. (2008). *Applied Software Measurement: Global Analysis of Productivity and Quality*. McGraw-Hill Education Group.

Lazic, L. ve Mastorakis, N. (2008). Cost effective software test metrics. *WSEAS Transactions on Computers*, 7(6): 599-619.

Lions, J. L. (1996). *ARIANE 5, Flight 501 Failure*. Paris, Temmuz 1996 http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html, (04.05.2018)

Mogyorodi, G. (2001). Requirements-based testing: an overview. In *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39* (pp. 286-295). IEEE.

Naik, K., Tripathy, P., Ammann, P. ve Offutt, J. (2009). Software Testing and Quality Assurance: Theory and Practice.

Oracle Call Interface, (2018). *Oracle Call Interface* (OCI). https://www.oracle.com/database/technologies/appdev/oci.html, (12.02.2019)

Patton, R. (2006). *Software testing*. Pearson Education India.

Programmable Web, *Google Visualization API* https://www.programm-ableweb.com/api/google-visualization, (15.02.2019).

Rajal, J. S. ve Sharma, S. (2015). A review on various techniques for regression testing and test case prioritization. *International Journal of Computer Applications*, *116*(16).

Rex, B. (2002). Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing.

Sharma, L. (2016). *Test Process in Software Testing*. https://www.toolsqa.com/software-testing/test-process-in-software-testing/, (28.05.2018).

Smart Bear, (2018). QAComplete, https://qacomplete.com/about/, (15.10.2018).

Software Testing Fundamentals, Unit Testing http://softwaretestingfundamentals.com/unit-testing/, (19.06.2018).

Software Testing Fundementals (2018). *Defect Priority*. http://softwaretestingfundamentals.com/defect-priority/, (18.05.2018).

Software Testing Fundementals. Defect Severity http://softwaretestingfundamentals.com/defect-severity/, (18.05.2018).

Software Testing Tools Guide, (2018). *HP ALM*, http://www.testingtoolsguide.net/tools/hp-alm/,(15.10.2018).

Test Lodge, (2018). *Getting Started With TestLodge* https://help.testlodge.com/hc/en-us/articles/236206547-Getting-Started-With-TestLodge, (26.10.2018).

Test Rail, (2018). Modern test management for your team, http://www.gurock.com/testrail/, (15.10.2018).

TRY QA, (2018). *What Are Test* ClosureActivities*? Evaluating Exit Criteria and Re porting*.http://istqbexamcertification.com/what-are-test-closure-activities-evaluating-exit-criteria-and-reporting/, (17.06.2018).

User Snap, User Acceptance Testing – How To Do It Right! https://usersnap.com/blog/user-acceptance-testing-right/, (10.01.2019).

Watkins, J. ve Mills, S. (2010). *Testing IT: an off-the-Shelf Software Testing Process*. Cambridge: Cambridge University Press.

EKLER

Ek 1: PHP OCI Örnek Kullanımı

```
<?php
$conn1 = oci_connect('username', 'password','Veritabanı TNS','AL32UTF8');
$stid_proje = oci_parse($conn1, 'Raporlama için oluşturulacak SQL komutu');

oci_execute($stid_proje);
?>

<main class="page-content">
<div>
        <h4>Proje Açık Defect Durumları</h4>
        <hr />
</div>
<br />

<form  action ="index.php?modul=proje_acik_defectler"method="post">
        <select name="projects">
                <?php
                        while ($row = oci_fetch_array($stid_proje, OCI_RETURN_NULLS+OCI_ASSOC))
                        {?>
                <option
                        <?php if($row['RCYC_NAME'] == $_POST["projects"]) {?> selected="selected" <?php } ;
        ?> value="<?php echo $row['RCYC_NAME'];?>"> <?php echo $row['RCYC_NAME'];?>
                        </option>
                        <?php
                        }
                ?>
```

```
        </select>
        <input type="submit" name="submit" value="Projeyi Seç"/>
        <input type = "hidden" name ="process" value="liste"/>
</form>
</main>
```

Ek 2: Google Görselleştirme API Örnek Kullanım

```
<?php
$stid_proje = oci_parse($conn1, 'SQL Sorgusu: projelerin ismini çeken SQL Sorgusu');
oci_execute($stid_proje);
?>

<br />
<main class="page-content">
<div>
        <h4>Proje Açık Defect Durumları</h4>
        <hr />
</div>
<br />
<form action ="index.php?modul=proje_acik_defectler"method="post"><!--Projede Aksiyon Alınması Gereken Hatalar-->
        <select name="projects">
                <?php
                        while ($row = oci_fetch_array($stid_proje, OCI_RETURN_NULLS+OCI_ASSOC))
                        {?>
                <option
                        <?php if($row['RCYC_NAME'] == $_POST["projects"]) {?> selected="selected" <?php } ;?> value="<?php echo $row['RCYC_NAME'];?>"> <?php echo $row['RCYC_NAME'];?>
                        </option>
                        <?php
                        }
                ?>
                </select>
                <input type="submit" name="submit" value="Projeyi Seç"/>
                <input type = "hidden" name ="process" value="liste"/>
```

```
</form>
<?php
    if(isset($_POST['process']) ? $_POST['process'] : false ==
"liste")
    {
 ?>
<?php
    $selected_val = isset($_POST["projects"]) ?
$_POST["projects"] : "  ";
    $selected_val =substr($selected_val ,0,70);
?>
<?php
    $stid1 = oci_parse($conn1, 'Test Ekibinde Aksiyon Bekleyen
Hatalara Ait SQL Sorgusu'
    );

    $stid2 = oci_parse($conn1, 'Geliştirme Ekibinde Aksiyon
Bekleyen Hatalara Ait SQL Sorgusu'
    );

    oci_execute($stid1);
    oci_execute($stid2);
?>

<div style="width: 900; height:1800; padding: 5px;">
    <div  id="table_testte_bekleyen_1"  style ="width:100%;
padding: 2px;">
        <p>
            <h4>Test Ekibinde Bekleyen Defectler</h4>
        </div>
        <!-- Test Ekibinde Bekleyen Hatalar-->
```

```
<div id="table_testte_bekleyen" style ="width:100%;
padding: 2px;">
<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript" >
google.charts.load('current', {'packages':['table']});
google.charts.setOnLoadCallback(drawTable);

function drawTable() {
var data = new google.visualization.DataTable();
data.addColumn('string', 'DEFECTID');
data.addColumn('string', 'STATUS');
data.addColumn('string', 'SEVERITY');
data.addColumn('string', 'DETECTED_ON_NODE');
data.addColumn('string', 'CATEGORY');
data.addColumn('string', 'ASSIGNEDTO');
data.addColumn('string', 'DETECTION_DATE');
data.addColumn('string', 'OZET');
data.addColumn('string', 'LAST_ACTION_DATE');
data.addRows([
<?php
while ($row
=oci_fetch_array($stid1,OCI_ASSOC+OCI_RETURN_NULLS)){
echo
"['".$row["DEFECTID"]."','".$row["STATUS"]."','".$row["SEVERITY"]."','".$row["
DETECTED_ON_NODE"]."','".$row["CATEGORY"]."','".$row["ASSIGNEDTO"].
"','".$row["DETECTION_DATE"]."',
'".$row["OZET"]."',
'".$row["LAST_ACTION_DATE"]."],";
}
?>
]);
```

```
                    var table = new
google.visualization.Table(document.getElementById('table_testte_bekleyen'));
                    table.draw(data, {showRowNumber: true, width: '900%',
'title': "Now you see the columns, now you don't!",height: '900%' });
                }
                        </script>
                    </div>
                    <!-- END OF Testte Bekleyen Defectler-->

                    <div id="table_testte_bekleyen_2" style
="width:100%; padding: 2px;">
                    </br>
                </br>
                <p>
                    <h4>Çözüm Bekleyen Defectler</h4>
                </div>
                <!-- Çözüm Ekibinde Bekleyen Hatalar-->
                <div id="cozum_bekleyen" style ="width:100%; padding:
2px">
                    <script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
                    <script type="text/javascript" >
                google.charts.load('current', {'packages':['table']});
                google.charts.setOnLoadCallback(drawTable);

                 function drawTable() {
                 var data = new google.visualization.DataTable();
                 data.addColumn('string', 'DEFECTID');
                data.addColumn('string', 'STATUS');
                data.addColumn('string', 'SEVERITY');
                data.addColumn('string', 'DETECTED_ON_NODE');
                data.addColumn('string', 'CATEGORY');
```

```
                    data.addColumn('string', 'ASSIGNEDTO');
                    data.addColumn('string', 'DETECTION_DATE');
                    data.addColumn('string', 'OZET');
                    data.addColumn('string', 'LAST_ACTION_DATE');
                    data.addRows([
                                        <?php
                      while ($row
=oci_fetch_array($stid2,OCI_ASSOC+OCI_RETURN_NULLS)){
                                echo
"['".$row["DEFECTID"]."','".$row["STATUS"]."','".$row["SEVERITY"]."','".$row["
DETECTED_ON_NODE"]."','".$row["CATEGORY"]."','".$row["ASSIGNEDTO"].
"','".$row["DETECTION_DATE"]."',
                                '".$row["OZET"]."',
                                '".$row["LAST_ACTION_DATE"]."],";
                       }
                       ?>
                    ]);

                      var table = new
google.visualization.Table(document.getElementById('cozum_bekleyen'));
                    table.draw(data, {showRowNumber: true, width: '900%',
height: '900%'});
                    }
                            </script>
                    </div>
                    <!-- END OF Çözüm Bekleyen Defectler-->
                    <?php }; ?>
                </div>
        </main>
```

Ek 3: Aylık Test Senaryo Sayısı ZingChart Örneği

```
<!-- Aylık Test Kosum Pie Chart-->
<div id="pie_proje_test_case" style="width:530px; height:550px;
float: left;">
<div id='aylik_kosum'><a class="zc-ref"
href="https://www.zingchart.com/"></a></div>
<script src=
"https://cdn.zingchart.com/zingchart.min.js"></script>
<script> zingchart.MODULESDIR =
"https://cdn.zingchart.com/modules/";
ZC.LICENSE =
["569d52cefae586f634c54f86dc99e6a9","ee6b7db5b51705a13dc2339db3edaf6d"];</
script>
<script>
var str2 = "<?php echo $selected_val ?>";
var myConfig = {
type: "pie",
noData:{
text:"Empty Series",
backgroundColor: "#20b2db"
},
plot: {
borderColor: "#2B313B",
borderWidth: 0,
// slice: 90,
valueBox: {
rules: [
{
rule: '%v === 0',
text: ''
}
```

```
],
            placement: 'out',
                text: '%t\n%v',
                decimals:0,
            fontFamily: "Open Sans"
          },
          tooltip:{
            fontSize: '14',
            fontFamily: "Open Sans",
            padding: "1 2"
          },
          animation:{
    effect: 2,
    method: 10,
    speed: 500,
    sequence: 1
  }
        },
          subtitle: {
          text: "Aylık Test Kosum Sayısı",
          align: "center",
          offsetX: 5,
          offsetY: 1,
          bold:true,
          fontFamily: "Open Sans",
          fontSize: 18
        },
        plotarea: {
          margin: "20 0 0 0"
        },
        series : [
        <?php
```

```
                    while ($row
=oci_fetch_array($stid,OCI_ASSOC+OCI_RETURN_NULLS)){
                        echo "{values: [".$row["CNT"]."],text :
"."'".$row["GRP"]."'},";
                    }
                    ?>
                ]
        };

        zingchart.render({
            id : 'aylik_kosum',
            data : myConfig,
            height:  550,
            width: 550
        });
                    </script>
                    </div>
                <!-- EOF Aylık Test Kosum Pie Chart-->
```