T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM TEST SÜRECİNDE HATA YÖNETİMİ VE YENİ BİR HATA YÖNETİMİ UYGULAMASININ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Ayşe Betül KARAGÖZ

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Prof. Dr. Ahmet ZENGİN**

Şubat 2021

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Ayşe Betül KARAGÖZ

10.02.2021

TEŞEKKÜR

Tez çalışmamda bana danışmanlık eden hocam Prof. Dr. Ahmet ZENGİN'e,

Yüksek Lisans öğrenimim boyunca bilgilerinden, tecrübelerinden faydalandığım tüm hocalarıma,

Bu tezin hazırlanmasında öncülük eden Kuveyt Türk Ar-Ge Merkezine,

Teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

BOA	: İş Odaklı Mimari (Business Oriented Architecture)
BT	: Bilgi Teknolojileri
IDE	: Entegre Geliştirme Ortamı (Integrated Development Environment)
ISO	: Uluslararası Standardizasyon Organizasyonu (International Organization for Standardization)
IEC	: Uluslararası Elektroteknik Komisyonu (International Electrotechnical Commission)
IEEE	: Elektrik ve Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electronics Engineers)
IIS	: İnternet Bilgi Servisi (Internet Information Services)
ISTQB	: Uluslararası Yazılım Test Yeterlilikler Kurulu (International Software Testing Qualifications Board)
MVC	: Model Görünümü Denetleyicisi (Model View Controller)
WPF	: Windows Sunum Temeli (Windows Presentation Foundation)

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar Kelimeler: Yazılım testi, hata yönetimi, hata raporlama, gereksinim

Yazılım Yaşam Döngüsü, bir yazılım işinin veya projesinin fikir aşamasından itibaren analiz, geliştirme, test ve canlı ortama çıkış aşamalarını kapsayan bir süreçtir. Yazılım Testi ise bu süreçte yapılan yazılımın doğrulanma işlemlerini içeren önemli kısmını oluşturmaktadır. Yazılım Testi genel olarak Hata Yönetimi araçları ile yönetilmektedir. Bu süreç boyunca araçlar kullanıldığında birçok paydaşın olduğu bu uzun sürecin izlenebilirliği sağlanır ve güçlü bir raporlama avantajı sağlar. Birçok marka veya kurum bu süreçler için araç kullanımını tercih etmektedir. Araçların çoğu tek başına yetersiz kalmaktadır. Bu durum için çeşitli çözümler bulunmaktadır; Kullanılan araca uyumlu başka araçlarla eksiklikler tamamlanabilir. Açık kaynak kodlu araçlar üzerinde geliştirme yapılabilir veya kendi süreçlerine uygun araçlar geliştirilebilir. Bu tezde halihazırda kullanılan bir hata yönetimi aracı tecrübesinden yola çıkılmıştır. Araç ile ilgili çalışanlardan gelen olumsuz geri bildirimler ve öneriler değerlendirilmiş ve süreci daha pratik hale getiren yeni bir uygulama geliştirilmiştir. Uygulamada testin ana girdisi olan gereksinimler test edilebilir gereksinimler olarak alınmıştır. Gereksinim ekranında hata girişi sağlanmıştır. Gereksinimler, test senaryolarına dönüştürülerek aynı ekran içinde test durumları gösterilmiştir. Süreçler, mevcuttan ve çeşitli araçlardan çok daha kısa hale getirilmiştir. 39 adımda yapılan bir hata raporlama işlemi 8 adıma indirilmiştir. Buna ek olarak geliştirilen uygulamanın karmaşık olmaması, ayrı bir giriş ekranı olmaması, gereksinimlerle senaryoların aynı ekran üzerinde görülebilmesi, hata şablonu oluşturulabilmesi hata girişi maliyetini yaklaşık %80 azaltmıştır. Ayrıca çalışanın motivasyonunu artırması uygulamayı daha verimli hale getirecektir. Geliştirilen hata yönetiminde mevcut sistemde olmayan aşağıdaki özellikler bulunmaktadır. Kullanıcı ve yetki tanımlama, lisans ve bakım ücreti, araç eğitimi bu uygulamada olmayacaktır. Mevcut sistemdeki kullanım zorluğu en aza indirgenmiştir. Senaryolar otomatik olarak gereksinimlerden üretilmektedir. Hata kayıtlarında şablon oluşturulmaktadır. Hata kaydı ekranına işletim sistemi versiyonu, internet tarayıcı çeşidi gibi bir çok gerekli detay bilgiler eklenmiştir. İstatistiksel bilgiler tutulmasını kolaylaştıran bu ek geliştirmeler genel iyileştirmeyi sağlayacaktır.

DEFECT MANAGEMENT IN SOFTWARE TEST PROCESS AND A NEW DEFECT MANAGEMENT APPLICATION DEVELOPMENT

SUMMARY

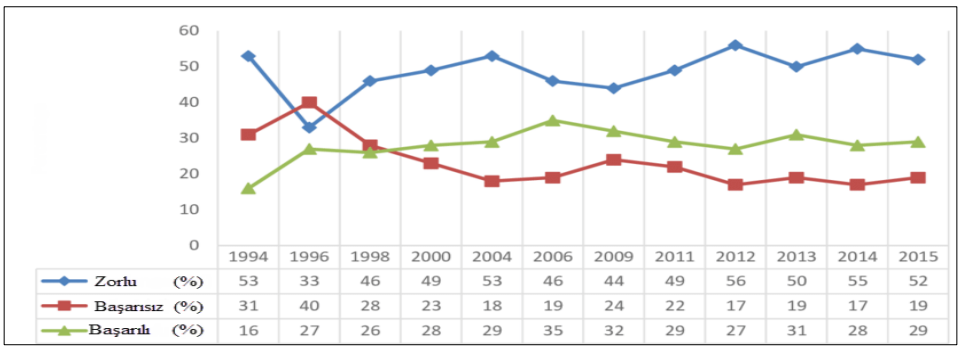
Keywords: Software test, defect management, defect reporting, requirement

Software Life Cycle is a process that covers the stages of analysis, development, testing and going live from the idea stage of a software business or project. Software Testing, on the other hand, constitutes the important part that includes the verification processes of the software performed in this process. Software Testing is generally managed by defect management tools. When tools are used during this process, the traceability of this long process involving many stakeholders as well as a strong reporting advantage is ensured. Many brands or institutions prefer to use tools for these processes. Most of the tools alone are insufficient. Various solutions are available for this situation; the deficiencies can be completed with other tools compatible with the used tool. Tools with open source codes can be developed. Or, tools that are suitable to their own processes can be developed. This thesis is based on the experience of a currently used defect management tool. Negative feedback and suggestions from employees about the tool were evaluated and a new application that makes the process more practical was developed. Requirements, which are the main input of the test in practice, were taken as testable requirements. A defect entry was provided on the requirement screen. Requirements were transformed into test cases and the test cases were displayed on the same screen. Processes have been made much shorter than existing ones and various tools. A defect reporting process made in 39 steps has been reduced to 8 steps. In addition, the developed application is not complicated, does not have a separate login screen, the requirements and scenarios can be viewed on the same screen, and the defect template can be created, reducing the cost of defect entry by approximately 80%. Furthermore, increasing the motivation of the employee will make the application more efficient. In the developed defect management, there are the following features that are not available in the current system. User and authorization identification, license and maintenance fee, vehicle training will not be in this application. The difficulty of use in the current system is minimized. Scenarios are automatically generated from requirements. Template is created in defect records. Many necessary detailed information such as operating system version and internet browser type have been added to the defect record screen.

BÖLÜM 1. GİRİŞ

Bilgi Teknolojileri (BT) projeleri düşük performans oranları ile ünlüdür (Joseph, Erasmus ve ark., 2014; Standish Group, 1995, 2013). Sayısına bakılmaksızın bu alanda yapılan araştırmaların neredeyse tamamı bunu göstermektedir (Eveleens ve ark., 2010; Carl Marnewick ve ark., 2013; Wateridge, 1998). Bilgi Teknolojileri projelerine büyük yatırımlar yapılmaktadır ancak bütün bu yatırımların karşılığı tam anlamıyla alınamamaktadır (Kossai ve Piget, 2014; Carl Marnewick, Basında).

2015 Kaos raporu başarısız ve zorlanılan projeleri yüzde 71 olarak bulunmuştur ve bu da 20 yıllık veri takibinde hiçbir gelişme olmadığını göstermektedir. (Hastie ve Wojewoda, 2015). Şekil 1.1.'deki yazılım projeleri başarı grafiğinde (Hastie ve Wojewoda, 2015) görüldüğü üzere BT projelerinin başarı oranları yatay devam etmektedir. Bu durum mevcudun iyileştirilmesinden ziyade mevcut işleyişe ilave performansı artıracak çalışmalar eklenmesi gerektiğini göstermektedir. Bu çalışmalar yazılım geliştiricinin dikkatini artırma, analisti geliştirme gibi sığ kalacak önlemler değildir. 20 yıllık bir grafikte bu tip eforların performansı ne kadar etkilediği Şekil 1.1.'deki grafikte çok açık şekilde görülmektedir. Bu efor öyle bir efor olmalıdır ki başarı ivmesini gözle görülür derecede artırsın ve en üst seviyelere çıkarsın.



Şekil 1.1. Yazılım projeleri başarı grafiği (Joseph, N., Marnewick, C. & Santana, MJ. 2016).

Yazılım kalite güvencesinin veya yazılım testinin nereden geldiğini anlamak için yazılım geliştiricilerin farklı becerilere sahip olması gerektiğini akılda tutmak önemlidir. Bunlardan biri, yazılımı kullanımı kolaylaştırarak yazılımın farklı ortamlarda ve yapılandırmalarda çalıştığını görmelidir (Tozzi, 2016). Yüzlerce ekran içeren sistemlerde çeşitli ekran standartları oluşturulması kullanılabilirlik açısından çok önemlidir. Yazılım projelerinde oluşturulan standartlara uyum sağlanması gerekmektedir. Özellikle kullanıcı deneyimleri göz ardı edilmemelidir. Bu bağlamda hem kullanılan hem müşteriye sunulan uygulamaların bu özellikleri içinde barındıran bir bakış açısıyla yazılmış olması önemlidir.

Kendall ve ark. (2013) tarafından yapılan çalışmaya göre yazılım yaşam döngüsü aşağıdaki 7 adımdan oluşmaktadır:

1. Problemlerin, hedeflerin, ihtiyaçların çıkarılması
2. İnsan enformasyonlarının netleştirilmesi
3. Sistem ihtiyaçlarının tespit edilmesi
4. Önceki adımlarda tespit edilen isteklere cevap veren bir sistemin tasarlanması
5. Geliştirmenin yapılması ve dokümana dökülmesi
6. Sistemin testlerinin yapılması ve hataların düzeltilmesi
7. Sistemin gerçek ortama alınması ve değerlendirilmesi.

Bu 7 adımda da görüldüğü üzere adımların yarıdan fazlası gereksinimlere dair adımlardır. Gereksinimler için bir şablon veya belirli kurallar oluşturulmalıdır. Çoğu zaman kapsam dokümanı veya müşterinin söylediklerini olduğu gibi yazmak ya da dokümanlardan elde edilen gereksinimleri aynen aktarmak, gereksinimleri yazmak gibi düşünülebilir. Gereksinim yazmanın bir tekniğinin olması gerektiği ve gereksinimlerin belli özellikler taşıması gerektiğinin farkına varılamaz (Hooks, 1993). Ancak hayati önem içeren sektörlerde analizin önemi yeterince anlaşılmıştır. Gereksinimlerin çıkarılmasından sorumlu olan analist projeye göre veya yapılacak işe göre iş müşteri veya dış müşteri ile görüşür. Kapsam toplantıları yapılır. Yapılacak işin kapsamı netleştirilir ve analizi/gereksinimleri çıkarma aşamasına getirilir. Bu gereksinimler sistem, güvenlik, iş vb. gereksinimlerin hepsini kapsar. Tüm bunları

içeren bir doküman oldukça uzun bir doküman olmaktadır. Bu doküman yazılım testinin adeta rehberidir. Analiz adımından sonraki tüm süreçler bu analize göre inşa edilir ve ilerler. Bu açıdan bu dokümanın anlaşılır detaylı, aynı zamanda sade olması çok önemlidir. Bunları göz önünde bulundurarak yapılmış gereksinim yazma araçları neredeyse yok gibidir. Mevcut kullanımda olan uygulamada gereksinimler sadece iş, güvenlik vb. şekilde gruplanarak metin şeklinde yazılmaktadır. Düz metin şeklinde oluşu hem okumada, incelemede çok zaman almakta hem de bazı kısımların gözden kaçırılmasına sebep olmaktadır. Aynı zamanda kullanılan hata yönetim aracının karmaşık bir yapıda olması gereksinimlerin sistemde tutulmayıp Excel, Word gibi dosyalarda tutulmasına da sebep olmaktadır. Analizlerin sistemde tutulması hem yasal bir zorunluluk hem de bilgi teknolojilerinin hafızası ve bilgi kaynağıdır. Ekran kullanımları, daha sonra yapılacak projelerde, yeni çalışan başladığında bu bilgi kaynağına ihtiyaç duyulmaktadır. Analiz inceleme, analizden hareketle geliştirme yapma, analizden test senaryosu yazmada zamandan kazanmak, analizin eksik kalması, analiz incelemede gözden kaçan kısımların olması gibi insan kaynaklı hataları en aza indirgemek maksadıyla maddeler halinde test edilebilir gereksinimlerden oluşan analizler oluşturulması çözüm sağlayacaktır.

Yazılım testi artık yazılım süreçlerinin temel basamaklarından, olmazsa olmazlarından biri haline gelmiştir. Yazılım testi, kalitenin başlangıç noktasıdır. Yazılım Testi, kurumların itibarının garantiye alan bir süreçtir. İnternet ve sosyal medya günümüzde çok aktif bir şekilde kullanılmaktadır. Pazara çıkmış sadece bir ürünün bile hata içermesi ilgili markanın veya kurumun bu hatadan kaynaklı itibar kaybını telafi etme şansını azaltmakta ve rakiplerini öne geçirmektedir. Özellikle güvenliğin ön planda olduğu banka, sağlık sektörlerinde test çok önemlidir. Doğru çalışmayan bir yazılım; para, zaman kayıplarına hatta ölümlere sebep olabilecek sonuçlar doğurabilir (Eser, 2019). İş kurallarının çok olduğu bankacılık sektöründe analiz, bu analizlerden çıkarılmış test senaryoları testin başarısının ön koşuludur. Çünkü yapılacak geliştirme de yazılım testi de analize göre yapılmaktadır. Markalar, kurumlar ana ihtiyaç haline gelmiş olan bu süreci en verimli ve en hızlı şekilde nasıl yapılabileceği konusunda çeşitli adımlar atmaktadır. Bazıları yazılım test süreçlerini, gereksinim yönetimi ve hata yönetimini yapan araçlar kullanmakta, bazıları bu araçların birkaçını entegre

olarak kullanmakta, bazıları araç tecrübesinden yola çıkarak kendi uygulamasını yazmakta, bazıları ise kendine uygun bir süreç yönetimini tamamen kendi ihtiyaçları çerçevesinde yazmaktadır. Kullanılan araçların ihtiyaçları tam karşılamaması, kullanıcı deneyimi konusunda başarısız oluşu çalışanları demotive etmektedir. Basit, anlaşılır, kullanıcı deneyimi başarısız araçları çalışanlar kullanmak istemektedir. Bu durum senaryoların excel dosyalarına yazılmasına, hataların excel dosyalarına kayıt olmasına hatta çoğu kez kayıt altına alınmamasına sebep olmaktadır. Oysa kayıt altına alınan tüm senaryolar kullanım kılavuzu gibidir ve yazılımın doğruluğunu, doğrulandığını teyit etmektedir. Kayıt altına alınan hatalar ile oluşturulan bir havuzda oluşturulan metriklerle ortam kaynaklı, fonksiyonel, tasarım, metin, kullanıcı deneyimi gibi kategorilere ayrıştırılması gibi istatiksel sonuçlar çıkararak hataların en aza indirgenmesi ve sistemdeki aksaklıkların büyük oranda giderilmesi sağlanmaktadır. Bu süreçleri kolaylaştırmak tüm süreçlerin kayıt altına alınmasını sağlamak bunu yaparken kaliteden, verimlilikten ödün vermemek üzere daha kullanışlı, çalışanı motive eden, pratik uygulamalar yapılması, zaman ve efordan tasarruf, çalışan motivasyonunu farkedilir derecede artıracaktır.

Tez ile birlikte çalışanın motivasyonu ve süreçlerin kayıt altına alınması öncelikli tutularak süreçte kullanılacak sade, anlaşılır, iş adımlarının en aza indirgendiği bir hata yönetimi sistemi yazılması amaçlanmıştır. Bu sistemin piyasadaki araçlara göre öne çıkan özelliklerini şöyle sıralayabiliriz:

- Kullanılan ana sistemin içindedir.
- Kullanıcıların erişimi kolaydır.
- Test edilebilir gereksinim üzerine oturtulmuş bir süreçtir.
- Anlaşılır, sade ve pratiktir.
- Mevcut işeleyişe göre maliyeti yaklaşık %80 oranında düşürmüştür.
- Bu oranın tecrübe edildikçe netleşmesi ve artması öngörülebilmektedir.

Bu tezin bilime katkısı, test edilebilir gereksinimlerle yazılmış bi hata yönetimi süreci ve bazı yaygın araçlardaki süreçlerin ne kadar kısaltılabileceğine dair bir deneyim içermesidir.

Tezin planı şu şekildedir;

Bölüm 1, Giriş bölümüdür. Bu bölümde Özet bir literatür taramasından sonra tezin motivasyonu, amacı, bilime katkısı hakkında hakkında bilgi vermekte ve tez planını içermektedir.

Bölüm 2, Literatür bölümüdür. Bu bölümde literatür araştırmaları, yazılım testi hakkında genel bilgiler, yazılım yaşam döngüsünde test, yazılım test seviyeleri literatür araştırmaları bulunmaktadır.

Bölüm 3, Hata Yönetimi bölümüdür. Bu bölümde Hata Raporlama, Hata Yaşam Döngüsü ve Yönetimsel Hata Raporlama konuları yer almaktadır.

Bölüm 4, Geliştirilen Hata Yönetimi bölümüdür. Bu bölümde Geliştirilen hata yönetiminin alt yapısı ve yazılımı ve geliştirilen hata yönetiminin ekranları ve tabloları gösterilmektedir.

Bölüm 5, Karşılaştırma bölümüdür. Bu bölümde mevcut hata yönetimi anlatılmakta ve geliştirilen hata yönetimi ile karşılaştırılmaktadır.

Bölüm 6, Tartışma ve Sonuç bölümüdür. Bu bölümde ise yapılan geliştirmeden alınacak sonuçlar, eleştiriler ve geliştirilebilir özellikleri anlatılmaktadır.

BÖLÜM 2. YAZILIM TESTİ

Yazılım testinin önemi yazılım projelerinde gün geçtikçe artmaktadır. Önceleri yazılım geliştiriciler kendi kodlarını test ederken yetersiz kalmıştır. Bir süre ve günümüzde bazılarınca tercih edildiği üzere analistler veya son kullanıcı yazılım testini yapmaktadır. Artık dünyada birçok yazılım geliştiren firma bağımsız test ekibi modelini ya da firma içinde kurulmuş ayrı bir test ekibini tercih etmektedir. Özellikle test ekibinin olduğu yerlerde süreç daha düzenli işler. Yazılım testi ile ilgili raporlar test mühendisi tarafından test ekibinden sorumlu olan test yöneticisine, test yöneticisi ile proje yöneticisine iletilir (Kaner ve ark., 1999).

2.1. Yazılım Testi

Yazılım Testi; bir sistemin/yazılımın belirlenmiş gereksinimleri yerine getirip getirmediğini belirlemek veya beklenen ile gerçekleşen sonuçlar arasındaki farkları tespit etmek amacıyla gerçekleştirilen bir dizi faaliyetler bütünüdür. Kendi içinde çeşitli süreç adımlarına sahiptir. Yazılım testinin çeşitli tanımlamaları mevcuttur;

Test, test edilen yazılımın kalitesi artırmak ve ölçmek amacıyla test yazılımlarını kullanıp gerekli değişiklikleri yapan mühendislik eşzamanlı yaşam döngüsü sürecidir (Craig ve ark., 2002).

Yazılım testi; yazılımın yapması için tasarlandığı işlemleri yaptığından, yapması beklenmeyenleri yapmadığından emin olmak için yapılan bir süreç ya da süreçler serisidir (Myers, 2004).

“Bir programın davranışını dinamik yöntemlerle, sonsuz bir küme içinden belirli sayıda seçilen test durumlarını kullanarak, beklenen davranışa uymadığı durumları

bulma işlemidir” (IEEE Standart Glossary of Software Engineering Terminology, 1990).

2.2. Yazılım Yaşam Döngüsünde Test

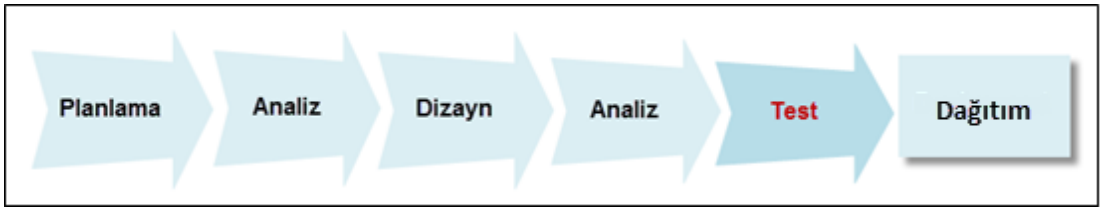
Yazılım Geliştirme Yaşam Döngüsü (SDLC), teknoloji tarafından kullanılacak bir kurulum olarak tanımlanabilir. Yazılımı tasarlamak, uygulamak, geliştirmek ve test etmek için kullanılır. Yazılım Geliştirme Yaşam Döngüsü yüksek kalite sağlar. Ürünleri müşteri beklentilerini düşük maliyetli bir şekilde karşılar (But, 2018).

Colin But'a göre Yazılım Geliştirme Yaşam Döngüsü, bir yazılım projesi veya bir yazılım organizasyonu sürecinin nasıl geliştirileceğini açıklayan ayrıntılı bir plandan oluşur. Yaşam döngüsü, yazılım kalitesini ve genel geliştirme sürecini iyileştirmek için kullanılan metodolojidir (But, 2018).

IBM’e göre yazılım yaşam döngüsü bir yazılım geliştirme projesinin yönetiminde tüm yönleri ile takımlara yardımcı olacak yazılım piyasasının ihtiyaçlarına uygun entegre araçları tanımlar. Özelleştirilmiş bir dizi araçlar sayesinde özelleştirilmiş kuralların yerine getirilmesini sağlayan bir dizi roller olarak tanımlanır (IBM, 2008).

2.2.1. Geleneksel/sıralı model

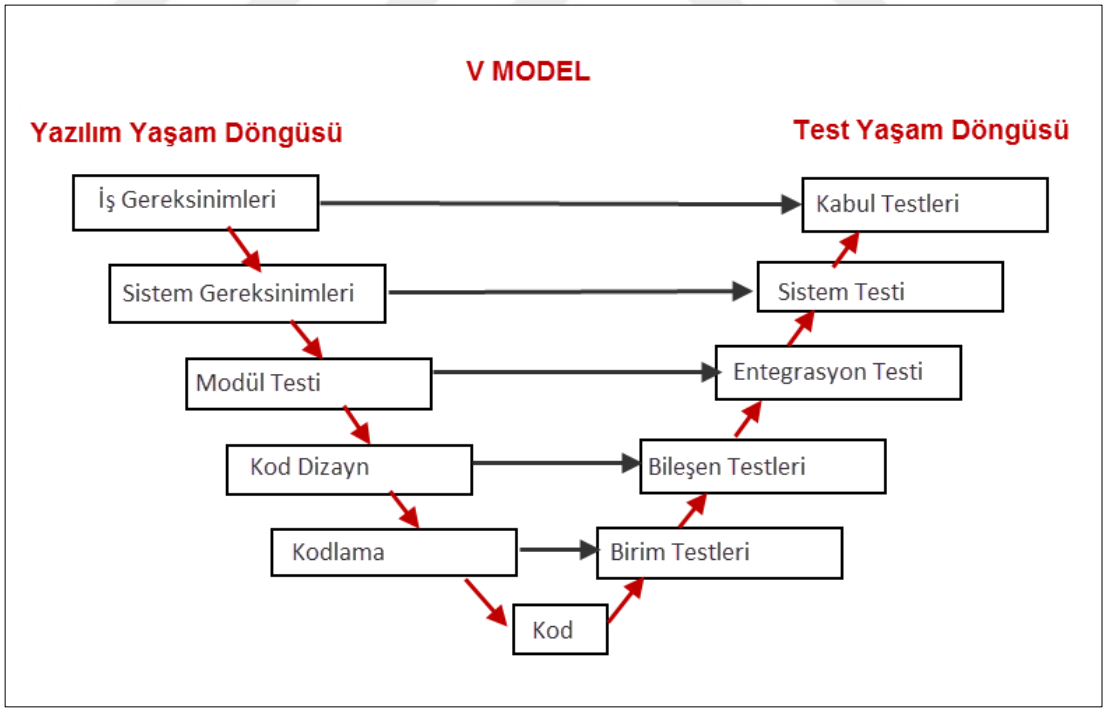
Bu modelde genel olarak süreç 6 adımdan oluşur: Planlama, analiz, dizayn, analiz, test, dağıtım (deployment) (Dayıbaşı, 2020). Bu modeli yazılım testi açısından değerlendirecek olursak verim düşüktür. Bunun sebebi yazılım testinin ve son kullanıcı kabul testinin Şekil 2.1.’de (Dayıbaşı, 2020) görüldüğü üzere son aşamada yapılmasıdır. Bu durum hata çözüm maliyetini artırır. Bazı analiz hataları bile farkedilmez ve bu adımda bulunabilir. Kullanıcıya ürün çıktıktan sonra ürünün gösterilmesi projenin anlaşılmayan gereksinimlerden dolayı projenin iptaline bile sebep olabilir.



Şekil 2.1. Geleneksel/sıralı model yaşam döngüsünde test (Dayıbaşı, O. 2020)

2.2.2. V-model

Bu yazılım yaşam döngüsü çeşidinde tüm süreç boyunca her basamakta test bulunmaktadır. Her basamakta yapılan testler Şekil 2.2.'de gösterilmektedir. Test ile ilgili planlama, dizayn gibi ön hazırlıklar geliştimeden önce başlar. Ön hazırlıkların geliştirmeden önce başlaması gereksinimler belirlendiğinde bazı hataların erken tespitini sağlar. Bu geliştirme öncesi bulunan hatalar maliyetten oldukça kazanım elde ettirir. Hata tespiti aşaması ilerledikçe hata maliyeti artar. Bu sebeple hata maliyetleri açısından verimli bir modeldir.



Şekil 2.2. V Model yazılım yaşam döngüsünde test (Dayıbaşı, O. 2020)

2.2.3. Şelale (waterfall) modeli

Şelale modeli, geliştirme süreçlerinde yaygın kullanılan ve bilinen bir modelidir.

Her aşama tek tek birbirini takip eder. Aşamalar; analiz, tasarım, uygulama, test, dağıtım ve bakımdır. Bu süreç tamamen önceden tanımlanmıştır. Başlamadan önce tüm şartname ve gelişim planı tanımlanmalıdır (Osetskyi, 2019).

Bu modelde yazılım süreci 6 adımdan oluşur;

1. Proje Planlama
2. Gereksinim Açıklama (Analiz)
3. Tasarım
4. Geliştirme
5. Entegrasyon ve Test
6. Kullanıcı Kabul Testi

Bu yazılım geliştirme modelinde, bir adım tamamlanmadan sonraki adıma geçilmez. Bu nedenle yazılım yaşam döngüsünde gereksinimlerin belirlenmesi, tasarımın gerçekleştirilmesi ve yazılımın tamamlanması aşamasından sonra tamamlanmış ürün test sürecine dâhil edilir. Bu yöntemin çeşitli avantaj ve dezavantajları vardır.

Şelale yazılım modelinde yazılım test sürecinin avantajları şunlardır;

– Test safhasına gelindiğinde yazılım süreci kesin olarak tamamlanmıştır. Test sürecinde yeni kodsal geliştirmeler yapılamaz. Bu durum, projenin test sürecine ek maliyet oluşturmayacağı gibi testin kalitesini de artırır.
– Dokümana önem verilen bu modelin her safhasında gerekli dokümanlar üretilir. Test sürecine girdi sağlayacak bilgi akışı sağlanmış olur.
– Modelin her safhası kendine özgüdür. Süreçler birbirinden bağımsız bir şekilde ilerler. Bu test sürecinin ideal akışının uygulanmasını sağlar.

Şelale yazılım modelinde yazılım test sürecinin dezavantajları şunlardır;

– Bu yaşam döngüsünde testin son adımda yer alması hata maliyetlerini büyük oranda artırmakta ve süreci uzatarak projenin başarısız olmasına sebep olabilmektedir.
– Son kullanıcı veya müşteri yapılan geliştirmeyi veya ürünü bittikten sonra görür. Bu durum projenin iptaline kadar varmakta ve ilişkileri zedelemektedir.
– Test son adımda olduğundan test sürecinde bulunan hataların düzeltimesi çok zor olur ve yapılan değişikliği adapte etmek zor olur.
– Tespit edilen gereksinim hataları projenin en başa sarmasına sebep olur. Bu temel hatayı düzeltmek proje kaynak ve bütçesinde büyük açıklar oluşturur.

2.2.4. Artırımlı model (iterative/incremental) model

Yaşam döngüsünün belli periyotlarda çıkarılacak iş veya ürünün belli bir kısmının oluşturulması için baştan sona çalıştırılmasıdır. Analiz, yazılım ve test belirlenen kısım için yapılır ve bitirilir. Projedeki işler fazlar veya sprintlere bölünerek yapılır. Her faz veya sprintte tüm yazılım yaşam döngüsü gerçekleştirilir. Fazların birbirleriyle olan bağımlılıklarını (dependency) yönetmek zordur.

Hızlı Prototipleme (rapid prototyping), Çevik (agile), Sarmal (spiral) bu modelin örneklerindendir.

2.2.4.1. Spiral Model

Bu modelde yazılım süreci 5 adımdan oluşur;

1. İçeriğin açıklanması
2. Risk analizi
3. Dizayn
4. Kodlama
5. Çalıştırma

Bu adımlar ürün çıkarılana kadar belli periyotlarda gerçekleştirilir.

2.2.4.2. Çevik (agile) model

Çevik yazılım yaşam döngüsü modelinde yapılacak geliştirme, sistem veya uygulama vb. üzerinde çalışılacak iş teslimat yapılabilecek parçalara bölünür. Bu teslimat parçaları müşterinin de dahil olduğu toplantılarda tüm paydaşlar ile birlikte öncelik ve önem sırasına göre gruplanır. Bu teslimat parçalarının her birinde bir nevi şelale model uygulanır tüm teslimatlar bittiğinde iş bitmiş olur.

"Çevik Test'in belki en değerli özelliği ise test sürecinde ürün sahibi ile birebir ilişki kurma imkanının olmasıdır. Analizden itibaren süreçte yer alan test mühendisi, öz nitelikteki (feature) eksiklik ya da muğlaklıklar ile ilgili durumları yüzyüze iletme imkanına sahiptir. Çevik proje yönetimi geri besleme özelliği taşıdığından doğru sonuca ulaşmada önemli bir yöntemdir" ("Agile (Çevik) Metod Ve Agile Test Yöntemi", 2016).

Çevik Model, firmanın veya kurumun pazara erken çıkmasını ve projenin daha emin adımlarla ilerlemesini sağlar. Çevik Model, dokümantasyon açısından çeşitli sıkıntılar oluşturabilmektedir. İşin teslimatlara ayrılmış olması bütünden kopmalara sebep olabilmektedir.

2.3. Yazılım Test Seviyeleri

Yazılım Test Seviyeleri, bir geliştirmenin test aşamalarını ifade etmektedir. Bu seviyelerde farkı rollerde kişiler yer almaktadır. Her test seviyesi, bir projede rollere göre sorumluluklara bağlanır. Yazılım Test Seviyeleri temelde 4 adımdan oluşur. Bu adımlar Birim Testleri, Entegrasyon Testleri, Sistem Testleri ve Kullanıcı Kabul Testlerinden oluşur.

Uygulamanın bulunduğu yazılım döngüsündeki ilerleyen aşamalara göre özel yapılan testler test seviyeleri kavramını oluşturmuştur. Test seviyeleri yazılım yaşam

döngüsünün baştan sona tüm evrelerinde hataları bulmayı sağlar. Test Seviyeleri hataların erken bulunup çözülmesinde en etkili ve sistematikleşmiş uygulama biçimidir.

Tablo 2.1. Yazılım test seviyeleri (ISTQB, 2018).

Birim Testi	Yazılımcı tarafından yapılır
Entegrasyon Testi	Yazılımcı ve Test Uzmanı tarafından yapılır.
Sistem Testi	Test Uzmanı tarafından yapılır.
Kullanıcı Kabul Testi	Son Kullanıcı/Müşteri tarafından yapılır.

2.3.1. Birim testi (unit test)

Birim testi sırasındaki hedeflerden biri, olabildiğince çok sayıda hata tespit etmek ve böylece bunlara neden olan hataların erkenden tespitini ve düzeltilmesini sağlamak olabilir. Diğer bir hedef, birim testinin kod kapsamını artırmak olabilir. Birim testleri genellikle kodu yazan yazılımcı tarafından gerçekleştirilir, bu da test edilecek koda erişim gerektirir. Yazılımcılar bu seviyede geliştirme yapma ve hataların bulunup giderilmesini sağlama arasında gidip gelebilirler. Yazılımcılar genellikle bir birimin kodunu yazdıktan sonra testleri yazar ve yürütür. Bununla birlikte, özellikle çevik yazılım geliştirmede, otomatikleştirilmiş birim test senaryoları yazmak uygulama kodunun yazılmasından önce gelebilir.

2.3.2. Entegrasyon testi

Entegrasyon testinde farklı, biribirne bağlı olmayan iki ayrı geliştirmenin, ürünün, uygulamanın, ekranların veya çeşitli küçük değişikliklerin aynı sistem üzerinde birleştirilmiş son hali test edilir. Entegrasyon testinde amaç yapılan bu entegrasyonda ortaya çıkabilecek hataları bulmaktır.

Birim entegrasyon testleri genellikle yazılımcıların sorumluluğundadır. Sistem entegrasyon testleri ise genellikle test uzmanlarının sorumluluğundadır. İdeal olarak, sistem entegrasyon testlerini yapan test uzmanları sistem mimarisini anlamalı ve entegrasyon planlamasında rol almış olmalıdır.

2.3.3. Sistem testi

Geliştirmenin belirlenen gereksinimleri sağladığını doğrulamak için uygulanan test aşamaları sistem testi olarak adlandırılır. Sistem testi, eksiksiz ve entegre bir sistemde gerçekleştirilir. Gereksinimlere göre sistemin uygunluğunun kontrol edilmesini sağlar. Bileşenlerin genel etkileşimini test eder. Yük, performans, güvenilirlik ve güvenlik testlerini içerir. Sistem testi, sistemin şartnameye uygun olduğunu doğrulamak için yapılan son testtir. Bu yüzden test için hem fonksiyonel hem de fonksiyonel olmayan gereksinimler değerlendirir.

Fonksiyonel gereksinimleri doğrulamak amacıyla yapılacak teste en uygun test kara kutu testidir (ISTQB, Sertifikalı Test Uzmanı Temel Seviye Ders Programı, 2018). Kara kutu testinde tespit edilemeyen hatalar için kara kutu testi sonrası beyaz kutu (white box) test teknikleri uygulanabilir. Testi yapılan bir geliştirmenin hataları bulma açısından canlı ortama en yakın şekliyle bir test ortamı olması çok önemlidir.

2.3.4. Kabul testi

Kullanıcı Kabul Testi müşterinin veya son kullanıcının yapılan geliştimede üzerinde anlaşılmış gereksimleri ve sistemin çalıştığını görmesi amacıyla yapılmaktadır. Kabul testinde asıl hedef hataları bulmak değildir, sistemin üretim ortamına hazır olduğunu göstermektir. Ancak ana hedef hata bulmak olmasa bile yazım hatalarından uygulamada büyük bir soruna neden olabilecek hatalara kadar birçok hata açığı çıkabilir. Kabul testleri genel olarak kullanıcı veya müşteri tarafından yapılır. Ancak, diğer hissedarlar da bu sürece dahil olabilir.

2.4. Yazılım Test Çeşitleri

Farklı alanlarda kullanılan çok sayıda yazılım test çeşidi bulunmaktadır. Yazılım test çeşitleri kullanım durumuna göre çeşitlendirilmiştir. Temel olarak Fonksiyonel Test,

Fonksiyonel Olmayan Test Çeşitleri şeklinde iki ana grupta sınıflandırılmıştır (ISTQB, Sertifikalı Test Uzmanı Temel Seviye Ders Programı, 2018).

2.4.1. Fonksiyonel test

Fonksiyonlar sistemin "ne" yaptığı sorusuna verilen cevaptır. Fonksiyonel test fonksiyonları ve dokümanlarda tanımlanan özellikleri esas alır. Güvenlik testi, birlikte çalışabilirlik testi fonksiyonel testin çeşitlerindendir. Bir geliştirme sürecinde yapılan fonksiyonel testler, geliştirme sonucunda gerçekleşmesi gereken fonksiyonlar test eden gereksinimleri doğrulamaktadır.

Fonksiyonel gereksinimler; iş gereksinimleri, kullanıcı gereksinimleri, kullanıcı hikâyeleri, kullanım senaryoları veya fonksiyonel spesifikasyonlar gibi çalışma ürünlerinde tanımlanmış olabilir. Fonksiyonlar gereksinimler sistemin ne yapması gerektiğini tanımlar. Fonksiyonel testler tüm test seviyelerinde yapılmalıdır (örneğin; birim seviyesindeki fonksiyonel testler ilgili birimin gereksinimlerine dayandırılmalıdır), ancak odağı her seviyede farklıdır.

Fonksiyonel testler yazılımın davranışını göz önüne alır; bu nedenle, birim veya sistemin fonksiyonalitesi için test şartları ve test senaryolarını belirlemede kara kutu testi teknikleri kullanılabilmektedir. Fonksiyonel testlerin bütünlük derecesi, fonksiyonel kapsam ile ölçülebilir. Fonksiyonel kapsam, fonksiyonel gereksinimin testlerle ne kadar ele alındığını belirtir ve gereksinimin yüzdesi olarak ifade edilir. Testler ve fonksiyonel gereksinimler arasındaki izlenebilirlik kullanılarak bu gereksinimlerin testlerde ele alınan yüzdesi hesaplanabilir ve böylece potansiyel olarak kapsam eksiklikleri belirlenebilir.

2.4.2. Fonksiyonel olmayan test

Bir geliştirmenin fonksiyonel olmayan testleri, sistemlerin ve yazılımların, uygulanılabilirlik, performans veya güvenlik gibi özelliklerini değerlendirir. Yazılım ürün kalitesi özelliklerinin sınıflandırması için ISO standardına göre (ISO/IEC 25010)

içeriği kontrol eder. Fonksiyonel olmayan testler, geliştirmenin gereksinimleri tamamen karşıladığını doğrulamaktadır. Geliştimenin fikir aşamasından itibaren fonksiyonel olmayan testler uygulanmalıdır ve sık sık tekrar edilmelidir. Erken başlayan fonksiyonel olmayan test en verimli fonksiyonel olmayan testtir. Erken başlamayan fonksiyonel olmayan test hataların vaktinde bulunmamasına sebep olacak ve proje başarısı kritik hale gelecektir.

Fonksiyonel olmayan testler için test şartlarını ve test senaryolarını oluşturmakta kara kutu test teknikleri kullanılması verimli olacaktır. Performans testi için stres senaryolarını belirlerken sınır değer analizi iyi bir teknik olacaktır. Fonksiyonel olmayan testlerin bütünlük derecesi, fonksiyonel olmayan kapsam ile ölçülebilir. Fonksiyonel olmayan kapsam, fonksiyonel olmayan bir gereksinimin testlerle ne ölçüde ele alınmış olduğunu belirtir ve kapsanan gereksinimin bir yüzdesi olarak ifade edilir. Örneğin, bir mobil uygulama için testler ve desteklenen cihazlar arasındaki izlenebilirlik kullanılarak uyumluluk testleriyle ele alınan cihazların yüzdesi hesaplanabilir ve böylece potansiyel olarak kapsam eksiklikleri belirlenebilir. Fonksiyonel olmayan test tasarımı ve koşumunda, bir tasarım veya teknolojinin yapısal açıdan zayıf yönleri (örneğin, belirli programlama dilleriyle ilgili güvenlik açıkları) veya belirli bir kullanıcı tabanı (örneğin hastane yönetim sistemi kullanıcı grupları) hakkında bilgiler gibi özel beceriler veya bilgiler yer alabilir.

2.4.3. Beyaz kutu testi

Beyaz kutu testleri, sistemin iç yapısını doğrulayan testleri oluşturur. Kod, mimari, iş akışları ve/veya sistem içindeki veri akışlarını dikkate alır. Beyaz kutu testlerinin bütünlük derecesi yapısal kapsam ile ölçülebilir. Yapısal kapsam, bir yapısal öğenin testlerle ne ölçüde ele alınmış olduğunu belirtir ve kapsanan öğenin yüzdesi olarak ifade edilir.

Birim testi seviyesinde, kod kapsamı test edilen birim kodunun yüzdesine dayanır ve birimde test edilen komutların yüzdesi veya test edilen kararların yüzdesi gibi kodun farklı yönleri (kapsam öğeleri) açısından ölçülebilir. Bu kapsam çeşitlerine toplu

olarak kod kapsamı denir. Birim entegrasyon testleri seviyesinde beyaz kutu testleri, birimler arasındaki arayüzler gibi sistem mimarisine dayanabilir ve yapısal kapsam, testler ile ele alınmış arayüzlerin yüzdesi olarak ölçülebilir. Beyaz kutu test tasarımı ve koşumu, kodun yazılma şekli (örneğin kod kapsamı araçlarını kullanmak), verilerin nasıl depolandığı (örneğin olası veritabanı sorgularını değerlendirmek), kapsam araçlarının nasıl kullanılacağı ve sonuçlarının doğru şekilde yorumlanması gibi özel bilgi veya becerileri içerebilir.

2.4.4. Değişiklik testi

Bir geliştirmede, mevcut bir hatayı çözmek için, yeni meydana gelmiş veya değişen gereksinimler sebebiyle değişiklikler yapıldığında, hatanın çözüldüğünü, değişen gereksinimin doğru bir şekilde çalıştığını veya başka kısımları etkileyerek bozmadığını, farklı hatalara sebep olmadığını doğrulamak amacıyla yapılan testler değişiklik testleridir.

Onaylama testleri: Bir hata çözüldükten sonra, hata nedeniyle başarısız olmuş tüm test senaryoları tekrar test edilebilir, bu testler yazılımın yeni versiyonunda yeniden koşturulmalıdır.

Hatanın fonksiyonalite eksikliğinden kaynaklanması durumunda, yazılımın test edilmesi için yeni testler de yazılabilir. En azından hatadan kaynaklanan hataları veya eksikliği yeniden oluşturmak için gereken test adımları, yazılımın yeni versiyonunda tekrar koşturulmalıdır.

Regresyon testleri: Geliştirmede yapılan bir değişikliğin geliştirmenin diğer kısımlarını bozması, sistemin diğer kısımlarına etki etmesi, iş akışını bozması, başka ekranlarda ortak kullanılan bir komponenti bozması gibi bazı istenmeyen durumlara sebep olması mümkündür. Değişiklikler, işletim sisteminin veya veritabanı yönetim sisteminin yeni bir versiyonu gibi ortamda yapılan değişiklikler de olabilir. Bu istenmese de bazı kısımları bozan duruma regresyon denir (ISTQB Sertifikalı Test Uzmanı Temel Seviye Ders Programı, 2018)

Regresyon testleri, bu gibi bozulmuş, başka yerelere etkisi olmuş düzeltmelerin etkisini görmek için yapılan testlerdir. Tüm test seviyelerinde regresyon testleri ve onaylama testleri yapılabilir.

Özellikle çevik modelde, döngüsel ve artımlı yazılım geliştirme yaşam döngülerinde yeni özellikler eklenmesi, mevcut özelliklerde yapılan değişiklikler ve geliştirmenin yeniden düzenlemesi, kodda sürekli değişiklik yapılmasını gerektirir. Bu da değişiklikle ilgili testler gerektirir. Sistemin sürekli evrilen yapısını kontrol altında tutmak amacıyla onaylama ve regresyon testleri çok önemlidir. Bu durum nesnelerin sıklıkla güncellendiği veya başkasıyla değiştirildiği Nesnelerin İnterneti Sistemleri (IOT) için özellikle önemlidir.

Regresyon testi grupları yazılım geliştirme yaşam döngüsü içinde birçok kez koşturulur ve genellikle fazla değişikliğe uğramazlar, bu nedenle regresyon testlerinin en çok kullanılabileceği alan otomasyon testleridir. Bu testlerin otomasyonu projenin erken aşamalarında başlamalıdır.

BÖLÜM 3. HATA YÖNETİMİ

Yazılım testi literatüründe yaygın olarak kullanılan bug, defect, error, fault terimleri dilimizde "hata" olarak karşılanır. ISTQB Yazılım Testi Terimler Sözlüğü'nde hatanın tanımına şu şekilde yer verilmiştir: "Bir bileşen ya da sistemin gerekli işlevini gerçekleştirmesini engelleyen kusur. (Örneğin doğru olmayan komut veya veri tanımlaması). Hata, bileşen ya da sistem çalışırken ortaya çıkarsa arızaya neden olabilir" (Yazılım Test ve Kalite Derneği, 2014).

Hata kelimesinde ziyade böcek (bug) kelimesi sıkça kullanılır hale gelmiştir. Debug kelimesinin yaygın oluşu kadar bug kelimesi de yaygındır. Bug İngilizce bir kelimedir ve Türkçe anlamı böcektir. Böceğin küçük oluşu ve ilk bakışta görülmemesi, hatta belirli nesneler yardımı ile olduğu yerden çıkarılması bu terimin yazılımda kullanımını desteklemiştir.

Yazılım dünyasında "bug" kelimesi de hatalar için kullanılmaktadır ve defect'le eşanlamlı olduğu düşünülebilir ancak İngilizce'de "böcek" anlamında kullanılan bu tabir bir bilgisayar programında kodlama hatası olarak kullanılmaktadır. Bu terim, 1947 yılında Harvard'da Hesaplama Laboratuarında Mark II isimli bilgisayarda iki elektrikli röle arasında gerçek bir böceğin -bir güve olduğu söylenmektedir- hataya yol açmasından sonra bilgisayar dünyasında ilk "bug"ın gerçek bir böcek tarafından kaynaklanmasından sonra kullanılmaya başlanmıştır. Grace Hopper (programlama dili için ilk derleyiciyi yapan Harvard Mark II çalışanı) "o andan itibaren, bilgisayarda bir şey ters gittiğinde bunun içinde bug olduğunu söyledik" der.

Yazılım testi altındaki birçok alan IEEE 610 ve IEEE 982 standartları kapsamında ele alınmıştır. Defect kavramının da yazılım dünyasında daha anlaşılır olması için IEEE

610.12-90 (IEEE Standard Glossary of Software Engineering Terminology) standardı hazırlanmıştır. Bu tanımlar:

- Hata (error): Hesaplanan sonuç ile doğru sonuç arasındaki fark.
- Yanlışlık, hata, kusur (fault): Bir bilgisayar programında yanlış bir adım, işlem veya veri tanımı.
- Başarısızlık (failure): Bir hatanın [yanlış] sonucu.
- Hata (mistake): Yanlış bir sonuç üreten bir insan eylemi.

Yazılımın doğrulanması ve geçerlenmesi süreci, birçok alt faaliyeti içinde barındırır. Uygun sonuçlar elde edilene kadar yazılım ürününün teknik yeterliliği değerlendirilir ve bu aşamadaki çıktıların ürünün kalitesine katkısı büyüktür (Şahinoğlu ve ark., 2013).

Hata yönetimi ile amaç, hataların mümkün olan en yüksek seviyede önlenmesini sağlamakla birlikte hataları yazılım geliştirme yaşam döngüsünün erken safhalarında tespit etmek ve tespit edilen hataların raporlanması ve etkin bir hata çözüm sürecinin takip edilmesini sağlamaktır. Hata yönetim sürecinde kabul edilen temel prensipler aşağıda belirtilmiştir (Ergin, 2020).

- Hatanın oluşmasını önlemek birincil amaçtır. Bu koşulun sağlanamadığı durumlarda hatayı erken safhada bulmak önemlidir.
- Hata yönetim süreci risk odaklıdır. Riski düşürme merkezli strateji belirlenir, önceliklendirme yapılır.
- Hata yönetim süreci yazılım geliştirme süreci ile bütünleşiktir.
- Bilginin toplanması ve hatanın analiz edilmesi mümkün olduğunca otomatikleştirilir.
- Hata raporlamasında yer alan bilgiler süreci geliştirmek için kullanılır.
- Hataların oluşmasına sebep olan kusurlu süreçler iyileştirilir.
- Hatalar gereksinim ve test senaryolarına bağlı olarak izlenebilir olmalıdır.

3.1. Hata Raporlama

Yazılım Testinde tespit edilen hataların çözüm sürecinin takibi için ortak bir platform üzerinden ilgili kişiye iletilmesine Hata Raporlama denir (Molu ve ark., 2016).

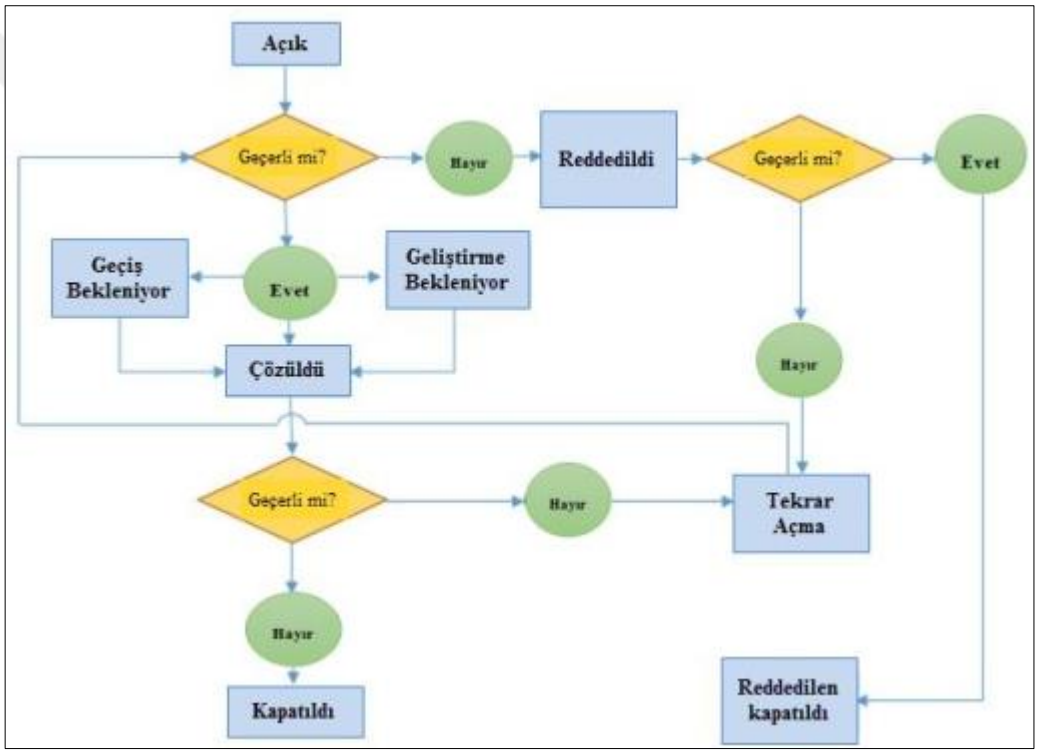
Hata yönetim sürecinin kalitesini tespit edilen hataların özelliklerinin en iyi şekilde belirlenmesi ve kaliteli raporlanması belirler (Kolkhorst, 2000). Aynı zamanda proje ve ürünün performansını değerlendirmek için girdilerin özelliklerinin doğru belirlenmiş olması kritiktir. Yapılan geliştirmelerde geçmiş sürümlerde bulunan hata sayılarını raporlamak ve değerlendirmek yazılım kalitesi hakkında bilgi verebilir. Ancak kalite yönünde yapılan ölçüm yalnız hata adedi bazında olursa bu ölçüm sağlıklı olmayabilir. Hatanın önceliği, ciddiyeti ve daha önceki hatasebeplerinin yer aldığı bir ölçüm kalite hakkında daha doğru bilgi verecektir (Öztürk, 2016). Hatanın başlıca özellikleri arasında statüsü, tipi, tekil olması, kritiklik seviyesi, öncelik seviyesi, hata kaynağı, tekrarlanabilir olması sayılabilir. Bunun yanında bir hata raporunda, hatanın kim tarafından, ne zaman, hangi ortamda, uygulamanın hangi versiyonunda tespit edildiği bilgisi yer almalıdır. Hatanın geliştirici ekibe hata detayını gösteren bir ekran görüntüsü veya video kaydıyla iletilmesi de hatanın çözüm sürecini hızlandıracaktır. Hataya ilişkin bir log kaydı, bir doküman çıktısı gibi ek özellikler var ise bunlar da hata ile birlikte ilgili ekibe raporlanmalıdır (Cirit, 2010).

Hataların yazılım geliştiricilere bildirilmesi, bildirilen hataların düzeltildikten sonra hatanın kapatıldığının izlenmesi testler açısından önemlidir. En üst düzeyde hatalardan arındırılmış bir yazılımın geliştirilmesi isteniyorsa testler sırasında çıkan tüm hataların kapatıldığından emin olunmalıdır. Bulunan hatalar, hataların kaynakları, yapılan düzeltme faaliyetleri kontrol altına alınmalıdır.

Bir hata raporu veya hata kaydı şu detay maddeler ve benzeri detayları içermelidir: Hata No, Hatanın Detayı, Yazılımcı, Analisti, Hatanın Açılış Tarihi, Hatanın Kapanış Tarihi, Testin Yapıldığı Ortam, Hatanın Tipi, Atanan Kişi, Atayan Kişi, Hata Senaryosu, Hatanın Öncelik Durumu, Önem Durumu, Hatanın Bulunduğu Ortam (Masaüstü, Mobil gibi) ve benzeri.

Kayıt altına alınan bu hatalar çeşitli metrikler belirlenerek bazı analizler yapılır. Bu analizler ekip bazlı, müdürlük bazında veya tüm Bilgi Teknolojilerini kapsayacak şekilde yapılabilir. Çeyreklerde veya yarı dönemlerde raporlar oluşturulabilir. Bu yapılan analizlerle birçok alanda iyileştirmeler yapılabilir. Sorunlar tespit edilerek hatalar en aza indirgenebilir veya süreçler daha kısa hale getirilebilir ("Defect Management Process", 2020).

3.1.1. Hata yaşam döngüsü



Şekil 3.1. Hata yaşam döngüsü akış diyagramı (Molu ve ark., 2016).

Şekil 3.1.'de bir hatanın yaşam döngüsü görülmektedir. Hata yaşam döngüsünde önce hata tespitinin yapılmasıyla birlikte hata raporu oluşturulur. Açılan hata başta açık statüsünde olur. Hatayı çözmekle sorumlu kişi hatanın geçerliliğini kontrol eder. Hata, hata değilse hatayı reddedildi statüsüne günceller. Hata kaydını oluşturan kişi reddedildi statüsünü doğrularsa hatayı reddedilen kapatıldı statüsüne günceller. Reddedildi statüsünü doğrulamazsa hatayı tekrar açık statüsüne günceller. Hatanın atandığı kişi hata geçerliliğini doğrularsa hatayı çözer ve çözüldü statüsüne günceller.

Hatayı raporlayan kişi çözümü kontrol eder ve geçerliliğini onaylarsa hatayı kapatıldı statüsüne günceller.

Bir hata, hata yaşam döngüsünde çeşitli statülerde gösterilir. Hatanın açılması, çözülmesi ve kapatılması bir hatanın temel statüleridir. İhtiyaca göre veya yazılım geliştirme modeline göre geliştirme bekleniyor, reddedildi, askıya alındı gibi çeşitli statüler de hata yaşam döngüsünde kullanılmaktadır.

Kapatıldı (closed): Çözülerek kapatılan hatalardır. Hata kaydı çözüldü statüsünden kapatıldı statüsüne çekilmiştir.

Reddedilen kapatıldı (closedFromRejected): Hatanın atandığı kişi tarafından reddedilen hata, hata girişini yapan kullanıcı tarafından bu statüye getirilir. Ret sebebinin geçerli olup kabul edildiğini gösterir.

Geçiş bekleniyor (deploymentPending): Yapılan düzeltmenin geliştirme ortamında yapıldığını, test ortamına geçiş işleminden sonra test edilebileceğini gösterir. Örnek uygulamamızda test ortamları Geliştirme ortamlarından belirli saatlerde güncellenmektedir. Bu nedenle bu ara statüye ihtiyaç duyulmuştur. Bu statünün kullanılmasıyla birlikte hatanın ne zaman çözüldüğü, açık kalma süresi verilerinin sağlıklı alınması sağlanmıştır.

Çözüldü (fixed): Hatanın çözülerek tekrar test edilmesi için düzeltmenin test ortamına geçildiğini gösteren statüdür.

Geliştirme Bekleniyor (ImprovementPending): Hatanın çözümü için proje kapsamında yapılan geliştirmelerin haricinde ek Geliştirme gereken, bu geliştirmenin değişiklik yönetimi veya proje yönetimi süreçleriyle takip edilmesi gerektiği durumlarda kullanılan statüdür.

Reddedildi (rejected): Hatanın iş analisti tarafından geçerli sayılmayıp reddedildiği statüdür. İş kuralları gereği hatanın geçerli sayılmadığı durumlarda bu statü kullanılır.

Yazılımcı tarafından reddedildi (rejectedByDev): Hatanın yazılım geliştirici tarafından geçerli sayılmayıp reddedildiği statüdür. Hatanın istenen şekilde çözümünün teknik olarak mümkün olmadığı veya hatanın geçerli olmayıp hata olarak belirtilen durumun teknik bir özellik olduğu durumlarda bu statü kullanılır.

Hataların reddedilme nedenlerinden bazıları: tekrarlanamayan hata senaryosu, test mühendisinin hata olmayan bir durumu (özellik, iş gereksinimi vs.) bildirmesi, değişen iş ihtiyaçları, test ortamı ve test verisi kaynaklı durumlar.

Tekrar açık (reopen): Reddedilen hatanın, hatayı giren kişi tarafından ret sebebinin geçerli sayılmayıp tekrar açıldığı durumlarda hata "reopen" statüsüne getirilir.

Yazılım projesinde kullanılan yazılım geliştirme metoduna, proje ekibinin iletişim yöntemine veya organizasyonel yapıya uygun olarak genel döngüyü etkilemeyecek şekilde farklı statüler de eklenebilir. Örneğin, hatanın çözülmesinden önce "geçiş bekleniyor" veya "geliştirme devam ediyor" gibi ara statüler hata statüleri arasındaki geçiş sürelerinin raporlandığı olgunluktaki kurumlar için ortaya çıkan bir ihtiyaçtır.

Hata yönetimi sürecinde, her hatanın mevcut ve önceki statüleri kayıt altına alınır. Hata üzerinde yapılan değişiklikler projenin bilgi birikiminin önemli bir parçasıdır. Statü değişimlerinde yapılan değişiklikle ilgili yorum veya açıklama yazılması sürecin kişiden bağımsız olarak tekrarlanabilmesini sağlar.

Hata yaşam döngüsünde önemli olan bir diğer nokta, statüler arasındaki geçişlerin belirli kurallara bağlanmış olmasıdır. Örneğin Açık (open) statüsündeki bir hata direkt olarak Kapalı (closed) statüsüne getirilmesine izin verilmemelidir. Sistemsel olarak bu ve benzeri kuralların işletilmesi sürecin sağlıklı uygulanması için önemlidir. Kuralların sistemsel kısıtlamalar ile sağlanması insan kaynaklı hataların minimize edilmesini sağlar. Benzer şekilde, proje ekibindeki kişilerin görevlerine uygun olarak sahip oldukları rolün gerektirdiği yetkilere sahip olması ve buna bağlı olarak hatayı ilgili statüye getirebilmesi önemlidir. Örneğin bir hata sadece hatayı açan test mühendisi veya iş analisti tarafından kapatılabilmeli, yazılım geliştiricinin hatayı

çözdükten sonra kapatma yetkisi olmamalıdır. Bu şekilde sürecin verimli ve kaliteli işletilmesi sağlanmış olur.

3.2. Yönetimsel Hata Raporlama

Yönetimsel Hata Raporlama, proje yönetimine veya organizasyondaki ihtiyaca göre iş yöneticilerine süreç geliştirme, proje yönetimi, performans değerlendirme gibi konularda girdi sağlaması için yapılan raporlamadır (Ergin, 2020).

Hata önleme: Hatanın kök sebeplerinin analiz edilerek hatanın oluşmasına sebep olan durum, kaynak veya süreçlerin iyileştirilmesi, değiştirilmesi, hata riskini azaltacak yöntemlerin belirlenmesini içerir.

Ana rapor oluşturma: Değişikliklerin kontrol altına alınması için detaylı bir şekilde oluşturulan rapor ile hatanın kabul kriterleri belirlenmiş olur.

Hata bulma: Hatanın tespit edilerek geliştirici tarafından tekrar üretilmesini sağlayacak bilgileri kapsayacak şekilde raporlanması aşamasıdır.

Hata çözümleme: Tespit edilen ve geliştiriciye bildirilen hataların düzeltilmesi ve çözüldüğünün kontrolüyle birlikte onaylanması safhasıdır.

Süreç geliştirme: Hata bazında inceleme ve analiz yapıldıktan sonra, benzer hataların tekrar oluşmaması adına süreçte yapılan iyileştirme çalışmalarını kapsar.

Hata sayısına göre değerlendirme: Test ortamındaki toplam hata sayısı tek başına yazılımın kalitesini ölçümlemek için yeterli bir veri değildir. Bulunan hataların kritiklik seviyesi, testin kapsama oranı gibi diğer verilerle birlikte değerlendirilmelidir (Boehm ve ark., 1998).

Hataları kritik seviyelerine göre değerlendirme: Test ortamında tespit edilen hataların kritiklik seviyeleri geliştiricinin performans göstergesi olmakla birlikte, kritiklik

seviyesi yüksek bir hatanın yazılım gerçek ortama taşınmadan önce test ortamında çözülmüş olması hedeflenen bir durumdur. Bunun yanında kritik hataların test ortamında tespit edilip çözülmesi, projenin gerçek ortama mümkün olan en hatasız şekilde geçmesi için güven vermektedir. Hataların kritiklik seviyesi aynı zamanda hatanın çözümünün ne kadar hızlı olması gerektiği ve hatanın tespit edildiği alanda nasıl bir etki analizi yapılması gerektiğini gösterir. (Molu ve ark., 2016).

Hata kaynağına göre değerlendirme: Hatalar, hataların oluşmasına sebep olan kök sebeplerine göre analiz edilerek hata yönetim süreci ve ilişkili diğer süreçler iyileştirilebilir. Bir hatanın ortaya çıkması yazılım yaşam döngüsünün farklı safhalarında olabilir ve hatanın oluşması ve tespiti farklı safhalarda gerçekleşebilir. Bu nedenle hatanın kaynağının tespiti hızlı karar verilebilen bir aktivite değildir. Hatayı giren kişiden sonra hatayı çözen kişinin detaylı inceleyerek sınıflandırma yapması beklenir.

Hata tiplerine göre değerlendirme: Fonksiyonel hatalar: Fonksiyonel hata tipinde uygulamanın fonksiyonalitesindeki eksiklik veya yanlışlıklıklar raporlanır.

Kullanıcı deneyimi (UX) hataları: Kullanıcı deneyimiyle ilgili yaşanan aksaklıklar bu hata tipinde raporlanır. Organizasyonel olarak uygulama bazında kullanıcı deneyimi görsel standartlarını belirleyen bir birim mevcuttur. Bu birimin belirlediği standartlara uyumsuzluk sağlayan durumlar bu kategoride raporlanır.

Tasarım hataları: Arayüz ve sistem tasarımıyla ilgili hatalar bu hata tipinde raporlanır.

Metin hataları: Uygulamalardaki kullanıcı bilgi ve hata mesajları, içerik ve isimlendirme gibi her türlü metinsel hata bu hata tipinde raporlanır.

Mimari uyum hataları: Uygulamanın belirlenmiş mimari uyum standartlarını karşılamadığı durumlar bu kategoride raporlanır.

BÖLÜM 4. KULLANILAN TEKNOLOJİLER

4.1. Alt Yapı ve Yazılım

Geliştirilen Hata Yönetimi uygulamasında yazılım dili olarak C#, teknoloji olarak WPF ve MVC, entegre geliştirme ortamı (IDE) olarak Visual Studio, web tarafı için .NET Framework ve veritabanı olarak SQL kullanılmıştır.

Yeni hata yönetimi mevcut kullanımda olan sistem üzerine ekranlar halinde eklenmiştir. Bu uygulama İş odaklı mimari uygulamasıdır (BOA). BOA'da kullanılan başlıca teknolojiler Microsoft .Net Framework, Windows Communication Foundation, Windows Presentation Foundation, IIS & Windows Process Activation Service, Microsoft Visual Studio, Microsoft SQL Server, Windows Client and Server, Citrix NLB'dir.

İş Odaklı Mimari (BOA) bir kurumsal mimari stratejisidir (Hurwitz J.S., Bloor R., Baroudi C., 2006, s.8). Bu stratejide iş modellerinin aynı sistem ve aynı yapıda uyum içinde birbiriyle çalışması temel amaçtır. Bu mimaride iş yapısı, iş kuralları baz alınarak tasarım ve uygulama yapılır. Türkiye'de katılım bankaları arasında en yüksek aktif büyüklüğe sahip olan Kuveyt Türk, mevcutta 5000'e yakın ekranı içinde barındıran BOA'yı hayata geçirmiştir (Kuveyt Türk Hakkımızda, 2020). Kuveyt Türk bu ölçekte banka olması sebebiyle bu kritiklik ve işlem yoğunluğu seviyesinde yapılacak bir proje altyapısının bağımsız bir laboratuar ortamında, tamamen bilimsel yöntemlerle test edilmesi zorunluluğuna inanarak, Microsoft firması sponsorluğunda, Ekim 2009 tarihinde, Almanya'nın Münih kentinde, Microsoft'un 3 birleşik kıtada bulunan en büyük laboratuarında bir performans çalışması yapmıştır. Yapılan testler sonucunda saniyede 2.950 işlem adeti gibi etkileyici bir sonuç alınmıştır. Ayrıca her bir işlemin saniyenin 35'te biri kadar bir cevap süresince gerçekleştirildiği sonucuna

varılmıştır. O tarihlerde Kuveyt Türk'ün mevcut sisteminde, günün en yoğun saatlerinde, aynı anda saniyede en fazla 30-40 işlem yapıldığı gerçeğiyle karşılaştıracak olursak, ulaşılan bu rakamlar, geliştirilen yeni yazılım altyapısının geleceğe yönelik artan kapasiteyi de kaldırabilecek isabetli bir karar olacağına inanılmış ve öyle olduğu da şu anki mevcut işleyişte görülmüştür. Kurumsal Mimari tarafından geliştirilen yazılım altyapısının ve geliştirme yöntemlerimizin doğruluğunun sağlanması, performansının Kuveyt Türk ortamında gözlenebilmesi için bir dizi pilot çalışması yapılmış ve daha sonra devreye alınmıştır ("Business Oriented Architecture (BOA) Hakkında", 2014).

BOA, iş hedefleri, müşteri beklentileri ve iş zorluklarının tanımları tarafından yönlendirilir ("Is BOA The New SOA?", 2020). BOA sistemleri, müşterilerin çeşitli teknolojileri ve işlevleri entegre ederek "tüm iş akışlarını veya işletim modellerini tasarlamasına, dağıtmasına ve çalıştırmasına" izin verir ("Industry making the transition to Business Oriented Architecture, 2020").

4.1.1. C sharp (C#)

C Sharp proğramlama dili genel amaçlı ve nesne tabanlı bir programlama dilidir. Microsoft tarafından tasarlanmıştır ve .NET platformuyla birlikte geliştirilmiştir. C # dili ile ve .NET üzerinde tasarlanmış çeşitli yazılımlar, masaüstü uygulamaları, web uygulamaları, ofis uygulamaları gibi platformlar; web siteleri, oyunlar, mobil uygulamalar ve birçok uygulama vardır (Skeet, 2013).

C Sharp'ın en önemli özelliği ve avantajı nesne tabanlı bir programlama dili olmasıdır. C, C++ ve Visual Basic dillerinden sonra Mikrosoft elde edilen tecrübe ile birlikte C Sharp'ı geliştirmiştir. Bunun da sonucunu en güzel şekilde almıştır. Dünyada en yaygın kullanılan proğramlama dilleri arasından ilk sıralarda yer almaktadır.

Güvenlik açısından donanımlı ve üst seviyelerdedir. Kullanıcıya sınıf (class) ve söz dizimi (syntax) hatalarını farklı bir ekranda detaylı olarak göstermekte ve düzeltme imkanı vermektedir.

4.1.2. Windows sunum temeli (WPF)

WPF (Windows Presentation Framework) Mikrosoftun .NET yapısı altında geliştirdiği grafik altyapısı oluşturan XAML tabanlı masaüstü uygulamalar geliştirmek için ürettiği bir teknolojidir. Bu teknoloji sayesinde C# geliştiricisi yapmış olduğu geliştirmenin görsel yapısına, özelliklerine her aşamada şekil verebilmektedir. İçinde bulunan standart tasarımların dışına çıkmak, onları değiştirmek kısıtlı olsa da tasarım ile arka planın birbirinden ayrılmış olması geliştirmede ve değişkliklerde büyük kolaylık sağlamaktadır (Mikrosoft, 2019)

4.1.3. Model görünüm denetleyici (MVC)

MVC (Model-View-Controller), 1979 senesinde Trygve Reenskaug tarafından tasarlanmıştır. Ortaya çıktığı ilk zamanlarda adı "Thing-Model-View-Controller" sonradan basitleştirilerek şu an ki halini almıştır. MVC (Model-View-Controller), yapılan geliştirmenin, uygulamanın iş kuralları ile kullanıcı arayüzünü ayrı yerlerde tutulmasını sağlayan, uygulamanın farklı özelliklerde olan kısımlarının birbirine karışmasını engelleyen bir yazılım mimari teknolojisidir.

MVC'de bir uygulamanın arayüzü 3 bölümden oluşur. Bu bölümler şu şekildedir;

Model: Veri işlemlerinin, gereksinimlerin, iş kurallarının tanımlandığı ve sınıfların oluşturulduğu kısımdır.

Görünüm (view): Kullanıcı arayüzünün tasarlandığı kısımdır.

Denetleyici (controller): Kullanıcı arayüzündeki işlemler ile model dediğimiz gereksinimlerin veya iş kurallarının arasındaki iletişim sağlayan kısımdır.

Arayüz ve iş kurallarının ayrı yerlerde kullanılıp çağırılarak kullanılması değişiklik gerektiğinde bunu ilgili kısımda rahatça yapmayı sağlar. Ayrıştırılan kısımlar farklı

uygulamalardan da çağırılıp kullanılabilir. Bu yapıda hatalar az çıkar. Bu yapı zaman ve efordan kazanç sağlar.

4.1.4. Visual studio geliştirme ortamı

Visual Studio, Mikrosoft'un ürettiği birden fazla programlama diline destek veren ve programlar, uygulamalar, web servisleri veya web sitesi yapılabilen bir IDE yani entegre geliştirme ortamıdır. Azure için bulut uygulamalar yapma imkanı da vermektedir. Visual Studio, üst seviyede fonksiyonel ve kullanıcı deneyimi ön plana çıkmış bir arayüz tasarımı sayesinde geliştiricilerin tercih ettiği bir platformdur. Bu derece işlevsel ve kullanışlı arayüz Visual Studio geliştiricilerinin eforunu oldukça azaltmaya elverişli bir kod editörü, hata bulucu (debugger), GUI tasarım aracı, veri tabanı şema tasarlama aracı ve ön güncelleme kontrol sistemleri sunmaktadır. Visual Studio yazılımının aynı zamanda açık kaynak olarak geliştirmeye açık ücretsiz sürümü de mevcuttur. Desteklediği programlama dillerinin başlıcaları şöyle sıralayabiliriz:

- Microsoft Visual C++
- Microsoft Visual C#
- Microsoft Visual Basic
- Team Foundation Server
- C, C++, C++.NET,
- Visual Basic.NET
- JavaScript
- CSS, HTML

4.1.5. Mikrosoft .net çerçevesi (.net framework)

Mikrosoft .Net Çerçevesi, bir yazılım geliştirme platformudur. Mikrosoft tarafından geliştirilmiştir. Windows uygulamaları oluşturmak ve çalıştırmak için kullanılmaktadır. Microsoft .Net Çerçevesi, masaüstü ve web uygulamaları oluşturmak için geliştirici araçları, programlama dilleri ve alabildiğine çok kütüphanelerden oluşur. Bu çerçevede web siteleri, web hizmetleri ve çeşitli oyunlar da yazılabilmektedir. Mikrosoft alt yapısını kullanan her uygulama, program oyun için

ortak bir dil oluşturularak birden fazla yazılım dilinin aynı platformda kullanılmasına olanak sağlamaktadır. Mikrosoft .Net Çerçeve, Windows Platformunda çalışacak uygulamalar oluşturma uzun zaman önce başlatmış ve .Net Çerçevenin ilk sürümünü 2002 yılında piyasaya sürmüştür. Böylelikle ilk sürüm olan .Net Çerçeve 1.0 piyasaya sürülmüştür. Microsoft .Net Framework hem Form tabanlı hem de Web tabanlı uygulamalar geliştirmeye uygun bir yapıya sahiptir. Web servisleri de .Net Çerçeve kullanılarak geliştirilebilir.

Mikrosoft .Net Çerçevenin avantajlarını sıralayacak olursak;

- En büyük avantajı kendisinin de yazılmasının en büyük amacı olan birden fazla proğramlama dilinin birbiriyle etkileşimini sağlamasıdır.
- Nesne yönelimli olması yenilenen senaryolar kolaylıkla çağırılır ve güncellenir.
- Büyük zaman tasarrufu sağlar ve maliyetten büyük kar getirir.
- Kendi içinde hata yakalama özelliğine sahiptir.
- Sızma vb. güvenlik açısından sağlam bir yapısı vardır.
- Kod yazılımı açısından basit bir yapıya sahiptir.
- Çok fazla sayıda hazır kütüphaneler içerir.
- Kodların kontrol edilebilirliği, tekrar kullanılabiliriliği açısından avantajlıdır.

4.1.6. Sql veritabanı yönetimi sistemi

SQL, Mikrosoftun geliştirdiği bir veritabanı yönetim sistemidir. Temel anlamda verilerin depolanmasını sağlar. İşlevsel anlamda çok büyük veriler de dahil olmak üzere verilerin yönetilmesini, istenilen şekilde tasarlanmasını sağlar (Demirli, 2008).

SQL veriler üzerinde birçok işlem yapılabilir. Bunlardan başlıcaları aşağıda sıralanmıştır:

- Verileri kaydetme, silme, güncelleme,
- Veritabanları oluşturma,
- Veritabanları altında birçok tablo oluşturup birbirine bağlama,
- Veri tekrarlarını engelleme,

- Veritabanına veri çekme,
- Veritabanındaki verileri istenilen şekilde filtreleme,
- Verileri yedekleme.

BÖLÜM 5. YENİ BİR HATA YÖNETİMİ UYGULAMASI TASARIMI

Geliştirilen Hata Yönetimi uygulaması firmanın kendi uygulaması içinde yer almaktadır. Ayrı bir giriş ekranı yoktur. Proje ve müşteri talep ekranları ile entegre çalışmaktadır. Yazılım Yaşam Döngüsü içindeki adımların büyük bir kısmı bu ekranlarda gerçekleşir. Müşteri talebini buraya bağlı ekranlarda oluşturur. Kapsam analizi onayından sonra ilgili analist burada talebin onaylanmış kapsamına göre analizini oluşturur. Akışı geliştirme adımına ilerletir. Bu ekranda analizi, inceleyen test mühendisi sadece gereksinimler üzerinden veya gereksinimler ile birlikte kendi eklediği senaryolar üzerinden test senaryolarını uygular. Yine buradaki gereksinim ve analize göre yazılımcı kodlarını yazar, geliştirmeyi yapar. Yazılımcı geliştirmeyi bitirdikten sonra akışı test adımında ilerletir. Test senaryoları onaylanır ve akış canlıya geçiş için test mühendisi tarafından ilerletilir, onaylanır.

5.1. Veritabanı Tabloları

Geliştirilen hata yönetimi veritabanında gereksinim tablosu, senaryo tablosu, hata tablosu, hata şablonu tablosu ve hata tarihçesi tablosu bulunmaktadır. Geliştirilen Hata Yönetimi mevcut bankacılık sistemine uygulanmıştır. Mevcut yazılım yaşam döngüsü akışı içine yerleştirilmiştir. Bağlı olduğu birçok tablo bulunmaktadır. Örneğin çalışan listesi/hatanın atanacağı kullanıcılar hata atanma alanında gelmektedir. Mevcut çalışan tablosu kullanılmış olup yeni bir kullanıcı tablosu oluşturulmamıştır. Modül Ağacı ayrı bir tabloda tutulmakta ve farklı birçok ekranda kullanılmaktadır. Gereksinimler İş Paketi Tablolarına bağlıdır. Dolayısıyla oluşturulan tablolar iş paketi tabloları ile de ilişkilendirilmiştir. Buna benzer birçok tablo bağlantısı içerdiğinden veritabanı diyagramı alınamayacak büyüklüktedir. Bu bağlamda geliştirmenin ana tabloları aşağıda ele alınmıştır.

5.1.1. Gereksinim tablosu

Gereksinim Tablosunda, Gereksinim Tanımlama ekranına ait işlemler tutulmaktadır. Kaydedilen gereksinimler tarih ve proje bazlı tutulmaktadır. Şekil 5.1.'de görülen Gereksinim tablosu alanlarında ProjectId ile proje tablosu referans alınmaktadır. Güncellenen gereksinimler UpdateUserName, UpdateHostName, UpdateSystemDate alanları ile tabloya yansıtılmaktadır (Şekil 5.1.).

Kaydedilen gereksinimler bu tabloya kaydolmaktadır. Bu tablo RequirementId, ProjectId, Description, StatusId, TypeId, PlanEffort, ResourceId, ResponsibleWorkGroupId, UserName, HostName, SystemDate, UpdateUserName, UpdateHostName, UpdateSystemDate, HostIP, DivitInstanceId, InstanceId alanlarını içermektedir (Şekil 5.1.).

SRVDEV\ATLAS.BOA - PRO.Requirement

	Column Name	Data Type	Allow Nulls
🔑	RequirementId	int	☐
	ProjectId	int	☐
	Description	varchar(4000)	☐
	StatusId	int	☐
	TypeId	int	☐
	PlanEffort	int	☑
	ResourceId	int	☑
	ResponsibleWorkgroupId	int	☑
	UserName	varchar(10)	☐
	HostName	varchar(20)	☐
	SystemDate	datetime	☐
	UpdateUserName	varchar(10)	☑
	UpdateHostName	varchar(20)	☑
	UpdateSystemDate	datetime	☑
	HostIP	varchar(15)	☐
	DivitInstanceId	varchar(38)	☑
	InstanceId	int	☑

Şekil 5.1. Gereksinim tablosu

5.1.2. Senaryo tablosu

Senaryo Tablosunda Test Planı ekranında yapılan işlemler tutulmaktadır.

Bu tabloda TestPlanId, WorkPackageId, WorkGroupId, TypeId, EnvironmentId, Description, SytemDate, UsarName, UpdateSystemDate, UpdateUserName alanları tutulmaktadır (Şekil 5.2.).

SRVDEV\ATLAS.BOA - PRO.TestPlan

Column Name	Data Type	Allow Nulls
TestPlanId	int	☐
WorkPackageId	int	☑
WorkgroupId	int	☐
TypeId	tinyint	☐
EnvironmentId	tinyint	☐
Description	varchar(2000)	☑
SystemDate	datetime	☐
UserName	varchar(10)	☐
UpdateSystemDate	datetime	☑
UpdateUserName	varchar(10)	☑

Şekil 5.2. Senaryo tablosu

5.1.3. Hata tablosu

Hata tablosunda hata tanımı ekranında yapılan işlemler tutulmaktadır.

Bu tabloda DefectId, BusinessRequirementId, Title, Hatanın bağlı olduğu gereksinim numarası tabo ile ilişkilendirilir. Bir çok alan parametrik olduğundan numaraları ile bu tabloda tutulmaktadır. Bu alanların parametrik olması esnekliği ve geliştirme ihtiyaç duyulmaksızın yeni özellikler eklenmesini kolaylaştırmıştır. Hata raporlamasını yapan, hataya atanan kişi ve bağlı oldukları ekip direk sistemden alınmaktadır.

Description, AssignerUserCode, AssignerWorkGroupId, AssingneeUserCode, AssingneeWorkGroupId, TypeId, StatusId, PriorityId, ReasonId, CriticalityId, EnvironmentId, PlatformId, OperatingSystamId, HardwareTypeId, ResolutionId, ResourceId, HardwareBrandId, SoftwareId, DivitInstanceId, HostName, UserName, SystemDate, UpdateSystemDate, UpdateUserName, UpdateHostName alanları tutulmaktadır (Şekil 5.3.).

SRVDEV\ATLAS.BOA - PRO.Defect

Column Name	Data Type	Allow Nulls
DefectId	int	☐
BusinessRequirementId	int	☐
Title	varchar(1000)	☐
Description	varchar(8000)	☑
AssignerUserCode	varchar(10)	☐
AssignerWorkgroupId	int	☐
AssigneeUserCode	varchar(10)	☐
AssigneeWorkgroupId	int	☐
TypeId	tinyint	☐
StatusId	tinyint	☐
PriorityId	tinyint	☐
ReasonId	tinyint	☐
CriticalityId	tinyint	☐
EnvironmentId	tinyint	☐
PlatformId	tinyint	☐
OperatingSystemId	tinyint	☐
HardwareTypeId	tinyint	☐
ResolutionId	tinyint	☐
ResourceId	int	☑
HardwareBrandId	tinyint	☐
SoftwareId	tinyint	☐
DivitInstanceId	varchar(38)	☑
HostName	varchar(20)	☐
UserName	varchar(10)	☐
SystemDate	datetime	☐
UpdateSystemDate	datetime	☑
UpdateUserName	varchar(10)	☑
UpdateHostName	varchar(20)	☑

Şekil 5.3. Hata tablosu

Hata tablosu, hata yönetimi sistemlerinin başlıca tablosudur. Bu tabloda tutulan bilgiler hata istatistiklerinde ve yazılım kalitesini artırmada ana kaynağı oluşturmaktadır. Burada tutulan veriler bu yüzden büyük öneme sahiptir.

5.1.4. Hata şablonu tablosu

Hata Şablonu tablosunda Hata Tanımlama ekranında kaydedilen hata şablonları tutulmaktadır.

Bu tabloda DefectTemplateId, Title, Description, AssigneeUserCode, TypeId, StatusId, PriorityId, ReasonId, CriticalyId, EnvironmentId, PlatformId, OperatinSystemId, HardwareTypeId, HardwareBrandId, SoftwareId, ResolutionId, ResurceId, SystemDate, UserName, HostName, UpdateSystemDate, UpdateUserName, UpdateHostName, HostIP alanları tutulmaktadır (Şekil 5.4.).

SRVDEV\ATLAS.BOA...O.DefectTemplate

Column Name	Data Type	Allow Nulls
DefectTemplateId	int	☐
Title	varchar(1000)	☐
Description	varchar(8000)	☑
AssigneeUserCode	varchar(10)	☐
TypeId	tinyint	☐
StatusId	tinyint	☐
PriorityId	tinyint	☐
ReasonId	tinyint	☐
CriticalityId	tinyint	☐
EnvironmentId	tinyint	☐
PlatformId	tinyint	☐
OperatingSystemId	tinyint	☐
HardwareTypeId	tinyint	☐
HardwareBrandId	tinyint	☐
SoftwareId	tinyint	☐
ResolutionId	tinyint	☐
ResourceId	int	☑
SystemDate	datetime	☐
UserName	varchar(10)	☐
HostName	varchar(20)	☐
UpdateSystemDate	datetime	☑
UpdateUserName	varchar(10)	☑
UpdateHostName	varchar(20)	☑
HostIP	varchar(15)	☐

Şekil 5.4. Hata şablonu tablosu

5.1.5. Hata tarihçesi tablosu

Hata Tarihçesi Tablosunda hata ile ilgili yapılan durum versiyonları tutulmaktadır. Bir hata kaydı üzerinde yapılan durum değişiklikler bu tabloda yer almaktadır.

Bu tablada DefectHistoryId, DefectId, BeginDate, EndDate, UserName, FromSatusId, ToStatusId, Comment alanları tutulmaktadır (Şekil 5.5.).

SRVDEV\ATLAS.BOA - PRO.DefectHistory

Column Name	Data Type	Allow Nulls
DefectHistoryId	int	☐
DefectId	int	☐
BeginDate	datetime	☐
EndDate	datetime	☐
UserName	varchar(10)	☐
FromStatusId	tinyint	☐
ToStatusId	tinyint	☐
Comment	varchar(8000)	☑

Şekil 5.5. Hata tarihçesi tablosu

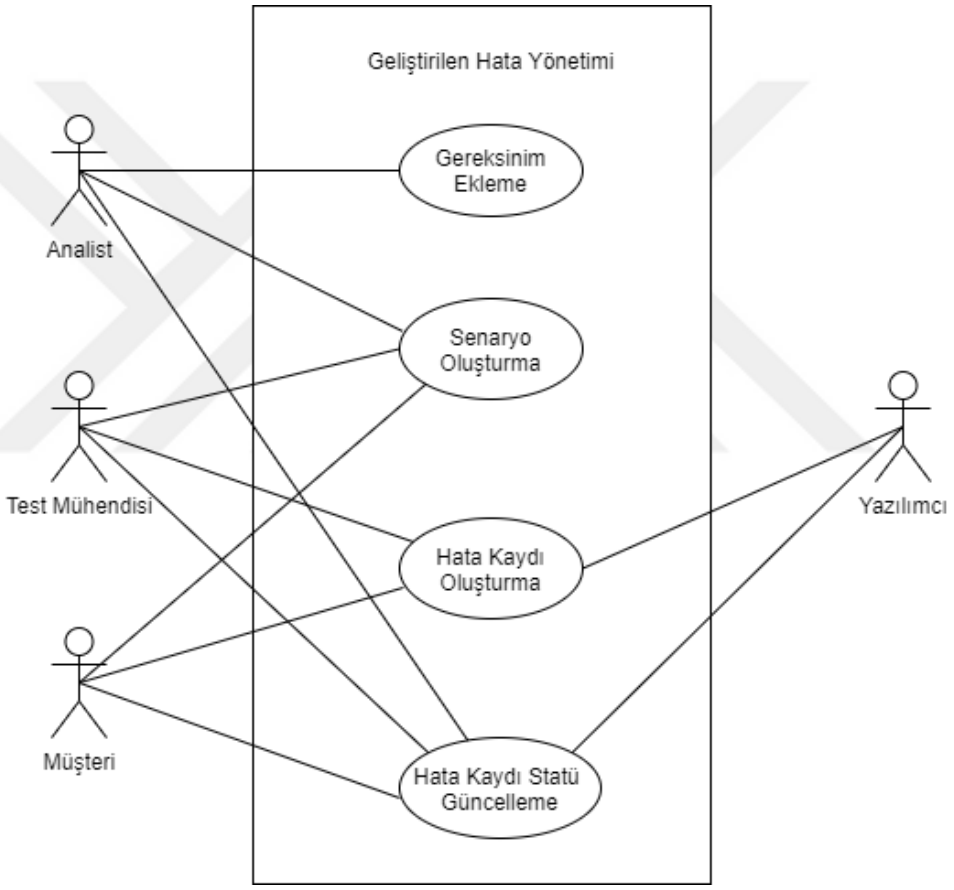
5.2. Diyagramlar

Diyagramlarda genel olarak kullanım durumu (use case) diyagramı ve buradaki aktörlerin iş detaylarını gösteren etkinlik diyagramları gösterilmiştir.

5.2.1. Kullanım durumu diyagramı

Şekil 5.6.'da gösterildiği üzere kullanım durumu diyagramında aktörler ve kullanım durumları mevcuttur. Hata yönetimi sisteminde aktif 4 aktör bulunmaktadır. Bu aktörler; analist, test mühendisi, yazılımcı ve müşteridir. Analist; gereksinim oluşturur, hata kaydı açabilir, test senaryosu oluşturabilir, hata statüsü güncelleyebilir. Test mühendisi; hata kaydı oluşturur, hata kaydı günceller, hata statüsü günceller ve test senaryosu oluşturur. Yazılımcı; hata kaydı statüsünü günceller, hata kaydı

oluşturabilir. Müşteri; hata kaydı ve test senaryosu oluşturabilir, hata kayıtları statülerini güncelleyebilir. Yapılan güncellemelerin tarihçesi tutulmaktadır. İhtiyaca göre aktörler/kullanıcılar sadece kendi hata kayıtlarında güncelleme yapıp diğer hata kayıtlarında güncelleme yapmaları kısıtlanarak sadece görebilmeleri sağlanabilir. Hata kaydı statü güncelleme işlemi yetkileri iki şekilde oluşturulabilir. Birincisi; hatanın atandığı kişi, hatayı atayan kişi ve diğerleri şeklinde gruplanarak yetkiler oluşturulabilir. İkincisi; yazılımcı, analist, test mühendisi, müşteri rollerine göre oluşturulabilir.

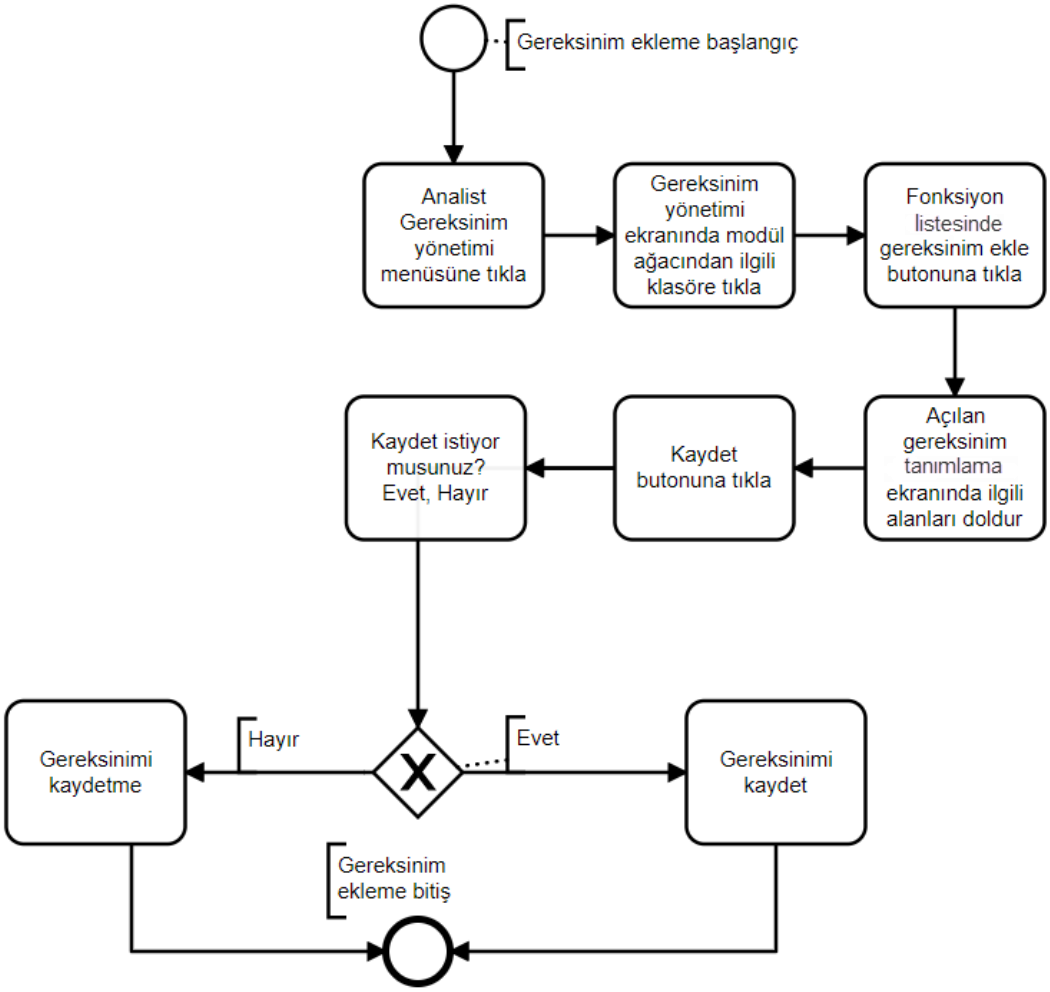


Şekil 5.6. Hata yönetimi kullanım durumu diyagramı

5.2.2. Gereksinim ekleme aktivite diyagramı

Şekil 5.7.'de gereksinimleri analist rolündeki aktörler ekleyebilir. Gereksinimler güncellenebilir. Gereksinimler silinebilir. Gereksinimlerden direkt olarak hata kaydı

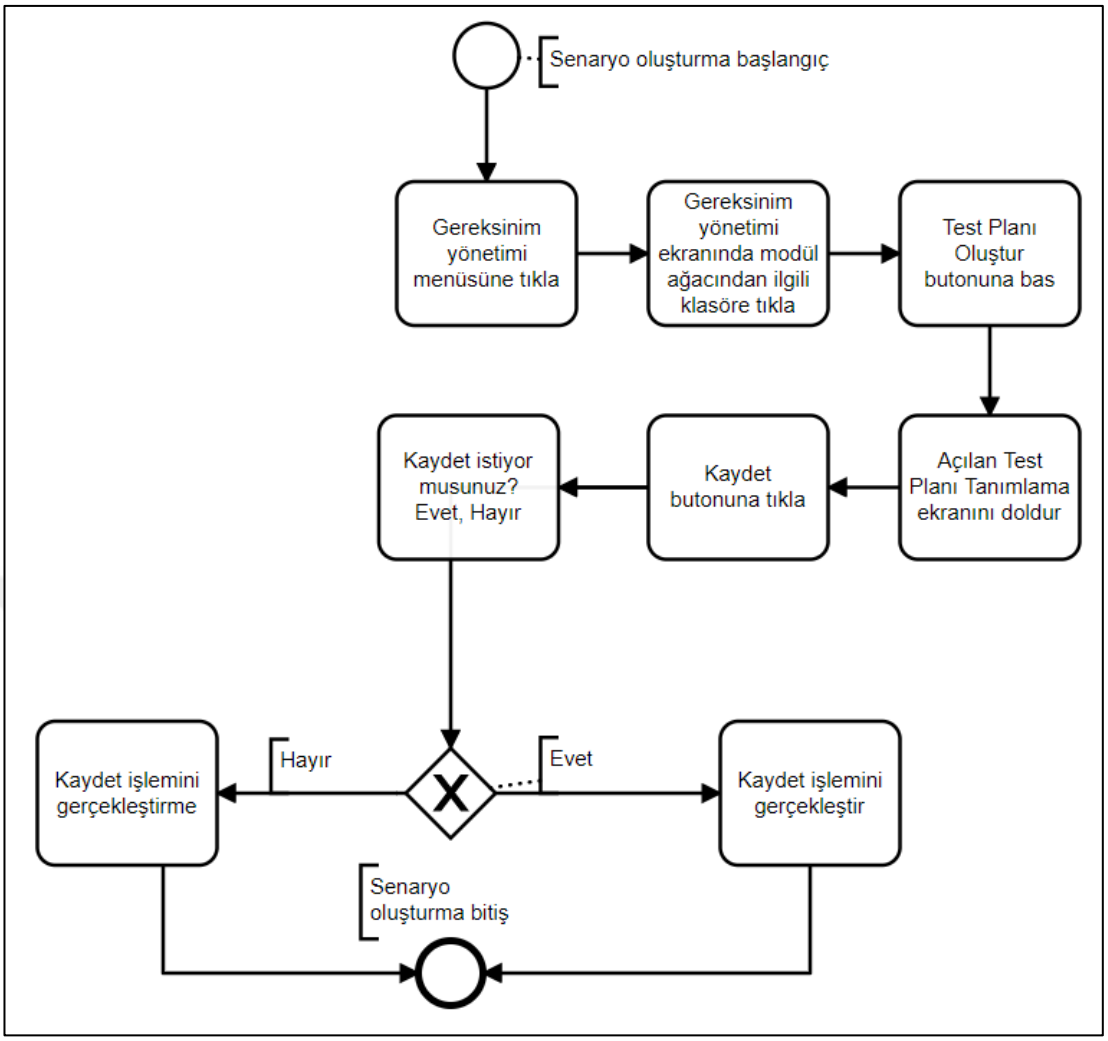
açılabilir. Gereksinimler, gereksinimlerin listelendiği satırlarda bulunan gereksinim güncelleme butonuna basılarak güncelleme tanımlama ekranında güncellenebilir.



Şekil 5.7. Gereksinim ekleme aktivite diyagramı

5.2.3. Senaryo oluşturma aktivite diyagramı

Şekil 5.8.'de senaryo oluşturma işlemi test mühendisi, analist veya müşteri rolleri tarafından yapılabilir. Yapıdaki modelleme gereksinim eşittir senaryo olarak tasarlanmıştır. Bir buton aksiyonu ile gereksinimler senaryo ekranına aktarılır, güncellenebilir özelliktedir.

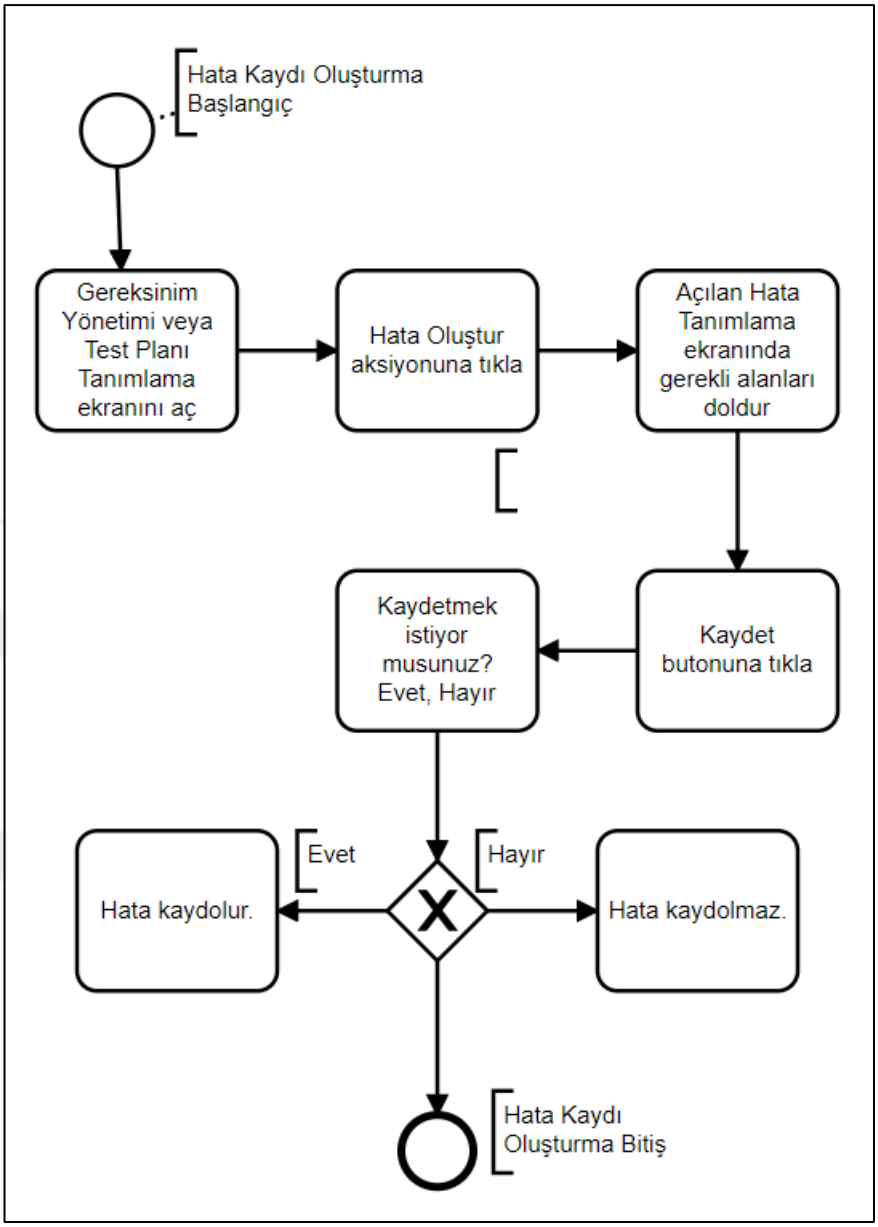


Şekil 5.8. Gereksinim ekleme aktivite diyagramı

5.2.4. Hata kaydı oluşturma aktivite diyagramı

Hata kaydı işlemi başta test mühendisi olmak üzere tüm roller tarafından yapılabilir. Hata kayıtları test ortamları ve üretim ortamı için oluşturulabilir. Hata kaydı işleminde doldurulması zorunlu olmayan ancak hata hakkındaki detayları istatistiksel bilgileri çıkarılabilecek birçok detay eklenebilir. Bu detaylar hata yönetiminde iyileştirme hedeflerine göre ve test yapılan sektöre göre farklılıklar gösterebilir.

Şekil 5.9.'da görüldüğü üzere hata kaydı iki şekilde; gereksinim listesinde bulunan hata oluştur aksiyonu ile veya senaryo listesindeki hata oluştur aksiyonu ile oluşturulabilir.



Şekil 5.9. Hata kaydı oluşturma aktivite diyagramı

5.3. Ekranlar

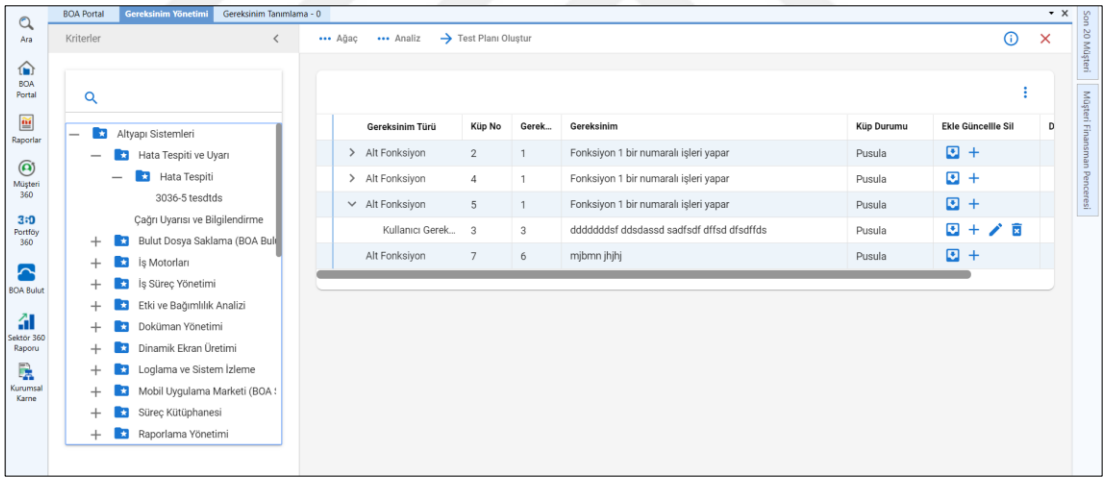
Geliştirilen hata yönetimi gereksinim yönetimi ekranı, gereksinim tanımlama ekranı, test planı tanımlama ekranı, hata tanımalama ekranı ve hata güncelleme ekranlarından oluşmaktadır. Firma içinde kullanılan uygulama içinde geliştrilen bu ekranlar için ayrıca yapılmış bir login ekranı, kullanıcı tanımlama ekranı bulunmamaktadır.

Gereksinim ekranında firma sistemine ait modül ağacı kullanılmaktadır. Kullanıcılar firma sistemindeki personel yapısından çağırılmaktadır.

5.3.1. Modül ağacı

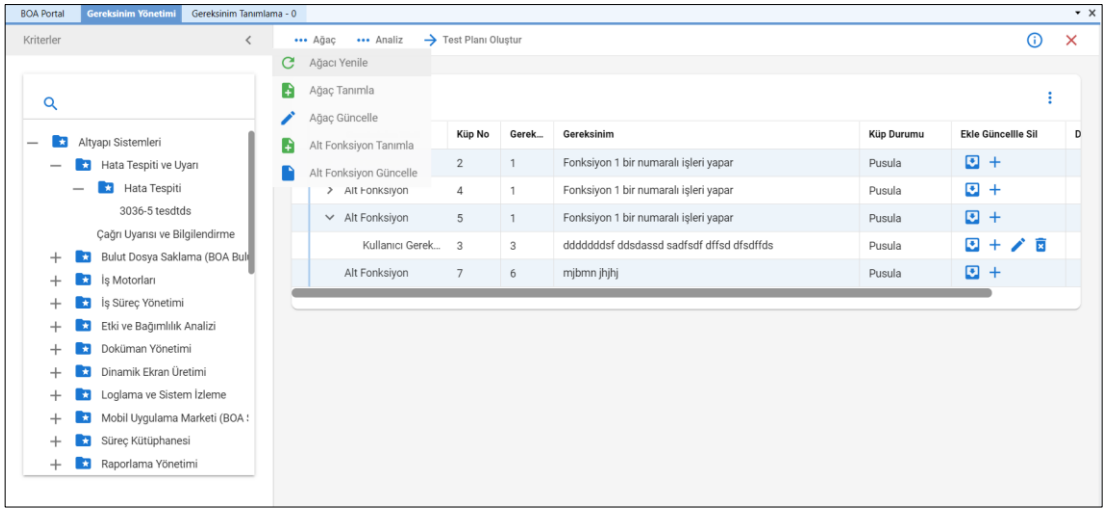
Şekil 5.10.'da Gereksinim Yönetimi ekranının sol yanında olan Modül Ağacı, mevcut sistemin tamında yer alan modüller ve menülerini kapsar. Bunun haricinde ihtiyaç duyulduğunda ana klasör ve alt kırılımlar oluşturulması mümkündür.

Bu Modül Ağacı yapısı sistemde yer alan tüm gereksinimlerin, analizlerin, senaryoların aynı yerde tutulmalarını, toplu şekilde görülmelerini sağlar. Çok paydaşlı projelerde herkes ilgili modülün altına ilgili projeye ait bir klasör, bir alt kırılım oluşturabilir. Bu yapı projelerin, analizlerin veya gereksinimleri daha kolay ulaşılmasını sağlar.



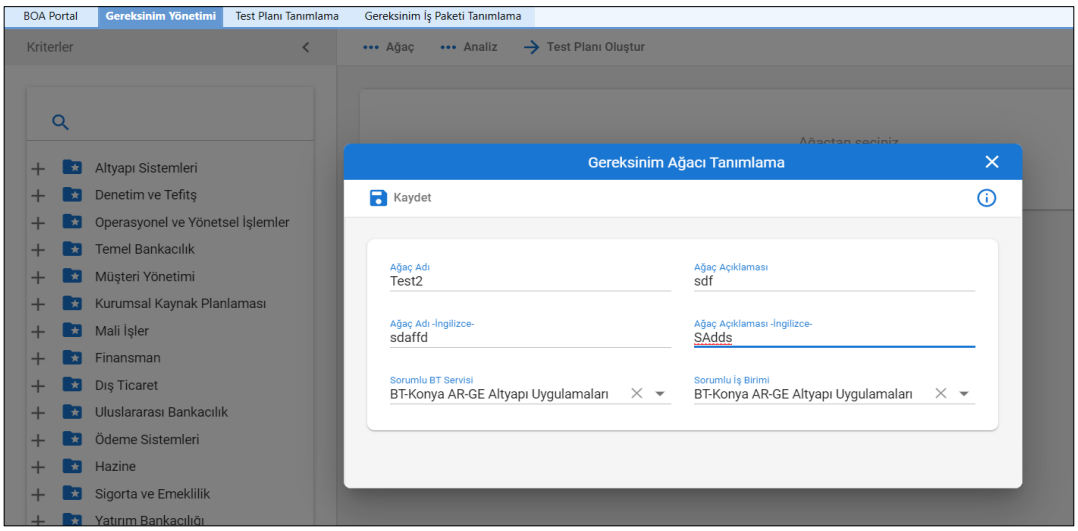
Şekil 5.10. Modül ağacı

Yetkisi olan kişiler örneğin analistler burada yeni bir ana klasör veya yeni bir alt klasör oluşturabilirler. Yukarıda bulunan Ağaç menüsünden güncellemeler, yeni tanımlamalar yapılabilmektedir. Ağacı Yenile butonu ile yaptığınız değişiklik ağaca yansır.



Şekil 5.11. Modül ağacı menüsü

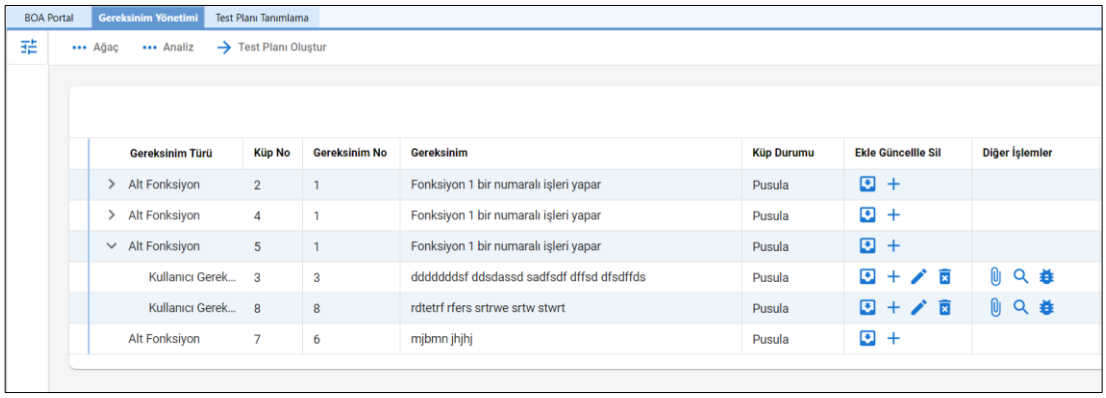
5.11. şeklinde görülen yeni ağaç tanımı ana uygulama içinde olmayan modüller için ihtiyaç olacaktır. Mevcut ağaçta bir klasörün kullanılması zorunlu tutulursa bu kullanıcılar için kısıtlayıcı olur. Alt Fonksiyon Tanımla aksiyonu ağaçta herhangi bir kırılım oluşturulmaz. Bu aksiyon oluşturulacak gereksinimleri gruplamaya yarayan, fonksiyon bazlı bir yapı oluşturulmasını sağlamaktadır. Gereksinim Ağacı Tanımlama ekranı ayrı bir ekranda açılmayıp ekran üzerinde pencere olarak açılır ve ağaç tanımlama işlemi yapılır. Sistemde genel olarak bu ekran gibi çok detay içermeyen, içeriği fazla olmayan ekranlar ayrı bir ekranda açılmayıp aynı ekran üzerinde açılarak işlemin yapılması sağlanır. Böylelikle hız ve pratiklik sağlanmış olur.



Şekil 5.12. Gereksinim ağacı tanımlama ekranı

5.3.2. Gereksinim yönetimi ekranı

Şekil 5.12.'de görülen Gereksinim Yönetimi ekranında soldaki kriterler alanında bankacılık sistemi içindeki tüm modüller ağaç şeklinde verilmiştir. Bu menü sol tarafa açma/kapatmaözelliği verilerek ağaç seçimi sonrası ekranda daha rahat çalışılması sağlanmıştır. Ekrana analistler için okuma, güncelleme yetkisi verilmiştir. Yazılımcılar ve test mühendisleri için izleme yetkisi verilmiştir. Bu ekrandaki Gereksinim Tanımla butonu ile açılan ekrandan ilgili ekrana, modüle veya onlara bağlı iş paketlerine gereksinim girilebilir. Gereksinim Güncelle butonu ile seçilen gereksinim güncellenir ve kaydedilir. Gereksinim Yönetimi ekranında Alt Fonksiyonların listelendiği satırlardan Gereksinim ekleme ve güncelleme işlemi yapılabilir (Şekil 5.13.). Bu ekrandaki en önemli özellik gereksinimlerin atomik olarak gözükmesidir. Test edilebilir gereksinim anlayışıyla yazılmış bu gereksinimler için aynı satırdaki hata tanımla butonu ile anında hata tanımlanabilir. Bu eski yapıdaki hata tanımlama öncesi yapılan birçok adımdan oluşan işlem adımlarını sıfıra indirmiştir. Aynı satırda bulunan gereksinim güncelle, sil, tanımla butonları ile de gereksinim tanımlama, ekleme, güncelleme, silme işlemleri pratik şekilde yapılır.

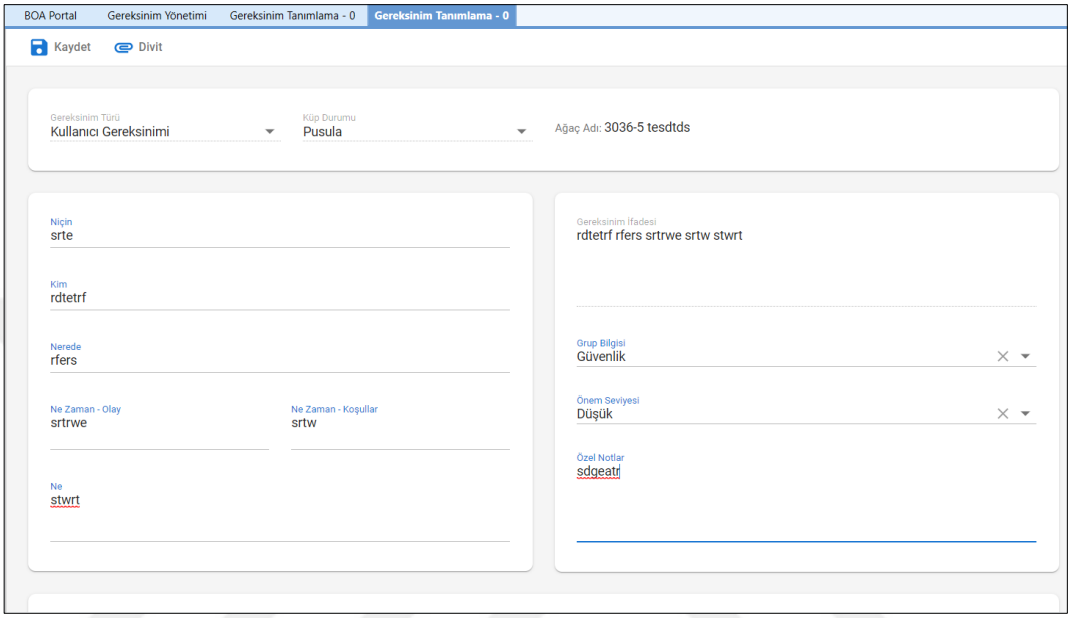


Şekil 5.13. Gereksinim yönetimi ekranı

5.3.3. Gereksinim tanımlama ekranı

Şekil 5.14.'te Gereksinim Yönetimi ekranında Gereksinim Tanımla butonuna basıldığında Gereksinim Tanımla ekranı açılır. Bu ekranda test edilebilir gereksinim yazma amacıya 5N+1K yapısı ile gereksinim tanımı yapılır. Ne? Niçin? Nerede? Ne

zaman-Olay? Ne zaman-koşullar nedir? Sorularının cevapları bir gereksinimi oluşturur. Grup Bilgisi alanında tanımlanan gereksinimin Ekran, Entegrasyon, Güvenlik, İş Akışı, Loglama, Performans, Task gruplardan hangisine girdiği işaretlenir (Şekil 5.14.).

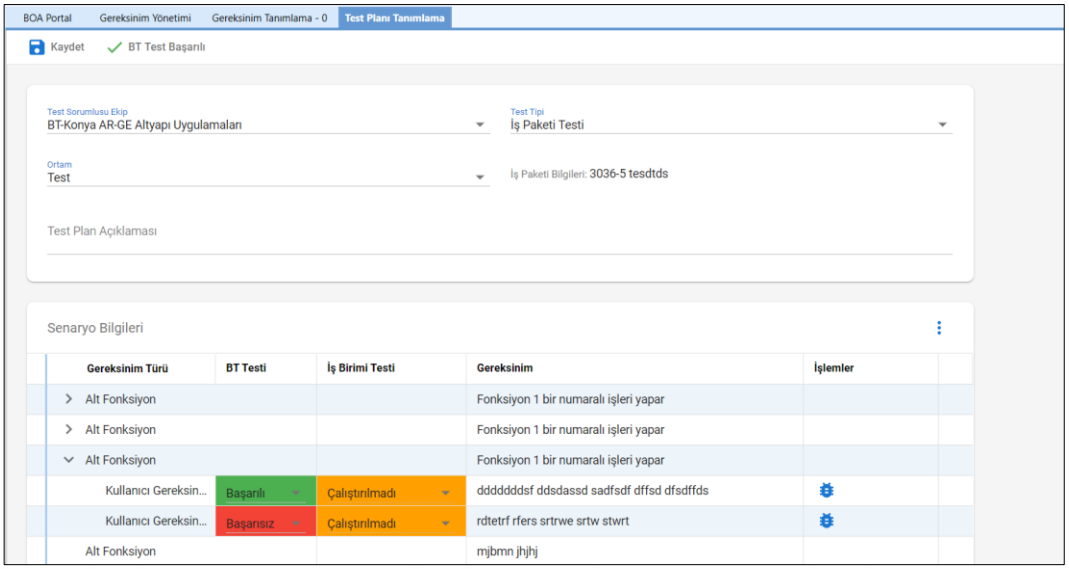


Şekil 5.14. Gereksinim tanımlama ekranı

5.3.4. Test planı tanımlama ekranı

Şekil 5.13.'te Gereksinim Yönetimi ekranındaki ilgili Gereksinimler Test Planı Oluştur butonu ile 5.15.'de Test Planı Tanımla ekranı açılır. Bu ekranda gereksinimler test senaryoları şeklinde listelenir (Şekil 5.15.). Gereksinim Türü sütununda gereksinimler listelenir.

BT Testi alanında Bilgi Teknolojileri tarafından atanan Test Mühendisinin test sonuçları seçilir. İş Birimi Testi sütununda ise ilgili Geliştirme için atanmış son kullanıcı test sonuçları belirlenir. Buradaki test sonuçlarında üç seçenek bulunmaktadır; "Başarılı", "Başarısız", "Çalıştırılmadı" (Şekil 5.16.).

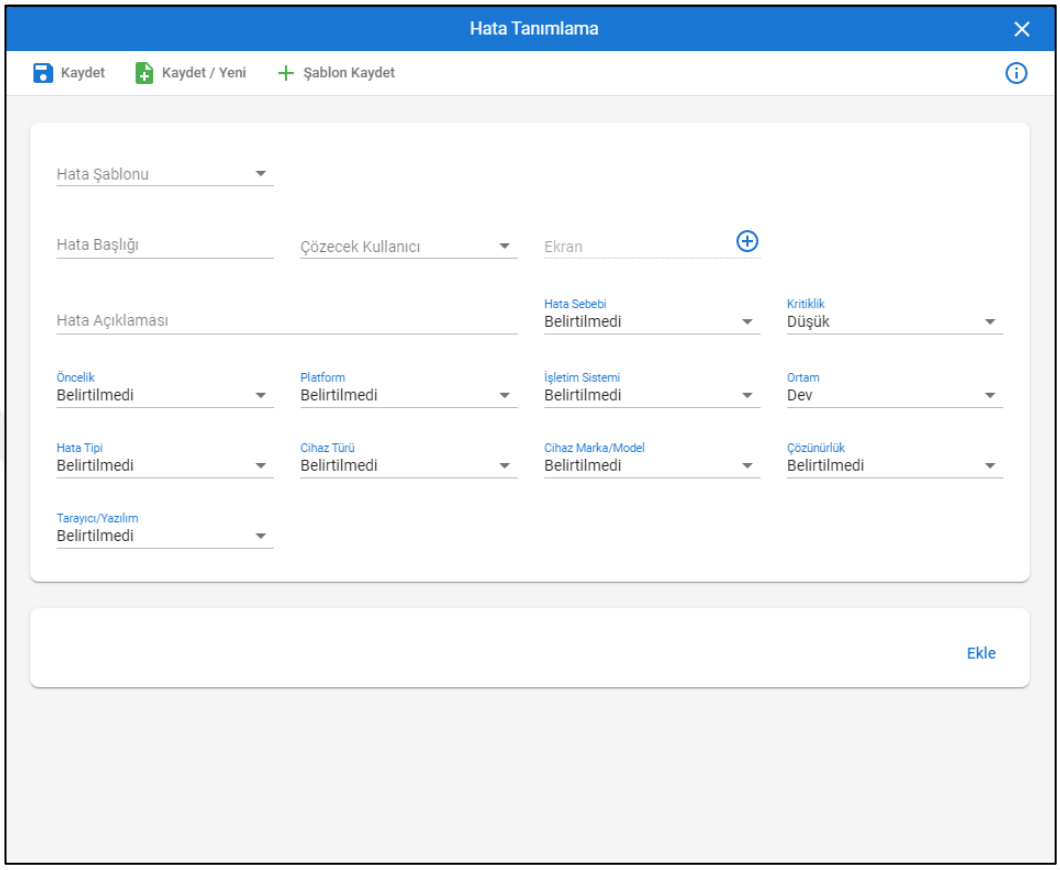


Şekil 5.15. Test planı tanımlama ekranı

5.3.5. Hata tanımlama ekranı

Şekil 5.15.'de Gereksinim Yönetimi ve Test Planı Tanımlama Ekranındaki listelenen senaryolar veya gereksinim satırında bulunan Hata Oluştur butonundan Hata Tanımlama ekranı açılır (Şekil 5.16.). Kaydet butonu, hatanın direk kaydedilmesini sağlar. Kaydet/Yeni butonu girilen hatayı kaydeder ve farklı bir hata girişi için yeni bir Hata Tanımlama ekranı açar. Şablon Kaydet butonu girilen hatanın bilgilerini içeren bir şablon oluşturur. Kaydedilen şablonlar ekrandaki Hata Şablonu açılır listesinden seçilir (Şekil 5.16.). Bu özellik aynı proje, aynı ekran veya aynı gereksinim için girilen hatalarda ilgili alanlar tekrar tekrar girişi konusunda pratiklik sağlar. Ekrandaki alan şablon sayesinde otomatik doldurulmuş olur. Hata Başlığı alanına hatanın kısa özeti veya başlığı girilir. Hata Açıklaması alanında ise hata alınan senaryo adım adım yazılır. Platform alanında kurumun alternatif kanalları dahil mobil şube, internet şube gibi tüm ürünleri yer alır. Ortam alanında ise kuruma ait test ortamları ve gerçek ortam seçilir. Hata Tipi alanında hatanın ortam, kullanıcı deneyimi, tasarım, metin hatası tiplerinden biri seçilir. Cihaz ile ilgili alanlar özellikle mobil testlerinden sıkça ihtiyaç duyulan alanlarlardır. Bu alanlar hatanın çözümünde olması gereken temel bilgilerdir. Bu özelliklerin senaryo içeriğinde yazılmayıp ekranda girişinin yapılması, sınıflandırılması istatistiksel bilgileri, hata yönetiminde çıkarılacak dersleri

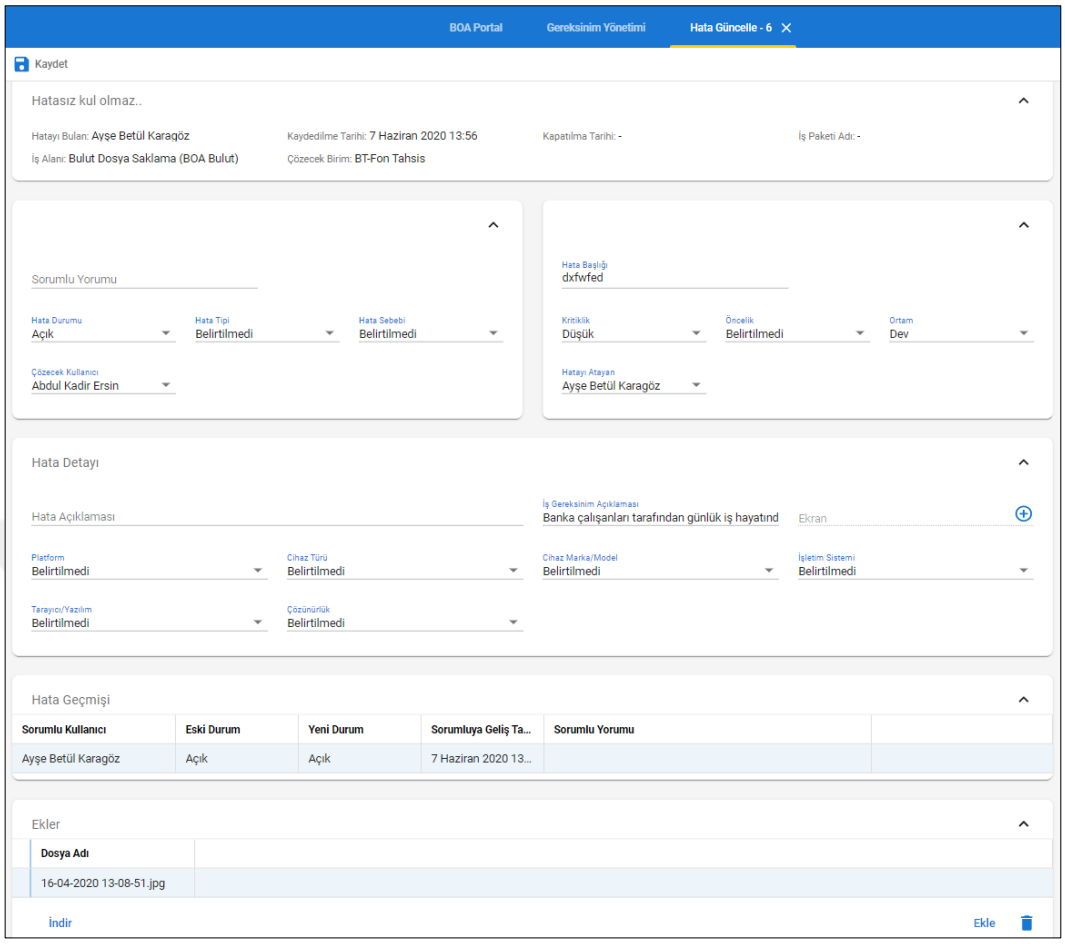
nokta atışı hale getirecektir. Ekle butonunda hataya dair alınan görüntü, kamera kaydı veya gerekli dokümanlar eklenir (Şekil 5.16.).



Şekil 5.16. Hata tanımlama ekranı

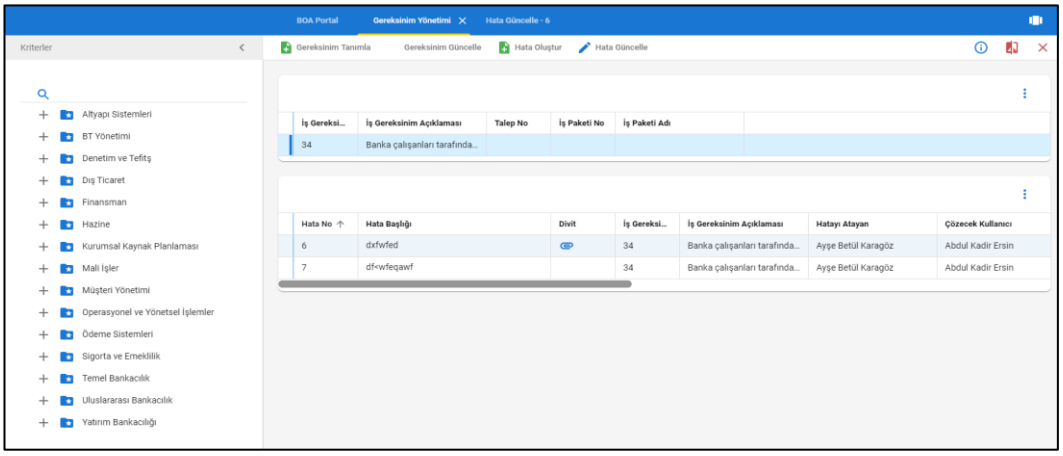
5.3.6. Hata güncelleme ekranı

Hata Yönetiminde Hata Güncelle işlemi en çok yapılan işlemdir. Hatanın açık, çözüldü, tekrar açıldı, reddedildi ve benzeri tüm durumları hata güncelle işlemi ile yapılır. Şekil 5.18.'de görülen ekranda Hata Güncelle ekranında ilgili hata seçilerek Hata Güncelle butonuna basılır ve Hata Güncelle ekranında hata detayı açılır (Şekil 5.17.). Güncelle işlemleri hata raporunu oluşturan kişi, yazılımcı, analist ve test mühendisine göre çeşitli yetkiler dahilinde aktif pasif alanlar ile açılır. Hata durumu için her yetkiye göre durumlar listelenir ve seçim yapılabilir. Hata Güncelle ekranında ilgili hatanın bağlı olduğu gereksinim veya senaryo gösterilir. Bu hata güncelle esnasında gereksinimi hatırlama açısından kolaylık sağlar.



Şekil 5.17. Hata güncelleme ekranı

Raporlanan hatalar Gereksinim Yönetimi ekranında ilgili gereksinim altından listelenir (Şekil 5.18.).



Şekil 5.18. Raporlanan hataların listelenmesi

BÖLÜM 6. KARŞILAŞTIRMA

Bu bölümde mevcut hata yönetimi ve geliştirilen hata yönetimi kullanıcıları geri dönüşleri ve işlem adımları üzerinden karşılaştırılmıştır.

6.1. Mevcut Hata Yönetimi ile İlgili Yapılan Geri Bildirimler

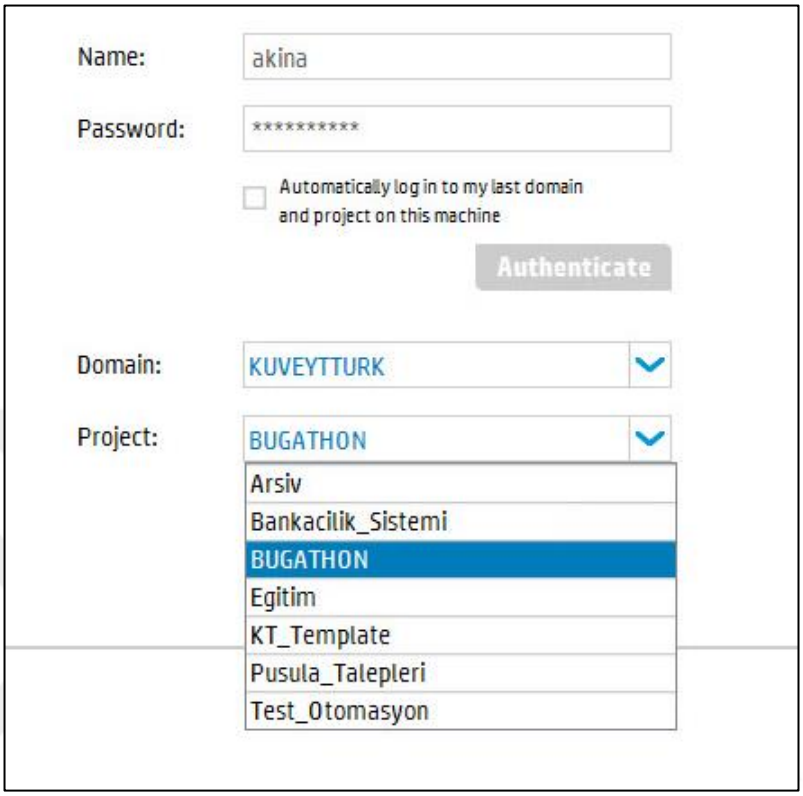
Mevcut Hata Yönetiminde kullanılan aracın kullanımında yaşanan sorunlar, aracın kullanılmasına yönelik çalışanların gösterdiği direnç ve çalışanlardan gelen olumsuz geri bildirimler üzerine yeni bir hata yönetimi yazılması kararlaştırılmıştır. Geliştirilen bu yeni hata yönetiminde göz önünde bulundurulan geri bildirim maddelerini genel olarak sıralayacak olursak şöyledir;

- Uygulama çok karışık yazdığınız yönergelere bakmadan işlemlerimi yapmakta güçlük çekiyorum.
- Uygulamadaki süreçler çok uzun Excel ve Wordde işlemlerimi daha kısa sürede yapabilirim.
- Klasörlerimi silemiyorum.
- Klasör adlarında tire, parantez gibi ifadelerin desteklenmediğine dair mesaj vermiyor, anlaşılmaz bir hata veriyor.
- Uygulamaya güvenemiyorum.

6.2. Mevcut Hata Yönetiminde Hata Raporlama Adımları

Mevcut Hata Yönetimi firmanın kendi uygulaması değildir. Firmanın kendi uygulamasına da entegre değildir. Firmanın ağına ve veritabanına entegredir. Uygulamaya giriş için bir giriş ekranı bulunmaktadır ve uzun süre işlem

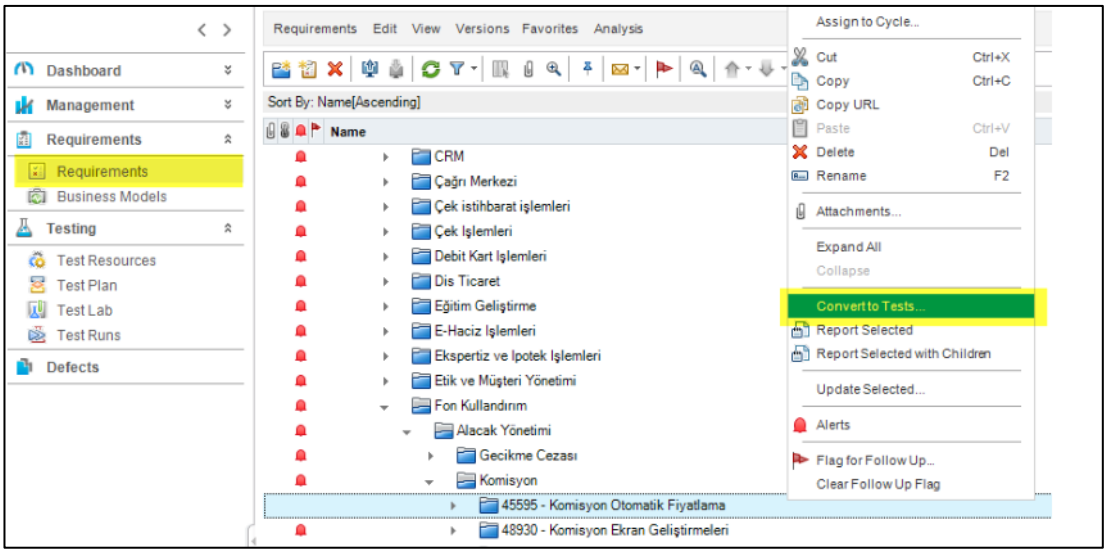
yapılmadığında kullanıcı girişi pasif hale getirildiğinden kullanıcı bu durumda tekrar giriş yapmalıdır. Belli sürelerle kullanıcı sertifikaları alınarak kullanılmaktadır.



Şekil 6.1. Giriş ekranı

Şekil 6.1.'de görüldüğü üzere mevcut hata yönetiminde bir kullanıcı giriş ekranı bulunmaktadır.

Bu uygulamadaki kullanım yapısı proje bazlı geliştirilmiştir. Her girişte proje seçimi yapılarak giriş yapılması gerekmektedir. Bu yapı tüm projeleri aynı anda görebilmeyi engellemektedir. Geliştirilen yapıda ise modül ağacı ile tüm projeleri tek ekran üzerinden görülebilmesi sağlanmıştır.



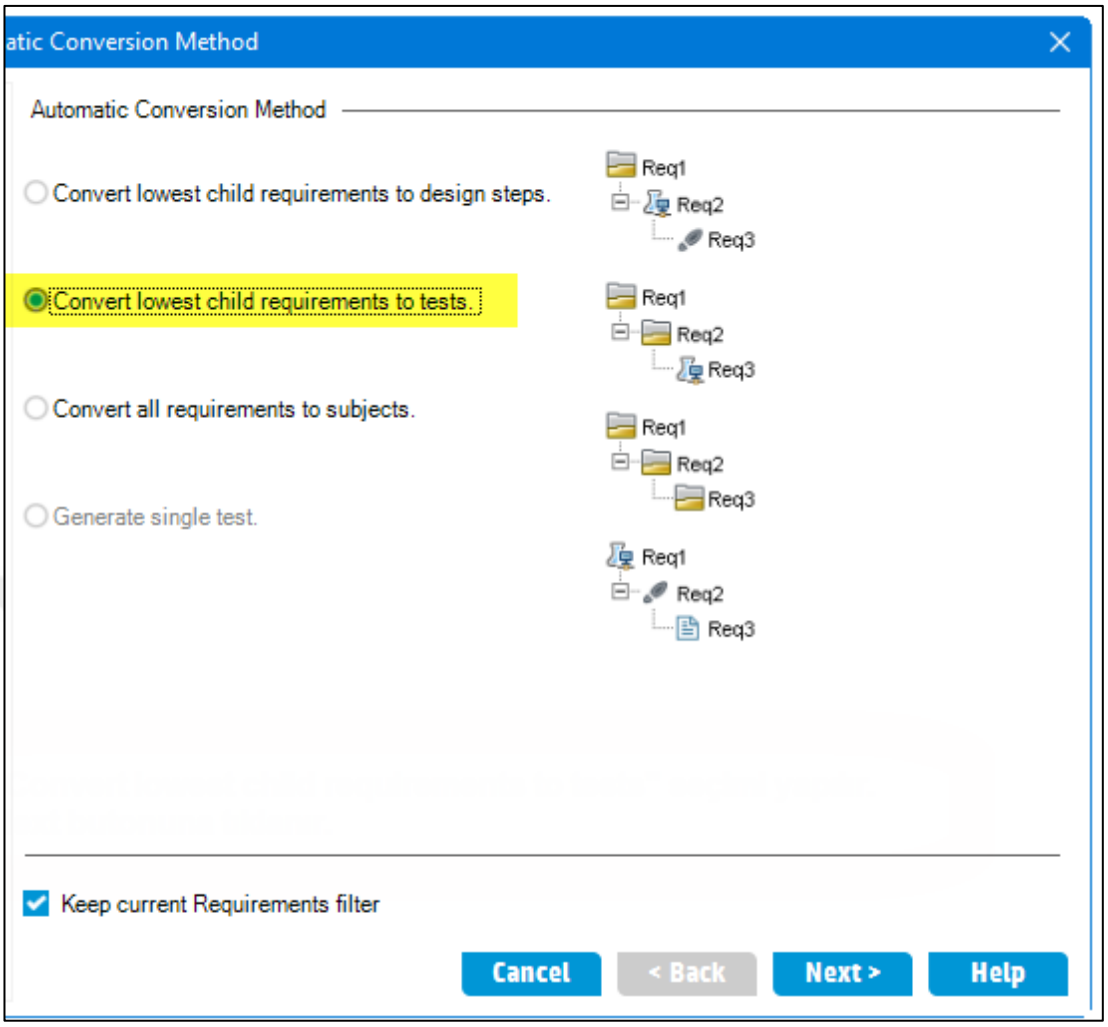
Şekil 6.2. Requirement (gereksinim) ekranı

Mevcut uygulamada Şekil 6.2.'de görülen sol tarafta Requirement (gereksinim) menüsüne tıklandığında gereksinimlerin listelendiği ekrandır. Bu ekrandan yeni gereksinimler proje bazlı olarak oluşturulabilmektedir.

Şekil 6.3.'teki ekran ise gereksinimlerin senaryo ekranına aktarıldığı ekrandır. Burada hangi katman seviyesinde senaryoya çevirileceği seçilmektedir.

Mevcut Uygulamada bir hata raporlama işlemi için giriş yaptıktan sonra izlenilen adımlar şu şekildedir;

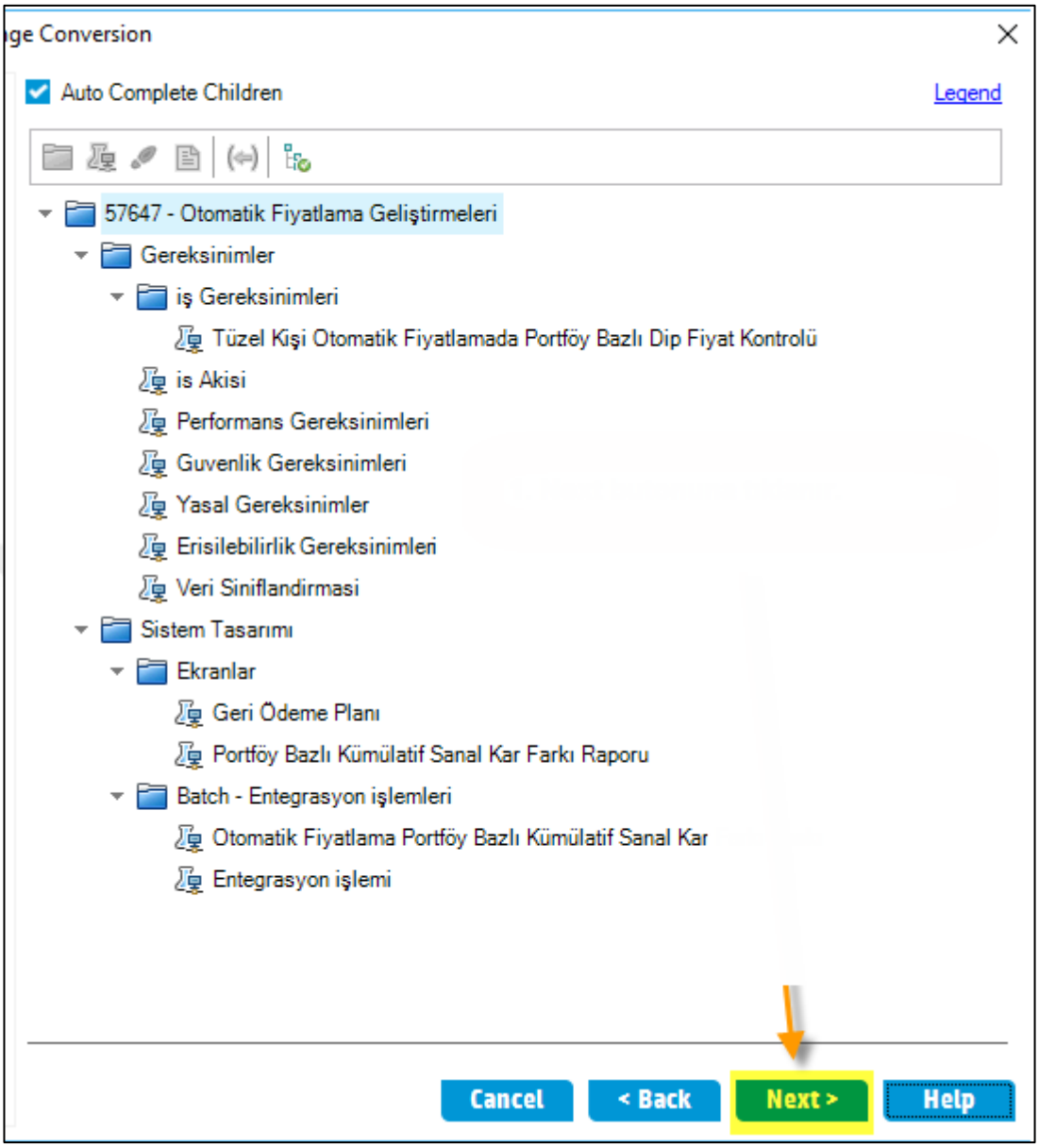
1. Sisteme giriş yapılır (Şekil 6.1.).
2. Gereksinimler (Requirements) modülü açılır (Şekil 6.2.).
3. Senaryosu yazılacak projenin analiz klasörüne sağ tıklanır (Şekil 6.2.).
4. Açılan menüde "Teste Çevir" ("Convert to Test") seçeneği tıklanır (Şekil 6.3.).



Şekil 6.3. Gereksinimlerin senaryo ekranına aktarılması

5. "En alt kırılım geresinimlerini teste çevir" ("Convert lowest child requirements to design steps to tests") seçilir (Şekil 6.3.).
6. Sonraki (Next) butonuna tıklanır (Şekil 6.3.).

Bu kısımda Şekil 6.3.'te görüldüğü üzere oluşturulan senaryoların analiz klasöründeki hangi kırılıma kadar alıncağı seçilir. Bu seçimler ile analiz, senaryo tarafında klasör adına, senaryo adına veya senaryo step adlarına dönüştürülebilir.

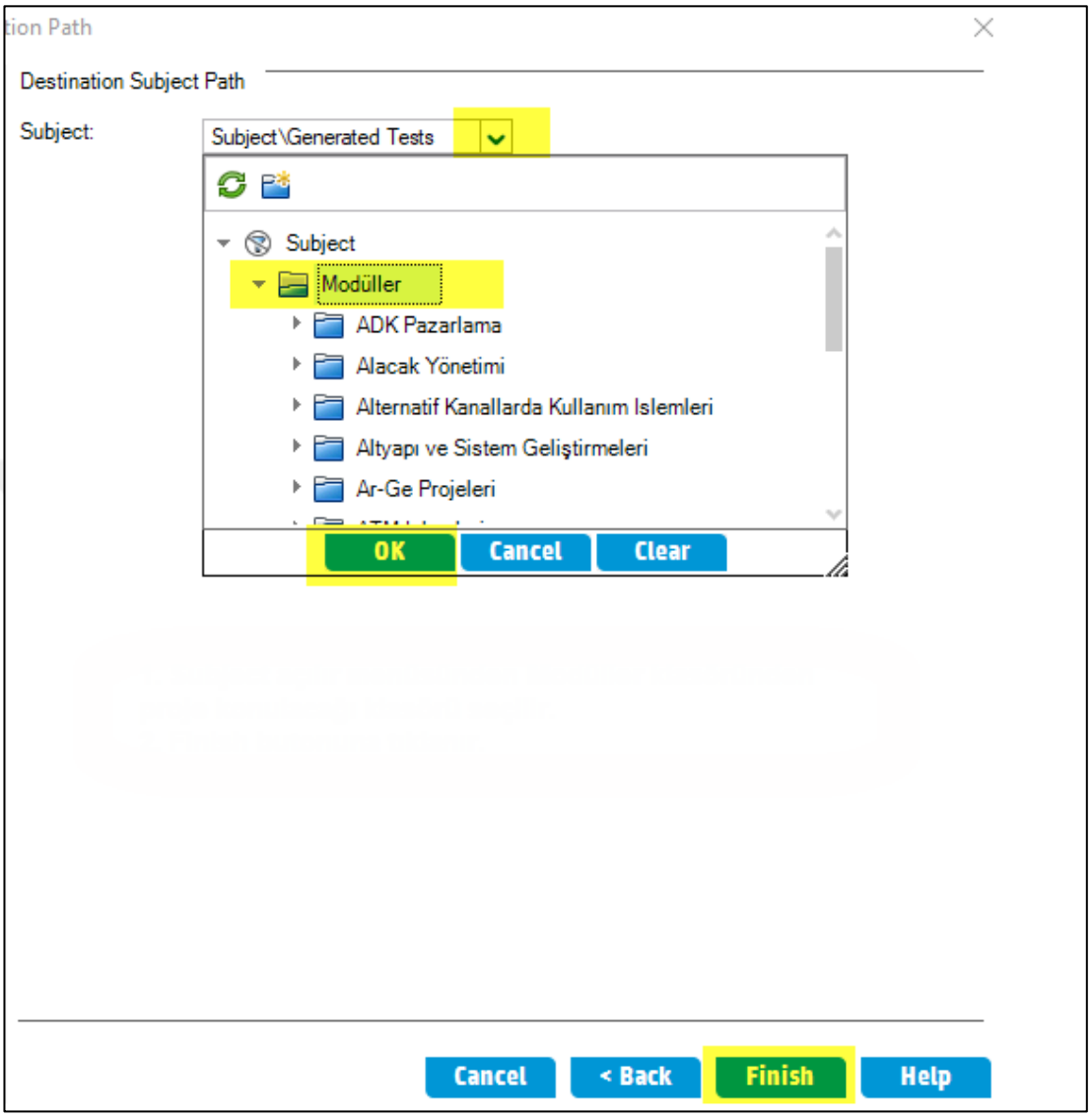


Şekil 6.4. Gereksinimlerin senaryo ekranına aktarılması adım-2

7. Şekil 6.4.'teki ekran açılır ve bu ekranda Sonraki (Next) butonuna tıklanır.

Şekil 6.4.'te görüldüğü üzere analiz klasörü gereksinim kırılımları senaryo kırılımları olarak dönüştütülmektedir.

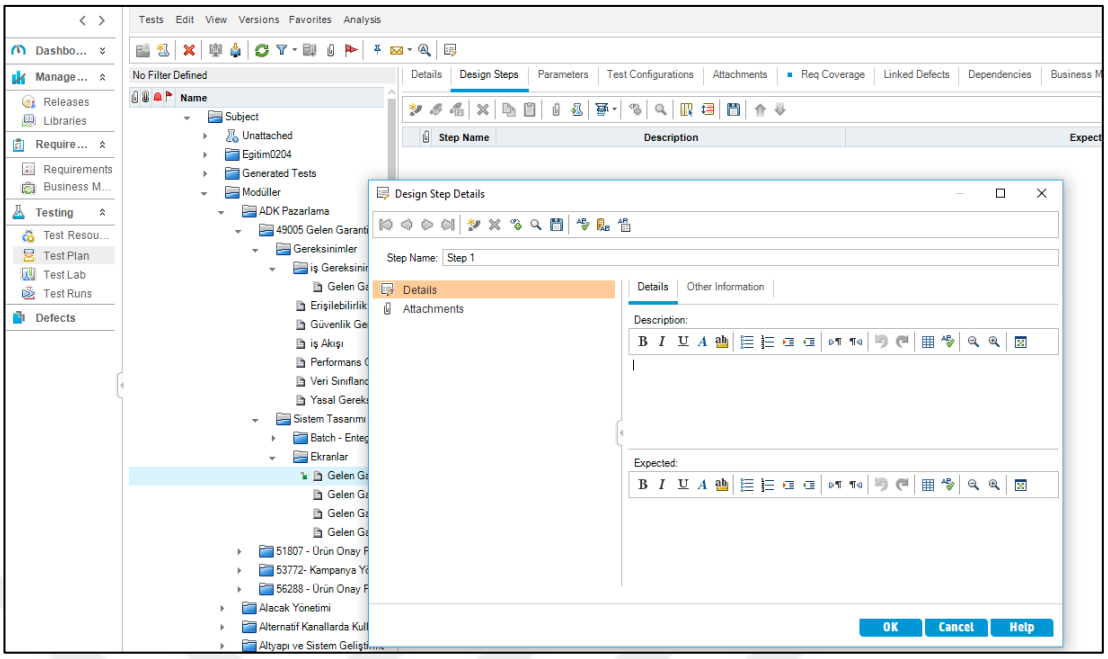
Buradaki analiz klasörünü kullanıcılar bir şablon şeklinde veya istedikleri gibi planlayabilirler. Standart bir şablon kullanılması zorunlulukların eksiksiz yerine getirilmesini ve kullanıcı deneyimi kolaylığı sağlar.



Şekil 6.5. Gereksinimlerin senaryo ekranına aktarılması adım-3

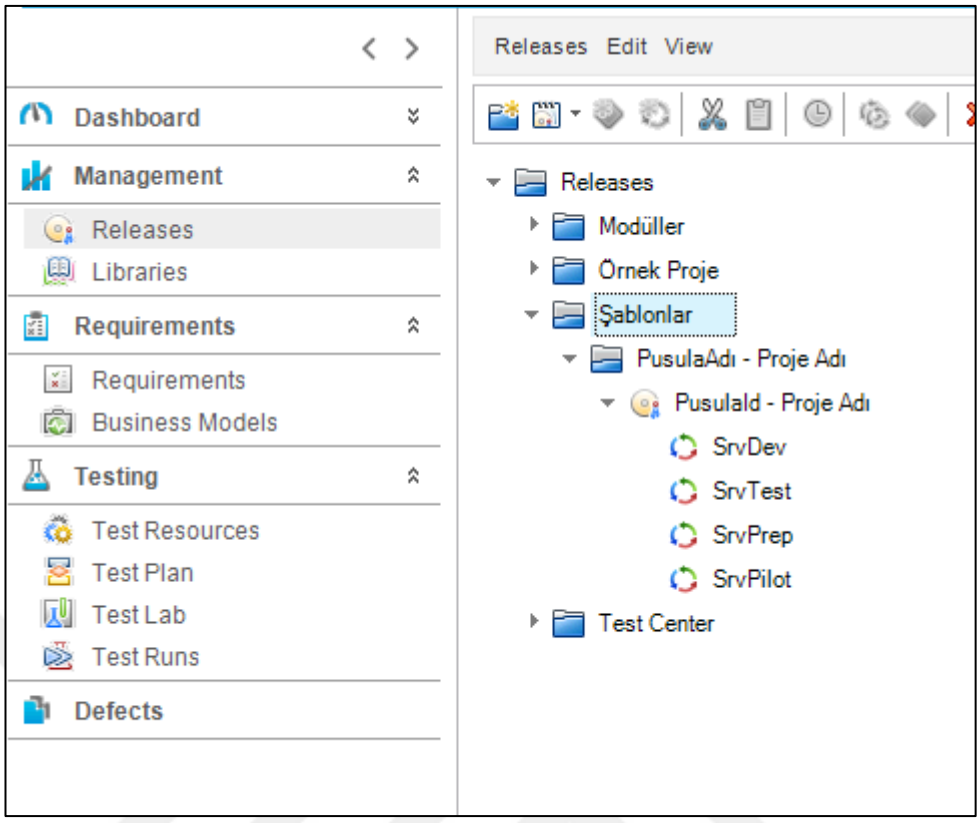
8. Konu (Subject) açılır menüsündeki Modüller klasöründen projenin konulacağı alt klasör seçilir (Şekil 6.5.).
9. Tamam ve Bitir (OK ve Finish) butonlarına basılır (Şekil 6.5.).

Bu adıma kadar Gereksinimler (Requirements) klasöründeki analizin başlıkları ile Test Plan menüsünde senaryo klasörün başlıkları oluşturulmuş olur. İlgili klasöre ek senaryo başlıkları da eklenebilir.



Şekil 6.6. Senaryo yazma ekranı

10. Test Plan modülüne tıklanır (Şekil 6.6.).
11. Proje klasörü seçilir (Şekil 6.6.).
12. Senaryosu yazılacak gereksinim veya ekran adı çift tıklanır (Şekil 6.6.).
13. Senaryo Adımı Dizaynı (Design Step) sekmesine tıklanır (Şekil 6.6.).
14. Yeni Adım (New Step) butonuna tıklanır (Şekil 6.6.).
15. Açılan "Senaryo Adımı Dizaynı Detayları" ("Design Step Details") ekranında senaryo adımları yazılır (Şekil 6.6.).
16. Açıklama (Description) alanı doldurulur (Şekil 6.6.).
17. Beklenen (Expected) alanı doldurulur (Şekil 6.6.).
18. Tamam (OK) butonuna tıklanır (Şekil 6.6.).
19. Tüm senaryolar bu şekilde adım adım yazılarak tamamlanır.

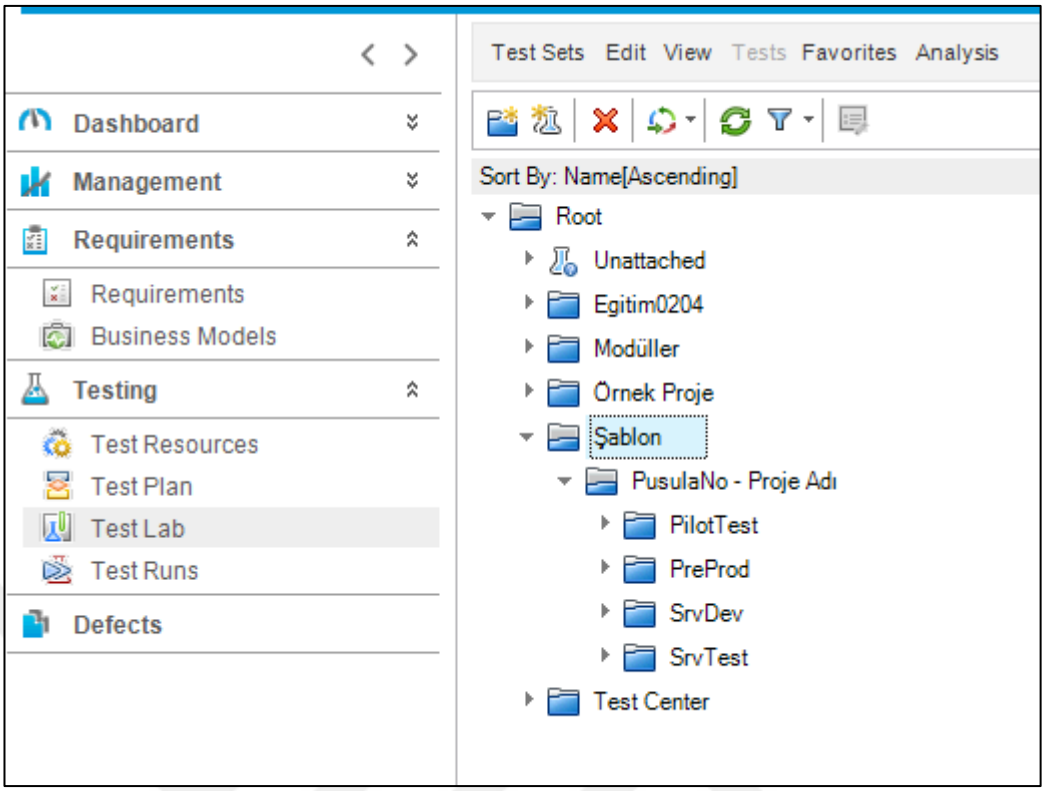


Şekil 6.7. Release oluşturma ekranı

20. Sürümler (Releases) menüsüne tıklanır (Şekil 6.7.).
21. Sürümler (Releases) klasörü altından Şablon klasörü içindeki PusulaNo-Proje Adı isimli klasör kopyalanır (Şekil 6.7.).
22. Sürümler (Relases) klasörü altındaki Modüller klasörü altından projemizin yer alacağı klasörün içine yapıştırılır (Şekil 6.7.).
23. Pusula No- Proje Adı klasör adı ilgili pusula no ve proje adı ile güncellenir (Şekil 6.7.).

Bir projenin testi en az iki ortamda gerçekleşir. Bu ortamlar test ve tüm geliştimelerin birleştirildiği entegrasyon ortamıdır. Projenin yapısına göre geliştirme ortamı ve gerçek ortam da testler için kullanılabilir.

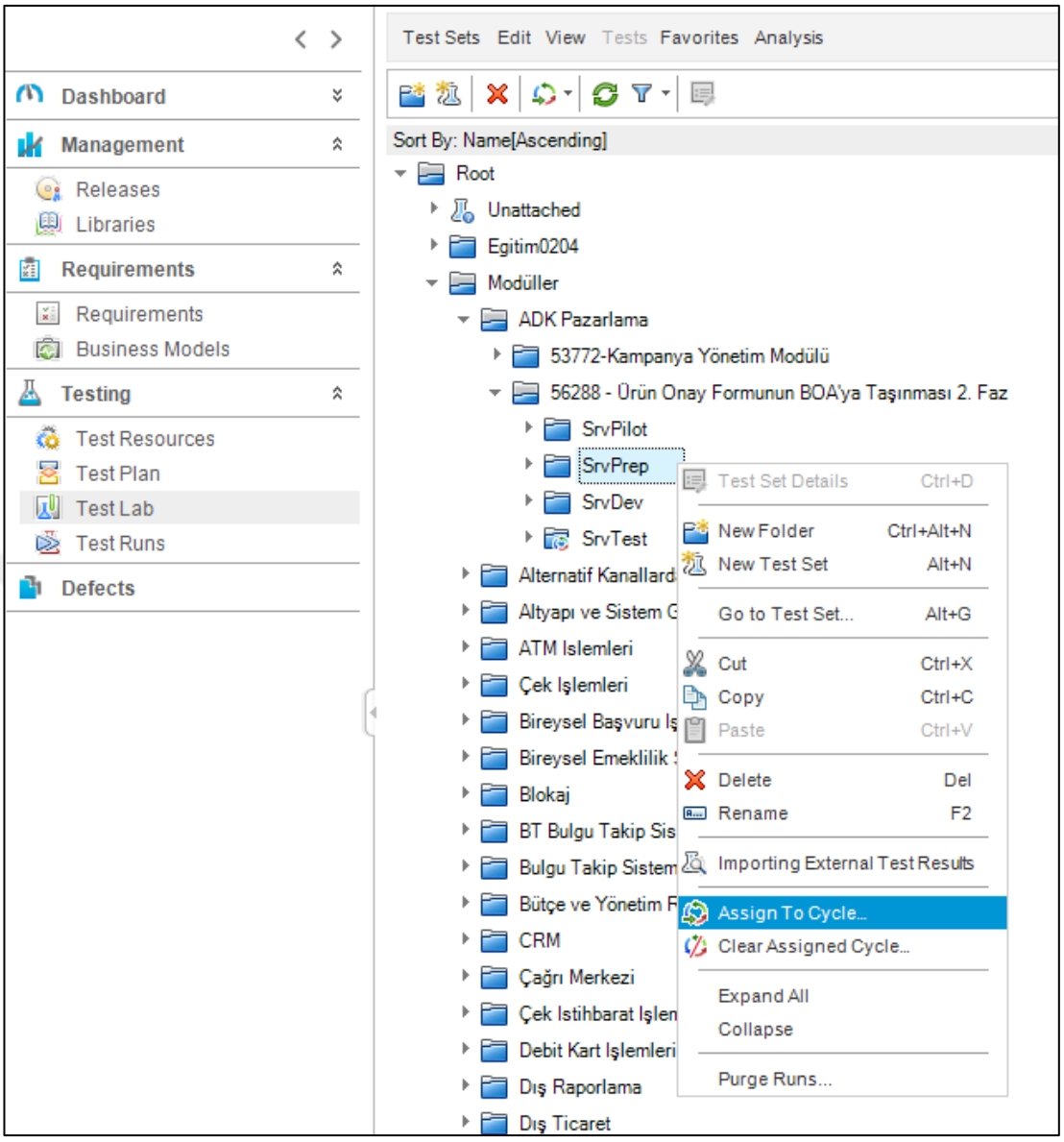
Release ve Cycle yapısı, proje hataları raporlanırken hatanın hangi ortamda alındığını ayrıştırmayı sağlar.



Şekil 6.8. Cycle oluşturma ekranı

24. Test Lab menüsüne tıklanır (Şekil 6.8.).
25. Kök (Root) klasörü altından Şablon klasörü içindeki Pusula No- Proje Adı isimli klasör kopyalanır (Şekil 6.8.).
26. Kök (Root) klasörü altındaki Modüller klasörü altından projemizin yer alacağı klasörün içine yapıştırılır (Şekil 6.8.).
27. Pusula No- Proje Adı klasör adı ilgili pusula no ve proje adı ile güncellenir (Şekil 6.8.).

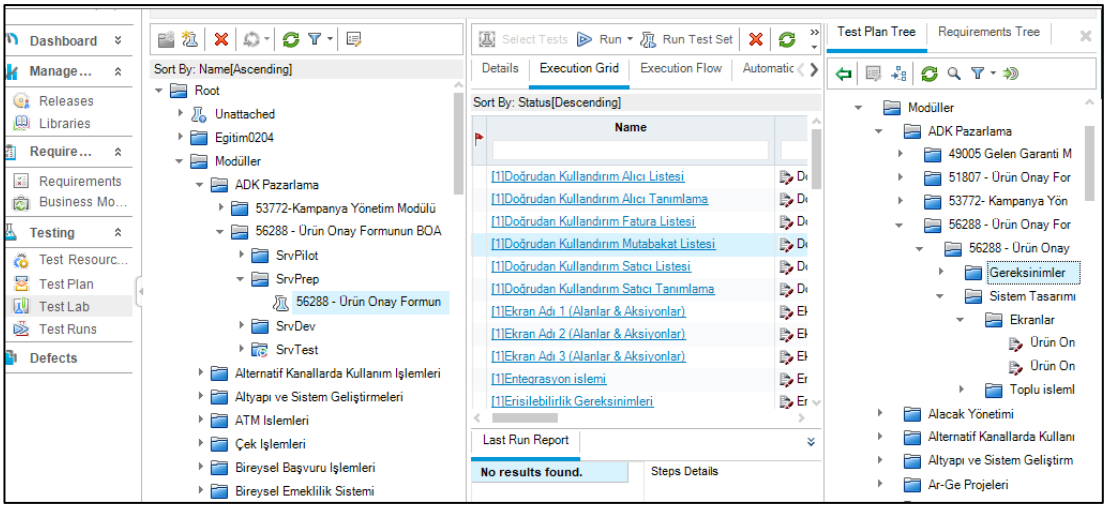
Test Lab yapısı çalıştırılan senaryonun Başarılı (Pass), Başarısız (Fail), Çalıştırılmadı (NoRun) statülerini göstermek için kullanılır. Bu kısımda ayrıca çalıştırma geçmişleri de tutulur. Çalıştırma geçmişleri genelde otomatik testlerde kullanılır. Periyodik olmayan testlerde çalıştırma geçmişinin çok önemi yoktur.



Şekil 6.9. Cycle bağlama ekranı

28. Oluşturulan Test Lab klasörleri testi yapılacak ortama göre ilgili sürüm (cycle) klasörüne Sürüme Ata (Assign To Cycle) menüsüne tıklanarak seçilir (Şekil 6.9.).

Şekil 6.9.'da SrvPrep klasöründe işlem yapılmaktadır. Prep adı entegresyon ortamı için kullanılmaktadır. Yapılan bağlantı Atanılan Sürümü Sil ("Clear Assigned Cycle") menüsüne tıklanarak silinebilir. SrvTest klasörü üzerinde bulunan yuvarlak simge ise test klasörünün ilgili cycle klasörüne bağlandığını göstermektedir.

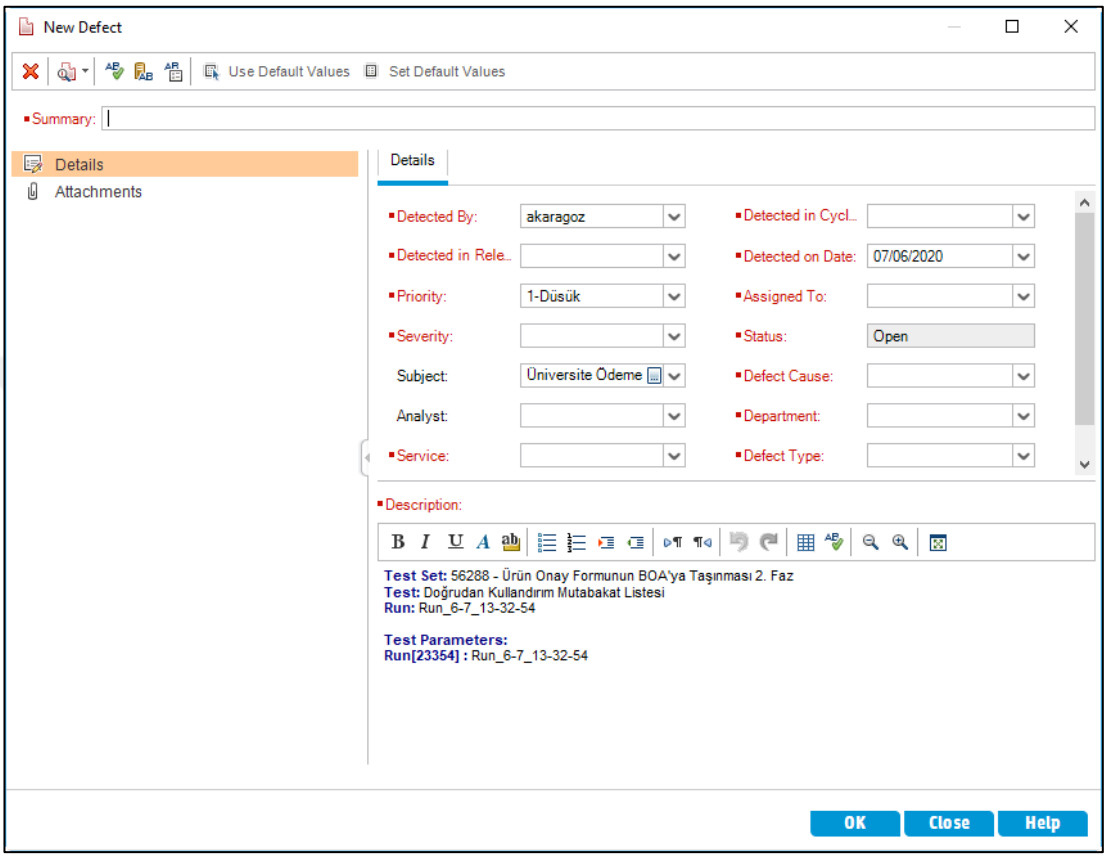


Şekil 6.10. Senaryoları seçme ekranı

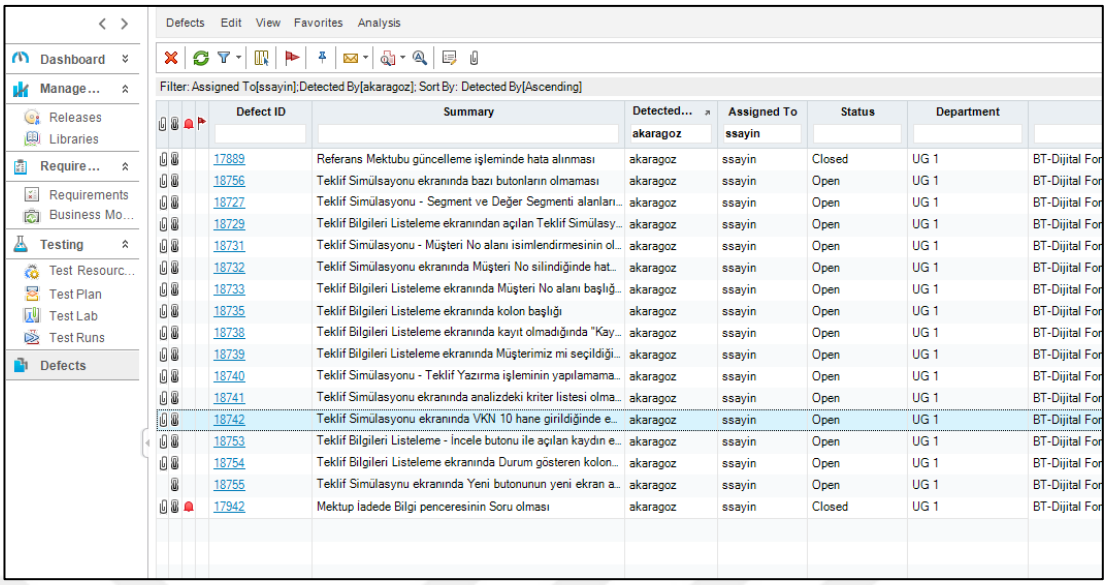
29. Test Lab menüsünde iken proje klasörü içindeki Test seçilir (Şekil 6.10.).
30. Test seç (Select test) butonuna basılır (Şekil 6.10.).
31. Test Planı Ağacı (Test Plan Tree) sekmesine basılır (Şekil 6.10.).
32. Klasör ağacından Modüller kökünden Test Plan (Senaryo Klasörü) proje klasörü seçilir (Şekil 6.10.).
33. Tamam (OK) butonuna tıklanır (Şekil 6.10.).
34. Çalıştır (Run) butonuna tıklanır. Açılan pencerede Yeni Hata (New Defect) butonuna tıklanır (Şekil 6.10.).
35. Yeni Hata (New Defect) ekranında gerekli alanlar doldurulur (Şekil 6.11.).
36. Ekler (Attachments) sekmesinden hata ilgili medya veya dokümanlar eklenir (Şekil 6.11.).
37. Tamam (OK) butonuna tıklanır (Şekil 6.11.).
38. Hatalar (Defects) menüsünden raporlanan hatalar listelenir ve güncellenir (Şekil 6.12.).
39. Hata raporu veya kaydı atanan kişiye uygulamanın otomatik tetiklemesiyle mail olarak iletilir.

Yeni Hata (New Defect) ekranı hata detaylarının girildiği ekrandır. Bu ekranda uygulama özelliklerinden gelen çeşitli alanlar bulunmaktadır. Bu ekrana kullanım

ihtiyacına göre hata kaynağı, hata türü, hatanın alındığı işletim sistemi vb. çeşitli alanlar eklenebilir ve doldurulması zorunlu hale getirilebilir.



Şekil 6.11. Yeni hata giriş ekranı



Şekil 6.12. Hata listeleme ekranı

6.3. Geliştirilen Hata Yönetiminde Hata Raporlama Adımları

1. Gereksinim Yönetim ekranı açılır (Şekil 5.11.).
2. Modül Ağacında analizin olduğu kırılıma tıklanır (Şekil 5.10.).
3. Hata girilecek Gereksinim seçilir (Şekil 5.13).
4. Hata Oluştur butonuna tıklanır. Hata Tanımlama ekranı Gereksinim Yönetimi ekranında pencere olarak açılır (Şekil 5.17.).
5. İlgili alanlar doldurulurur (Şekil 5.17.).
6. Ekle butonuna tıklanır. Hata ile ilgili medya veya dokümanlar eklenir (Şekil 5.18.).
7. Kaydet veya Kaydet/Yeni butonuna tıklanır (Şekil 5.17.).
8. Kaydedilen hata Gereksinim Yönetimi ekranında listelenir. Tıklanılan gereksinime ait hatalar ekranın alt bölmesinde listelenir ve güncellenebilir (Şekil 5.18.).

Kaydet butonuna basıldığında hata kaydolur ve Hata Tanımlama ekranı otomatik kapanır.

Kaydet/Yeni butonuna tıklanırsa hata kaydolur ancak Hata Tanımlama ekranı kapanmaz Hata Başlığı alanı temizlenir. Hata girişine devam edilir.

Şablon Kaydet butonuna tıklanırsa hata kaydı sonraki hata girişleri için şablon olarak kullanılır.

6.4. Mevcut ile Geliştirilen Hata Yönetiminin Karşılaştırılması

Mevcut ile geliştirilen hata yönetimi aşağıda detaylı bir şekilde ele alınmış ve Tablo 6.1.'de özet bir gösterim yapılmıştır.

- Tablo 6.1.'de mevcut hata yönetiminde giriş eforu var geliştirilen hata yönetiminde giriş eforu yok olarak gösterilmiştir. Geliştirilen hata yönetimi ekranı firmanın kendi uygulaması içinde olduğundan ayrıca bir giriş eforu oluşmamaktadır.
- Geliştirilen hata giriş ekranı firmanın kendi uygulaması olduğundan lisans, bakım vb. maliyetleri büyük miktarda düşmüştür. Lisans, bakım maliyetleri tabloda mevcut hata yönetiminde var geliştirilen hata yönetiminde yok olarak gösterilmiştir (Tablo 6.1.).
- Geliştirilen hata yönetimi ekranları firma uygulamasının içinde olduğundan tüm kullanıcılara yetki verilebilmektedir. Bu özellik Tablo 6.1.'de Kullanıcı Deneyimi maddesinde gösterilmiştir.
- Geliştirilen hata yönetimi ekranları firma uygulaması içinde olduğundan ekran kullanımları için ayrı bir kullanım eğitimi maliyeti gerekmemektedir. Bu özellik Tablo 6.1.'de Kullanıcı Deneyimi maddesinde gösterilmiştir.
- Geliştirilen hata yönetiminde son kullanıcı olan iç müşteri rahatlıkla hata girebilmektedir. Bu özellik Tablo 6.1.'de Kullanıcı Deneyimi maddesinde gösterilmiştir.
- Mevcut Hata Yönetiminde 39 adımda hata raporu oluşturulabilmektedir. Geliştirilen hata yönetimi ekranlarında ise 8 adımda hata raporu oluşturulabilmektedir. Tablo 6.1.'de Hata raporlama işlem adım sayısı maddesinde gösterilmiştir.

- Mevcut hata yönetimindeki testin yapıldığı ortam için Sürüm (Release) ve Alt Sürüm (Cycle) oluşturularak testin yapıldığı ortam ayrıştırılmaktadır. Geliştirilen hata yönetimi ekranlarında ise bu bilgi hata girişinde alınmaktadır.
- Geliştirilen hata yönetimi ekranında hata detaylarında ekran çözünürlüğü, versiyon, işletim sistemi bilgileri gibi hatanın çözümüne katkı sağlayacak bilgiler de hata raporlamada alınmaktadır (Tablo 6.1.).
- Geliştirilen hata yönetimindeki Kaydet/Yeni, Şablon Kaydet butonu yeni hata girişlerini oldukça kolaylaştırmış her hata girişinde alanların tekrar doldurulmasına gerek kalmamış alanlar dolu olarak gelmiştir. Tablo 6.1.'de Hata şablonu maddesinde gösterilmiştir.
- Geliştirilen hata yönetimi firmanın kendi ekranları olduğundan her ekip kendilerine özel yönetimsel raporlar alabilmektedir (Tablo 6.1.).
- Geliştirilen hata yönetiminde hata çözüm süreleri uzadığında istenilen uyarılar sistemsel olarak otomatik yapılabilmektedir. Tablo 6.1.'de İstenilen ek sistematik uyarı mesajları maddesinde gösterilmiştir.
- Geliştirilen hata yönetiminde yapılan yorumlar bir başka kullanıcı tarafından değiştirilememektedir.

Tablo 6.1. Mevcut ve geliştirilen sistemin karşılaştırılması

Kriterler	Mevcut Hata Yönetimi	Geliştirilen Hata Yönetimi
Giriş eforu	Var	Yok
Kullanıcı tanımlama	Var	Yok
Lisans, bakım maliyeti	Çok	Az
Kullanıcı deneyimi	Kötü	İyi
Hata raporlama işlem adım sayısı	39	8
Hata şablonu	Yok	Var
İstenilen hata detayları	Yok	Var
İstenilen ek sistematik uyarı mesajları	Maliyeti çok	Maliyeti az
İstenilen ek geliştirmeler	Maliyeti çok	Maliyeti az
İstenilen raporlamalar tipleri	Yok	Var

BÖLÜM 7. TARTIŞMA VE SONUÇ

Geliştirilen hata yönetimi kullanıcı deneyimi ve ihtiyaçları karşılama konusunda mevcut hata yönetiminden birçok üstün özelliğe sahiptir ancak daha avantajlı hale getirilibilecek geliştirilebilir yönleri bulunmaktadır;

- Geliştirilen hata yönetiminde gereksinim eşittir senaryo şeklinde bir yapı oluşmuştur. Bunun ne kadar verimli olacağı tartışılmalıdır. Ek senaryo girişlerinin sağlanması gerekir.
- Mevcut hata yönetimindeki gibi ekran bazlı hata girişinin de olması kullanıcı ara yüzü testlerini daha verimli hale getirecektir.
- Üzerinde işlem yapma durumunda olan açık, kapatılmamış olan personel terfi, tayin veya işten ayrıldığında sistem otomatik olarak ilgili personelin yöneticisine hata kaydını atamalıdır.
- Geliştirilen hata yönetiminde kullanıcılar kendi yorumlarını güncelleyememektedir. Kullanıcıların kendi yorumlarını güncelleyebilmeleri yeniden değerlendirilmelidir.
- Hata listesinde çoklu seçim yapılarak güncelleme yapılabilirse büyük bir avantaj olur.
- Parametre ekranları yapılarak mail atma, rapor maili gönderme gibi işlemler özelleştirilebilir.
- İleri aşamalarda Gereksinim tarafında anahtar kelimeler ile birlikte otomatik senaryolar oluşturulabilir.
- İlerleyen aşamalarda model bazlı test senaryoları oluşturularak test senaryoları otomatik olarak sisteme atılabilir.

Sonuç olarak kullanılan ticari amaçlı üretilmiş uygulamalar üzerinde işleyişe göre değişiklik yapılabilmesi sınırlıdır. Bu durumda kendi işleyişinize uygun bir hata

yönetimi yapılması, bu yapılan geliştirilen uygulamanın kullanıcı dostu olması daha ilk etapta çalışanın motivasyonunu artırmıştır. Bu uygulamanın geliştirilebilir olması esnekliği artırmış projeyi sınırlandırıp çıkmaza sokmamış hızlı şekilde kullanıma alınmasını kolaylaştırmıştır. Karşılaştırma bölümünde de anlatıldığı üzere birçok fayda sağlamıştır. 39 adımda yapılan bir hata raporlama işlemi 8 adıma indirilmiştir. Bu bir hata girişi maliyetini yaklaşık %80 azaltmıştır. Buna ek olarak geliştirilen uygulamanın karmaşık olmaması, ayrı bir login ekranı olmaması, gereksinimlerle senaryoların aynı ekran üzerinde görülebilmesi, hata şablonu oluşturulabilmesi hata girişi maliyetini yaklaşık %5 daha azaltmış ve böylece sağlanan fayda yaklaşık %80'e ulaşmıştır.

KAYNAKLAR

Algan, S. 2009. “Her Yönüyle C#”. Pusula Yayıncılık ve İletişim Ltd. Şti, İstanbul, s. 780.

Aytekin, A.İ., Tüzün, E.&ark. 2015. Ahsen İkbal Aytekin, Eray Tüzün, Yagup Macit, Bedir Tekinerdoğan, Uygulama Yaşam Döngüsü Yönetimi- Sistematik Eşleme Çalışması, UYMS, 2015.

Boehm, B., Rombach, H.D., Zelkowitz, M. V. 1998. “Foundations of Empirical Software Engineering”, Springer, USA.

Chappell, D. 2008. “What is Application Lifecycle Management?”, Chappell Associates.

Cirit, S. 2010. “Pardus Yazılım Testleri ve Hata Takip Sistemi”, Tübitak, İstanbul Bilgi Üniversitesi.

Craig, R. ve Jaskiel, S. 2002. Systematic Software Testing. Boston: Artech House Publishers, s. 4.

Demirli, N. 2008. Yüksel, İ., “Visual C# .NET 2.0”, 1107

Eveleens, J. L. ve Verhoef, C. 2010. The rise and fall of the Chaos report figures. Software, IEEE, 27(1), 30-36. doi:10.1109/ms.2009.154 International Association for Management of Technology IAMOT Conference Proceedings 355.

Hastie, S. ve Wojewoda, S. 2015. Standish Group 2015 Chaos Report- Q&A with Jennifer Lynch.

Hooks, I. 1993. “Writing Good Requirements”, 3. International Symposium of the INCOSE – Sayı 2.

Hurwitz J.S., Bloor R., Baroudi C., 2006, “Service Oriented Architecture For Dummies” Wiley Publishing inc.,8.

IBM 2008. "Collaborative Application Lifecycle Management with IBM Rational Products (Rapor No: SG24-7622-00)". New York: International Business Machines Corporation (IBM).

IEEE Standart Glossary of Software Engineering Terminology. 1990. New York: IEEE, The Institute of Electrical and Electronics Engineer, 610.12.

Joseph, N., Marnewick, C. & Santana, MJ. 2016. The 25th International Conference for Management of Technology, 335-355.

Joseph, N., Erasmus, W. ve Marnewick, C. 2014. The Idle State of Information and Communication Technology Project Management. Journal of African Business, 15(3), 184-196. doi:10.1080/15228916.2014.95664.

Kaner, C., Falk, J., Nguyen H.Q. 1999. Cem Kaner, Jack Falk, Hung Quoc Nguyen. Testing Computer Software. Testing Computer Software. s.l.: Wiley Computer Publishing, 1999, s. 344.

Kendall E. K. ve Kendall J. E. 2013. System Analysis and Design, 9th ed. Prentice Hall, ISBN: 013302344.

Kolkhorst, B. 2000. "Simple Software Defect Categorization for Defect Prevention", Software Management&Applications of Software Measurement.

Molu, F., Karagoz, A.B. 2016. "Yazılım Test Sürecinde Hata Yönetimi ve Finans Sektöründen Örnek Uygulama" Akademik Bilişim Konferansı, Pdf 143.

Myers, G. 2004. The Art of Software Testing. New Jersey: John Willey & Sons, Inc,8.

Öztürk, M.M. 2016. Tekrar eden veri analizini kullanarak yazılım geliştirme için iyileştirilmiş hata tahmini. Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar ve Bilişim Mühendisliği Bölümü, Doktora Tezi.

Sharp, J. 2009. Adım Adım Microsoft Visual C# 2008. Arkadaş Yayın Evi, Ankara, 696.

Skeet J. 2013. "C# in Depth", 3rd edition, Manning Publications.

Şahinoğlu, M., Sarı, M., Kurt, A., Kurnaz, S., Özbek, M. 2013. "Yazılım Test Sürecinde Problemler ve Çözüm Önerileri", 7. Ulusal Yazılım Mühendisliği Sempozyumu, İzmir.

Url-1: "Agile (Çevik) Metod Ve Agile Test Yöntemi", 2016. https://keytorc.com/blog/agile-cevik-metod-ve-agile-test-yontemi/, Erişim Tarihi: 11.11.2020.

Url-2: "Business Oriented Architecture (BOA) Hakkında", 2014. Kuveyt Türk personel kullanıcı linki, Erişim Tarihi: 11.11.2020.

Url-3: But, C. 2018. "Define Testing Strategy using the Testing Pyramid" https://medium.com/@Colin_But/define-testing-strategy-using-the-testing-pyramid-1dabee37e823, Erişim Tarihi: 15.03.2020.

Url-4: "Defect Management Process" http://www.defectmanagement.com, Erişim Tarihi: 12.11.2020.

Url-5: Ergin, T., "Hata Yönetimi Süreci", http://www.csharpnedir.com/articles/read/?id=254, Erişim Tarihi: 12.11.2020.

Url-6: Eser, O. 2019. "Yazılım Testine İlk Adım" https://medium.com/@ozaneseriu/yazilim-testi%CC%87ne-i%CC%87lk-adim-17daed6e63ab, Erişim Tarihi: 12.12.2020.

Url-7: "Industry making the transition to Business Oriented Architecture", Moran, A. Digital Journal, http://www.digitaljournal.com/article/343790, Erişim Tarihi: 17.10.2020.

Url-8: “Is BOA The New SOA?” https://www.information-management.com/news/is-boa-the-new-soa, Erişim Tarihi: 15.10.2020.

Url-9: ISTQB 2018. Sertifikalı Test Uzmanı Temel Seviye Ders Programı, https://www.turkishtestingboard.org/wp-content/uploads/2020/06/Turkce-Foundation-Level-Syllabus-T%C3%BCrk%C3%A7e-2018_20200629.pdf, Erişim Tarihi: 15.06.2020.

Url-10: Kossai, M., & Piget, P. (2014). Adoption of information and communication technology and firm profitability: Empirical evidence from Tunisian SMEs. The Journal of High Technology Management Research, 25(1), 9-20. https://www.sciencedirect.com/science/article/abs/pii/S1047831013000370?via%3Dihub, Erişim Tarihi: 23.03.2020.

Url-11: Kuvey Türk Hakkımızda, https://www.kuveytturk.com.tr/hakkimizda, Erişim Tarihi: 03.03.2020

Url-12: Mikrosoft, “What is WPF?”, https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019, Erişim Tarihi: 10.10.2020

Url-13: Tozzi, C. 2016. Quality Assurance and Software Testing: A Brief History, https://saucelabs.com/blog/quality-assurance-and-software-testing-a-brief-history, Erişim Tarihi: 15.03.2020.

Yazılım Test ve Kalite Derneği 2014. ISTQB Yazılım Testi Terimler Sözlüğü, V1, 11.

ÖZGEÇMİŞ

Ayşe Betül Karagöz, ilk ve ortaöğrenimini Ünye'de tamamladı. Yakındoğu Üniversitesi Bilgisayar Mühendisliği Bölümü'nü bitirdi. Mezun olduktan sonra Bilişim sektöründe iş hayatına başladı. Yazılım Test Mühendisliği alanında arge faaliyetlerinde bulundu. Bilişim sempozyumlarında bildiriler sundu. Katıldığı seminer, konferans ve eğitimlerle mesleki gelişimini sürdürdü. Bu kapsamda Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Bölümü'nde tezli yüksek lisans eğitimi alan Ayşe Betül Karagöz, bilişim alanındaki arge çalışmalarını finans sektöründe sürdürmektedir.