YÖNETİM BİLİŞİM SİSTEMLERİ DERGİSİ
http://dergipark.gov.tr/ybs

Yayın Geliş Tarihi: 02.10.2019
Yayına Kabul Tarihi: 03.12.2019
Online Yayın Tarihi:07.12.2019

Cilt:5, Sayı:1, Yıl:2019, Sayfa:52-63
ISSN: 2148-3752

YAZILIM TEST SÜRECİNDE DURUM RAPORLAMASINA GENEL BAKIŞ VE YAKLAŞIMLAR

Beytullah UZUN

Ericsson Araştırma Geliştirme ve Bilişim Hizmetleri A.Ş.

Kutan KORUYAN*

Dokuz Eylül Üniversitesi, Yönetim Bilişim Sistemleri Bölümü

Öz: Günümüz dünyasında hızla büyümekte olan teknoloji sistemleri hayatımıza hızlıca yerleşmiş ve yaşamımızın her alanında yerini almıştır. Teknoloji firmaları ürünlerinin pazarlanması ve piyasa içerisinde kalabilmeleri için büyük bir rekabetin içerisinde bulunmaktadırlar. Özellikle yazılım şirketlerinin piyasa içerisinde var olabilmek ve sunduğu hizmetlerle müşterilerinin memnuniyetini kazanabilmek için ürün ve hizmetlerinin iyi bir test sürecinden geçirmeleri önem taşımaktadır. Bir yazılım projesinin kalitesi müşteriye sunulmadan önce en erken test aşamasında görülebilmektedir. Test sürecinde projenin ne kalitede olduğu, projenin istenilen gereksinimleri karşılayıp karşılamadığı ve bulunan hataların durumlarıyla ilgili bilgiler test durum raporlamalarıyla görüntülenebilmektedir. Bu çalışmada yazılım test süreci boyunca yapılacak sürekli raporlamalar ve raporlamaların önemine değinilmekte, raporlamalara dair önerilerde bulunulmaktadır.

Anahtar Kelimeler: Yazılım Testi, Yazılım Test Durum Raporlaması, Yazılım Test Hata Raporlaması

* Sorumlu Yazar: kutan.koruyan@deu.edu.tr, Dokuz Eylül Üniversitesi, İzmir, Türkiye

AN OVERVIEW AND APPROACHES TO STATUS REPORTING IN THE SOFTWARE TEST PROCESS

Abstract: Rapidly growing technology systems in today's world have quickly settled in our lives and have taken its place in every aspect of our lives. Technology companies are in a big competition to market their products and stay strong in the market. It is especially important for software companies to have a good testing process in order to secure their place in the market and gain the satisfaction of their customers with the services they provide. The quality of a software project is provided first in the testing stage. The quality of the project, whether meet the requirements or not, the information about bugs found during testing process could be viewed in the test reports. In this study continuous reporting during testing process are discussed and also recommendations about reporting are made.

Keywords: Software Testing, Software Test Reporting, Software Defect Reporting

GİRİŞ

Kaliteli yazılımlar, kabul edilebilir seviyede hatasız, belirlenen beklentileri karşılayabilen yazılımlardır. Yazılım testleri ise kaliteli yazılımlar için vazgeçilmez bir süreçtir. 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 müşteriye ürün teslim edilmeden önce mümkün olduğunca çok sayıda olası hatanın ortaya çıkarılması ve düzeltilmesi için yapılan aktivitedir (Lewis, 2017: 3). Ayrıca yazılım testi, 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. Yazılım test sonuçları ve ilerleyişi sürekli olarak raporlamalar ile proje ekibine sunulmakta, böylece raporlama ile test süreci takip edilmektedir. Oluşturulan raporlar ile tüm paydaşlar projenin ilerleyişi ve alınacak aksiyonlar hakkında güncel bilgilere sahip olmaktadır. Projede oluşacak riskler ve alınması gereken önlemler için yine proje ekibine bilgi sağlayacak olan yazılım test durum raporlamalarıdır.

Yazılım testlerinin önemi geçmişte başarısızlıklarla sonuçlanmış olan birkaç yazılım projesiyle görülebilmektedir. Örnek olarak 1962 yılında fırlatılan Mariner 1 uzay roketi, 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). Denver Havaalanı Bagaj Otomasyon Sistemi örneğinde ise gerçeğe yakın olmayan yetersiz simülasyonlar ile sistemin test edilmesinden dolayı başarısız olduğu, zamanında bitirilemediği, sürdürülemediği ve maliyetli olduğu görülmüştür (Calleam Consulting Ltd, 2008: 4-6). Başka bir başarısız proje örneği olarak NASA'nın 1999 yılında Mars'a gönderilmesi hedeflenen iniş aracı gösterilebilir. Bu projenin başarısızlığının sebebi olarak farklı yazılım ekipleri tarafından geliştirilen modüllerin, proje ekipleri arasında ölçü biriminin yanlış anlaşılması sonucunda başarısız olduğu görülmektedir. Bu başarısızlık örneği modüller arasındaki entegrasyon hatasını göstermektedir. Düzeltilmesi çok kolay olan bu hata son derece pahalı olmuş ve prestij kaybına sebep olmuştur (Ammann ve Offutt, 2016: 32).

Yazılım testleri aşağıdaki maddelerin sağlanması için önem arz etmektedir (Naik ve Tripathy, 2008):

- 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.

Test sırasında bulunan belirlenen gereksinimleri karşılamayan ya da son kullanıcının beklemediği şekilde hata ile sonlanan senaryolara yazılım hatası

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

- Yazılım, gereksinimlere göre yapması gereken bir şey yapmaz.
- Yazılım, gereksinimlere göre yapmaması gereken bir şey yapar.
- Yazılım, gereksinimlerde belirtilmeyen bir şey yapar.
- Yazılım, gereksinimlerde belirtilmeyen ancak yapılması gereken bir şey yapmaz.
- Yazılım yavaş çalışıyor veya anlaşılması ve kullanımı zor ise yazılım hatası olarak adlandırılabilir.

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. Yazılım ürünlerinin büyüklükleri ve karmaşıklık seviyeleri arttıkça test sürecinde bulunan hataları yönetmek zorlaşmaktadır. Bu karmaşıklığı yönetmek ve kaliteli yazılımlar ortaya çıkarabilmek için piyasada çok sayıda yazılım test ve hata yönetim araçları bulunmaktadır. Testi gerçekleştiren kişiler çıkan hataları adresledikten sonra çözümüyle ilgili süreçleri takip etmeli, bu takibi ise sürekli raporlamalar ile gerçekleştirmelidir. Yapılan raporlamalar; projede bulunan hataların projeyi ne seviyede etkilediğini, projenin plana göre ilerleme uyumunu, kritik hatalar için öncelikli önlem alınmasını sağlamakta ve proje ekibine sayısal olarak zengin bir raporlama sunmaktadır.

Test süreç takibinin kontrol edilmesi ve izlenmesi, ayrıntılı ve tutarlı test verilerinin toplanmasıyla gerçekleştirilmektedir. Test durum raporlama ile proje test süreç durum bilgisi ve bu süreçte bulunan hata ve çıktılar diğer proje paydaşlarına etkili bir şekilde aktarılmaktadır (Graham ve diğerleri, 2008: 24-25).

Yazılım test tekniklerinde fonksiyonel test, entegrasyon testi, birim testi, regresyon testi, performans testi, kabul testleri ve son zamanlarda en populer teknik olan otomasyon testi bulunmaktadır. Test, yazılım geliştirmede yaşam döngüsünde en fazla zaman alan aşamasıdır, bu maliyetin ve zamanın azaltılması için ise test otomasyon yöntemleri kullanılmaktadır. Otomatik test prosedürleri yalnızca maliyetin düşürülmesine katkı sağlamamaktadır. Bunun yanında, projede kalite artışını da hedeflemektedir. Bu nedenle test otomasyonu ve yönetimi giderek kritik ve stratejik bir öneme sahip olmaktadır (Marijan ve diğerleri, 2010:125)

Proje yöneticisine, müşteri ve diğer paydaşlara proje durumu hakkında bilinçli kararlar verebilmelerine yardımcı olacak düzenli raporlar verilmelidir. Yazılım ürünlerinin kalitesi test senaryo ve gereksinimlerin başarılı/başarısız olma durumuna göre değerlendirilmektedir. Ayrıca raporlar yazılımda bulunan hataların özelliklerine göre proje durumu hakkında bilgilendirme yapılmasına ve kritik kararlar alınmasına yardımcı olmaktadır. Test raporları karar vericilerin kararlarını destekleyecek bilgiler sunmaktadır. Bunlar;

- Projeye devam edilmesine ya da projenin iptal edilmesine yön verecek kararlar test raporlarına göre oluşturulmaktadır. Projede

çıkan hataların tipine, şiddetine, önceliğine, durumuna, çözüm sürelerine ve hataların test senaryolarına etki analizine göre yapılacak olan analizlerle, projenin iptal edilmesine ya da devam edilmesine karar verme mümkün olmaktadır.

- Projede kaynak, süreç ve zaman planlamasının tekrar yapılmasına yönelik sonuçların çıkarılmasına yardımcı olacak sonuçlar vermektedir. Raporlamalarda sunulacak gerçekleştirilen ve planlanan zaman grafiğine göre projenin hedef tarihindeki sapmalar önceden belirlenmiş olmakta ve bu sapmaların nedenlerine göre gerekli aksiyonların önceden alınmasına katkıda bulunmaktadır.

Bu çalışmada yazılım test sürecinde yapılan raporlamalar incelenmiş ve bu raporlamaların nasıl ve hangi durumlarda kullanılacağı anlatılmıştır. Yapılan raporlamalar üzerine yorumlar getirilmiş olup yazılım projelerindeki önemi hakkında bilgi sağlanmıştır.

TEST DURUM RAPORLARININ İNCELEMESİ

Projenin kalitesinin en iyi görülebildiği aşama olan test sürecinin ilerleme durumunun sürekli raporlamalarla tüm proje paydaşları ile paylaşılması gerekmektedir. Bu sürecin takibinde oluşacak riskler ve önceden önlem alınabilmesi için raporlamaların devamlılığı önem arz etmektedir. Böylelikle raporların oluşturulmasında harcanacak efor, daha kaliteli bir test gerçekleştirmek için kullanılabilir. Raporlar testlerin uygulanması sonucunda elde edilen verilerin, test yönetim uygulamasından çekilmesiyle gerçekleştirilmektedir.

Piyasada bulunan pahalı ve lisanslı yazılım test ve hata yönetim araçlarının yeterli grafiksel arayüzleri (GUI) bulunmamakta, hali hazırda grafiksel arayüzleri bulunan yazılımlardan raporların oluşturulması da teknik bilgi ve uzmanlık gerektirmektedir. Test sürecinin en önemli kısmı olan raporlama modülünün oluşturulması bu arayüzler ile zaman kaybına sebep olmaktadır. Ayrıca bilişim sektörü içerisinde bulunan teknik ekipler dışındakiler tarafından raporların oluşturulması ve anlaşılması zor olmaktadır.

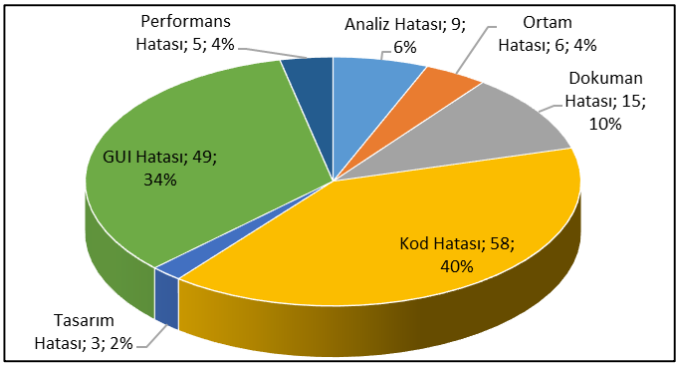
Test süresi boyunca bulunan hatalar ve gerçekleştirilen test senaryolarının başarı oranları yazılımın kalitesi hakkında bilgi vermekte, ürünün müşteriye sunulup sunulmamasına, alınması gereken önlemler için önceden bilgi vermesine, çıkabilecek maliyet ve risklerin ortaya çıkarılmasına yardımcı olmaktadır. Bu bilgilerin yöneticilere sunulması için çıkan hatalardan analiz ve raporlamaların yapılması gerekmektedir. Geliştirme ekiplerine ve yöneticiler için gerekli olan raporlamalar oransal ve grafiksel olarak farklı şekillerde olabilmektedir. Raporlar; hata tipine, hata şiddetine, hata önceliğine, hata durumuna, test senaryo ilerlemesine, hata ve senaryo sayısına göre test ekibi ve proje yöneticisine sunulmaktadır.

Hata tipine göre dağılımlar

Testlerde bulunan hatalar; dokuman hatası, kod hatası, analiz hatası, tasarım hatası, ortam hatası, diğer hatalar vb. olarak sınıflandırılmaktadır. Bulunan hataların sınıfına göre hataların ana sebebinin nereden ve hangi ekipten kaynaklandığı ile ilgili bilgi “Hata Tipine Göre Dağılım” grafiği ile gösterilmektedir (Şekil 1). Bu dağılım grafiğine göre hataların en çok hangi tipte olduğu, hangi bölüme daha çok odaklanılması gerektiği görülebilmektedir. Şekil

1'deki örnekte projede hataların %40'ının kod hatasından kaynaklandığı görülmektedir. Kod hataları geliştirmede bulunan kodun yanlış yazımından dolayı kullanıcıya istenilen sonucun verilememesi durumudur. Arayüz hatası ise %34 oranında bulunmaktadır. Arayüz hatası ise kullanıcının etkileşime girdiği önyüzün beklenildiği gibi çalışmamasından kaynaklı hatalardır. Bu projede ayrıca %10 oranında doküman hatası, %6 oranında analiz hatası, %4 ortam hatası, %4 oranında performans hatası ve %2 oranında tasarım hatası bulunmaktadır. Bu oranlar test sürecinin başından itibaren bulunan tüm hataların tiplerini göstermektedir. Bu hataların hangi durumda oldukları, çözüm süreçleriyle ilgili bilgiye Şekil 3'te bulunan hata durum dağılım grafiğinden ulaşılabilir.

Şekil 1. Hata Tipine Göre Dağılım



Hata şiddetine göre dağılımlar

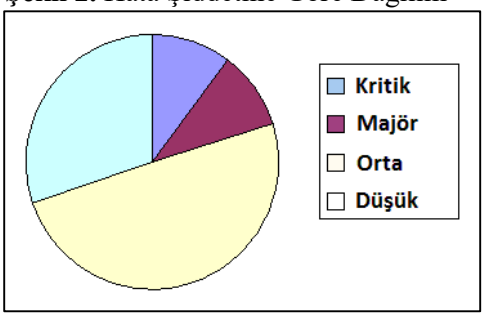
Test durum raporlarında kullanılması gereken ve hataların önem derecelerini yönetici ve ilgili düzeltme ekibine gösteren dağılım grafiği şiddet dağılım grafiğidir. Hataların şiddeti arttıkça öncelikli olan hataların düzeltilmesi büyük önem kazanmakta ve düzelene kadar da diğer gereksinim ve fonksiyonelitelerin 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 Şekil 2'de gösterildiği gibi pasta grafik yöntemiyle ya da bir tablo biçiminde raporlanabilmektedir.

Şekil 2'de gösterilen hata şiddetleri Software Testing Fundamentals'a (bt.) göre aşağıdaki gibi sınıflandırılmıştır;

- *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 yapılamamasıdır.
- *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.

Şekil 2. Hata Şiddetine Göre Dağılım



(Rational Software Corporation, 2001)

Hata önceliğine göre dağılımlar

Hata önceliğine göre verilen hata dağılım grafikleri, önceliklendirilmesi gereken hataları çözüm ekiplerine göstermekte ve bu öncelik sırasına göre hata çözümlerinin alınması hedeflenir. 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. Bu raporlar hata çözümlerinin öncelikle çözüme alınması gereken hataları göstermekte ve çözüm ekiplerine yol gösterici rolünü almaktadır. Hata önceliğine göre dağılım grafiği Şekil 2'de bulunan hata şiddetine göre dağılım grafiği gibi hata önceliklerinin gruplanmasıyla gösterilmektedir.

Hata durum dağılımı

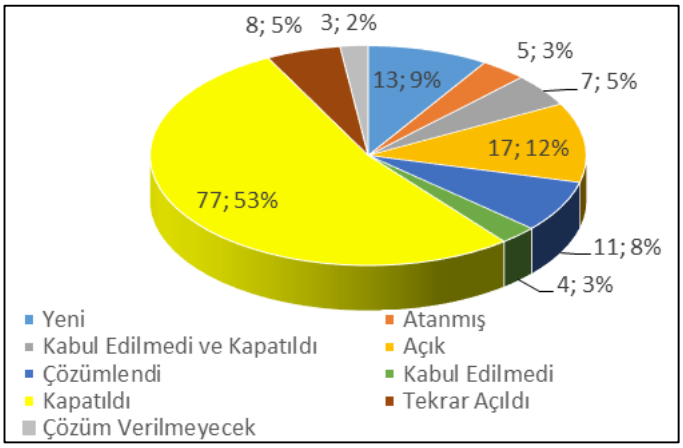
Yazılım hata durumları; yeni, atanmış, açık, çözümlendi, kabul edilmedi, kabul edilmedi ve kapatıldı, tekrar aç, çözüm verilmeyecek ve kapalı statüsünde olabilmektedir. Yazılımda çıkabilecek hatalar bu yaşam döngülerini gerçekleştirmektedir. Bu statülere göre çıkan hatalar sınıflandırılmalı ve raporlarda 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 ve çok fazla hatanın biriktiğini göstermektedir. Düzenli raporlamalar yapılarak hataların çözümleri sürekli takip edilmeli ve böylece 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 ya da çözümlendi statüsündeki oranın artışı, test ekibi üzerinde işlerin biriktiği ve aksiyon almaları gerektiğini göstermektedir.

Ayrıca bu grafik, ekiplerin kalitelerinin göstergesi olarak da kullanılabilmektedir. Açılan hatalarda çok fazla kabul edilmedi ve kapatıldı statüsü oranındaki artış, test ekibinin testi gerçekleştirme kalitesinin artırılması için alınması gereken aksiyon olarak yansıyacaktır. Test hata raporundaki bu artış test ekibinin hataları açmadan tekrar kontrol etmeleri ve test edilen uygulama üzerindeki yetkinliklerini artırması gerektiğini göstermektedir.

Şekil 3'te bulunan hata durumuna göre dağılım grafiği örneğinde açıkta bekleyen hataların test ve geliştirme ekipleri üzerinde olduğu görülmektedir. Test ekibinde 15 adet aksiyon alınması gereken hata bulunmakta ve projede gecikme yaşanmaması için çözümü verilen ve kabul edilmeyen bu hataların tekrardan kontrolleri ve testlerinin yapılması gerekmektedir. Ayrıca yeni, atanmış, açık ve tekrar açıldı statüsünde bulunan toplamda 43 adet hata için de geliştirme ekiplerinin hatalar için çözüm sağlaması beklenmektedir. Proje planının etkilenmemesi ve risklerin önceden görülebilmesi için bu statüde bulunan hata oranının azaltılması gerekmektedir.

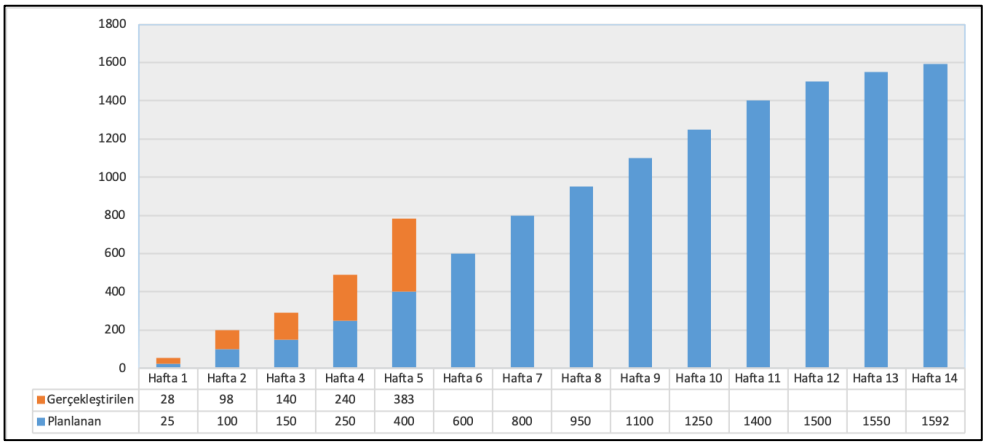
Şekil 3. Hata Durum Dağılım Grafiği



Test senaryosu ilerlemesi

Proje ekibine test sürecinin tamamlanma oranı ve plana uygunluk durumu Şekil 4'te örneği verilen "Test Senaryo İlerleme Durum Grafiği" ile gösterilmektedir. Bu raporda toplam test senaryo sayısına göre test senaryo ilerleme grafiği oluşturulur ve belirli periyotlar ile proje ekibine sunulan raporlarda paylaşılır.

Şekil 4. Test Senaryo İlerleme Durum Grafiği



Şekil 4'te 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. Böylece, test ilerleme grafiğine göre proje ekibi test ilerleme durumu hakkında bilgi sahibi olacaktır. Bu grafiğe 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: s. 421).

Hata ve senaryo sayısına göre analiz raporları

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

Test verimlilik durumu; testler sırasında hataların bulunması ve kaçırılmaması test verimliliğini göstermektedir. Test verimliliği ölçümünü Formül 1 ile hesaplanmaktadır (Lazic ve Mastorakis, 2008: 599-619):

$$Test\ Verimliliği = \frac{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı}{Testler\ Sırasında\ Bulunan\ Hata\ Sayısı + Canlı\ Sistemde\ Bulunan\ Hata\ Sayısı} \times 100 \quad (1)$$

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 aynı konu için birden fazla açılmış olmayan ve sadece kabul edilmiş hataların olması gerekmektedir. Aynı konuda birden fazla açılmış olan veya 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 tekrarlanmayan ve kabul edilmeyen hata sayıları çıkarılarak hesaplanmalıdır (Black, 2002: 164-166). Raporlamalarda kullanılan diğer oranlamalar aşağıda verilmektedir.

- *Başarısız Test Senaryo Oranı*, testler sırasında başarısız olan test senaryo sayısının toplam gerçekleştirilen test senaryo oranına göre hesaplanmaktadır. Başarısız test senaryo oranının düşük olması, test ekibine iletilen kodun ne kadar düzgün olduğunu göstermektedir. 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 alabilmektedir. Alınan önlemlerden sonra bu oranın takibine devam edilmekte ve ileride yapılacak projeler için yol gösterici metotlar geliştirilebilmektedir (Lazic ve Mastoralis, 2008: 603-604).
- *Kabul Edilmeyen Hata Oranı*, kabul edilmeyen hataların toplam açılan hataya bölümünden çıkan oran ile ifade edilir. 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 projede maliyetlerin arttığı görülebilir. Kabul edilmeyen her hatada, test ekiplerinin ve çözüm ekiplerinin boşa harcadığı efor görülmektedir. Çıkan oran sonucuna göre yöneticiler tarafından, kabul edilmeyen hata oranını düşürecek aksiyon alınması gerekecek ve maliyet, zaman tasarrufu yapılmış olacaktır.

- *Hata Çözümleri İçin Harcanan Ortalama Süre*, 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. Bulunan hatalar şiddetlerine göre gruplandırılmakta ve çözüm sürelerine uygun olarak çözülmeyen maddeler Tablo 1'de kırmızı renk ile gösterilmiştir. Çözüm süresine uygun olarak aksiyon alınan maddeler ise yeşil renk ile gösterilmiştir.

Tablo 1. Hata Test Çözüm Süreleri

Hata Şiddeti	Toplam Hata	Çözüm Süresi	Olması Gereken (Gün)
1-Düşük	21	5,83	8
2-Orta	43	3,45	4
3-Majör	11	2,58	2

- *Tekrar Açılan Hata Oranı*, tekrar açılan hata sayısının toplam bulunan hata sayısına oranıyla hesaplanır. Geliştirme ekipleri tarafından 'Kabul Edilmedi' statüsüne getirilen hatalar test ekibi tarafından incelenir. Kabul etmeme sebebi, açılan hatanın gerçekten bir hata olmadığını belirtmektedir. Hatanın kabul edilmeme sebebi hata olmadığının 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.
    - 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.

SONUÇLAR VE TARTIŞMALAR

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. Testlerdeki asıl amaç, piyasaya daha kaliteli bir ürün çıkarmak için çalışmaktır. Testin hangi ölçüde başarı ile sonuçlandığının en önemli göstergesi ise test verimliliği raporudur. Test süreci boyunca yakalanan hataların oranı %100'e yaklaşması testin başarı oranını ve projenin başarıyla test sürecinden geçirildiğini gösterecektir.

Günümüzdeki yazılım projelerindeki zaman kısıtından dolayı hataların bulunması için manuel test yöntemleriyle beraber otomasyon testleri de kullanılmaktadır. Son zamanlarda test otomasyonuna büyük bir eğilim bulunmakta, projelerin iyi bir test sürecinden geçirilmeleri için üründeki tüm özelliklerin çalıştığından emin olunmalı ve bu süreçte varsa hataların düzeltilmesi gerekmektedir. Bu çalışmada anlatılan ürünler üzerine yapılacak her bir değişiklikte ya da hata düzeltmelerinde, ürünün genel olarak tekrardan kontrolü için test otomasyon yöntemleri kullanılmalı ve bu otomasyon sonucunda bulunacak hatalar da tüm detaylarıyla birlikte anlık olarak ilgili kişilere raporlanmalıdır.

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 dışında olan yöneticiler tarafından raporların anlaşılması zor olmaktadır. Bu olumsuzluklarla başa çıkabilmek için açık kaynak kodlu yazılımlar kullanılarak, kullanımı kolay ve anlaşılır raporlama ekranları oluşturulması sağlanmalıdır.

Yazılım sürecinde yazılımın kalitesinin görüldüğü aşama test durum raporlardır. Yazılım test sürecinin en iyi şekilde görüntülenebilir hale gelmesi ve yazılımın kalitesi hakkında bilgi sağlayan raporlamaların oluşturulması için raporlama modülü kullanılmalıdır. Bu raporlama modülü sayesinde oluşturulması zor olan ve zaman kaybına yol açan raporlara kolayca erişilebilecektir. Tüm proje ekip ve yöneticileri tarafından kolaylıkla erişilebilir ve anlaşılabilir raporlamaların oluşturulması göz önünde bulundurulmalıdır. Bu yüzden, karar vericilerin zaman kaybetmeden anlık ve gerçek zamanlı olarak raporlara ulaşabileceği uygulamaların geliştirilmesi gerektiği düşünülmektedir. Test yönetimi için hali hazırda var olan ve yeni geliştirilecek yazılım test süreç uygulamalarının hepsiyle uyumlu çalışacak bir raporlama sistemi oluşturularak herkes tarafından kabul edilip kullanılacak bir rapor modülü haline getirilmiş olacaktır. Böylece bilişim problemleri karşısında, bilişim sistemleri kullanılarak yazılım test süreçleri ve raporlama tipleri analizi yapılıp, problem çözümü için karar gereksinimleri araştırılıp, geliştirilecek uygulama ile yazılım testlerinin karar verme süreçlerinde etkili rol alması ve kaliteli yazılımların oluşturulması hedeflenecektir.

Bu çalışmanın, yazılım testi yapan firmalara test süreçlerinin en iyi şekilde sonuçlanması ve sürecin proje ekibi tarafından en verimli şekilde takibi yönünden bir yol gösterici olduğu düşünülmektedir.

KAYNAKLAR

Ammann, P., ve Offutt, J. (2016). *Introduction to software testing*. Cambridge University Press.

Black, R. (2002). *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. USA: Wiley Publishing, Inc.

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

Calleam Consulting Ltd. (2008). "Case Study – Denver International Airport Baggage Handling System – An illustration of ineffectual decision making". https://www5.in.tum.de/persons/huckle/DIABaggage.pdf, (30.04.2018).

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

Johnson, P. (2012). *Mariner 1's $135 Million Software Bug*. https://www.itwo-rld.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.

Lewis, W. E. (2017). *Software testing and continuous quality improvement.* Auerbach publications.

Naik, K. ve Tripathy, P. (2008). *Software Testing and Quality Assurance: Theory and Practice*. USA: John Wiley & Sons, Inc.

Marijan, D., Zlokolica, V., Teslic, N., Pekovic, V., ve Tekcan, T. (2010). *Automatic functional TV set failure detection system, IEEE Transactions on Consumer Electronics*, 56(1): 125-133.

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

Rational Software Corporation (2001). *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).

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