T.C.
NECMETTİN ERBAKAN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İŞ GEREKSİNİMİ ODAKLI TEST SENARYOLARI ÜRETİM MODELİ

Büşra ÇÖLLÜ

YÜKSEK LİSANS

Endüstri Mühendisliği Anabilim Dalı

Mayıs-2017
KONYA

TEZ KABUL VE ONAYI

Büşra ÇÖLLÜ tarafından hazırlanan "İŞ GEREKSİNİMİ ODAKLI TEST SENARYOLARI ÜRETİM MODELİ " adlı tez çalışması 04/05/2017 tarihinde aşağıdaki jüri tarafından oy birliği / oy çokluğu ile Necmettin Erbakan Üniversitesi Fen Bilimleri Enstitüsü ENDÜSTRİ MÜHENDİSLİĞİ Anabilim Dalı'nda YÜKSEK LİSANS/DOKTORA TEZİ olarak kabul edilmiştir.

Jüri Üyeleri	**İmza**
Başkan Prof. Dr. Mehmet AKTAN	……..
Danışman Prof. Dr. Mehmet AKTAN	……..
Üye Yrd. Doç. Dr. Ahmet Reha BOTSALI	……..
Üye Prof. Dr. Orhan ENGİN	……..

Yukarıdaki sonucu onaylarım.

Prof. Dr. Ahmet Coşkun
FBE Müdürü

TEZ BİLDİRİMİ

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Büşra ÇÖLLÜ

ÖZET

İŞ GEREKSİNİMİ ODAKLI TEST SENARYOLARI ÜRETİM MODELİ

Büşra ÇÖLLÜ

Necmettin Erbakan Üniversitesi Fen Bilimleri Enstitüsü

Endüstri Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Mehmet Aktan

2017, 73 Sayfa

Jüri
Prof. Dr. Mehmet Aktan
Yrd. Doç. Dr. Ahmet Reha BOTSALI
Prof. Dr. Orhan ENGİN

'İŞ GEREKSİNİMİ ODAKLI TEST SENARYOLARI ÜRETİM MODELİ' adlı bu tez çalışması kapsamında, yazılım ve test senaryosu üretim süreçleri analiz edilmiş ve test sürecinin yazılım yaşam döngüsüne test senaryolarının oluşturulmasıyla erken dahil edilmesi incelenmiştir. Modelin yazılım üretim sistemine uygulanmasıyla insan kaynaklarının tasarrufunu sağlayacak test senaryoları üretimi süreci, yöntem ve teknikleri araştırılmıştır.

Literatürdeki çalışmalara bakıldığında, test senaryolarının gereksinim odaklı olmadığı ve üretim sürecinin genellikle test uzmanlarına bırakıldığı görülmüştür. Bu çalışmada ise test senaryoları analiz ve tasarım aşamasından itibaren sistem tarafından otomatik olarak üretilmesi, birçok bilgi kataloglarından geçmiş verilerin kullanımı ve hataların erken çözümünün yazılım yaşam döngüsüne kattığı verimlilik ve tasarrufu açıklanmıştır.

Büyük ölçekli kurumsal firmalarda uygulanabilir, iş gereksinimi odaklı test senaryosu üretme modeli önerilmiş ve içermesi gereken özellikler anlatılmıştır. İş gereksinimlerini ekran tanımı, tasarımı, ekrandan alınacak veri tipleri, aksiyonlar ve özellikleri, tanımlı işlemler ve sistem entegrasyon tanımlarının yapılabileceği tanım tabanlı bir araç aracılığı ile alınabileceği ifade edilmiştir. Önerilen model, iş gereksinimleri tanımları referans alınarak senaryolaştırılmış kaynak kodu, test senaryolarını içeren test kaynak kodları, test senaryo dokümanı, birim testi kodları, otomasyonlara bağlı çalıştırılabilir kod blokları, analiz dokümanı ve test raporu üretebilmelidir. Bu sayede kurala bağlı üretilebilen birçok senaryo ve çalıştırılabilir kodlar üretilmiş olup, yazılım mühendisi, sistem analisti, test mühendisi gibi pahalı kaynakların yerinde ve verimli kullanılması hedeflenmiştir. Önerilen sistem bu çıktıları üretmek için güçlü, oturmuş, olgunlaşmış bir kurumsal mimariye ihtiyaç duymaktadır.

Anahtar Kelimeler: Erken Test, FMEA, İş Gereksinimi Odaklılık, Senaryolaştırılmış Kaynak Kod, Test Otomasyonu, Test Senaryosu

ABSTRACT

MS THESIS

BUSİNESS REQUIREMENT ORIENTED TEST SCENARIO GENERATION MODEL

Büşra ÇÖLLÜ

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE OF NECMETTİN ERBAKAN UNIVERSITY
THE DEGREE OF MASTER OF SCIENCE
IN INDUSTRIAL ENGINEERING

Advisor: Prof. Dr. Mehmet Aktan

2017, 73 Pages

Jury
Prof. Dr. Mehmet AKTAN
Asst. Prof. Dr. Ahmet Reha BOTSALI
Prof. Dr. Orhan ENGİN

In this thesis 'BUSINESS REQUIREMENT ORIENTED TEST SCENARIO GENERATION MODEL', software and test scenario generation process has been analyzed and the early involvement of the test cycle with the creation of test scenarios in the software development life cycle has been investigated. With the application of the model to the software production system, the production process, methods and techniques of the test scenarios that will save human resources have been investigated.

When we look at studies in the literature, it has been found that test scenarios are not requirements oriented and that the production process of test scenarios is often left to test experts. In this study, the test scenario is automatically generated by the system from the analysis and design phase, the use of historical data from many information catalogs and the productivity and savings that the early solution of the errors add to the software life cycle.

A model for producing a business-focused test scenario that is applicable to large-scale enterprise firms has been proposed and describes the features that need to be included. It has been stated that business requirements can be obtained through screen definition, design, data types to be taken from the screen, actions and properties, defined operations and a definition based tool which can define system integration. The proposed model should be able to produce scripted source code with reference to business requirement definitions, test source codes containing test scenarios, test scenario document, unit test codes, executable code blocks connected to automations, analysis document and test report. In this way, many scenarios and executable codes that can be produced according to the rule are produced and it is aimed to use the expensive resources such as software engineer, system analyst and test engineer in place and efficiently. The proposed system needs a strong, steady, ripe enterprise architecture to produce these outputs.

Keywords: Business Requirement Oriented, Early Testing, FMEA, Scripted Source Code, Test Automation, Test Scenario

ÖNSÖZ

Tez çalışmam boyunca yardım ve desteğini esirgemeyen Necmettin Erbakan Üniversitesi Endüstri Mühendisliği Öğretim Üyesi ve tez danışmanım Sayın Prof. Dr. Mehmet Aktan'a, tez çalışmama kaynak olarak katkılarından dolayı Kuveyt Türk Bilgi Teknolojileri çalışanlarına, hayatım boyunca maddi ve manevi desteğini hiçbir zaman esirgemeyen değerli aileme teşekkür ederim.

Bu çalışmanın, bu konuda yapılacak başka çalışmalara faydalı olmasını temenni ederim.

Büşra ÇÖLLÜ
KONYA-2017

İÇİNDEKİLER

KISALTMALAR

EFT : Elektronik Fon Transferi
FMEA : Failure Mode Effect Analysis
IBAN : International Bank Account Number
İGOT : İş Gereksinimi Odaklı Test Senaryoları Üretimi
STLC : Software Test Life Cycle
UML : Unified Modeling Language

1. GİRİŞ

Yazılım gün geçtikçe daha fazla sistemde kullanılmaktadır. Yazılımın bu kadar çok yayılması yazılım karmaşıklığını artırmaktadır. Karmaşık yazılımlar da hataya daha açık yazılımlardır ve test edilmeleri daha zordur. Doğru yaklaşım ve yöntemle yapılan yazılım testleri her ne kadar yazılımdaki hataların tespit edilmesine yardımcı olsa da yazılım testlerinin amacı yazılımda hata bulunmadığını kanıtlamak değildir. Yazılım testlerinin amacı yazılımda hatalar bulunabileceğini göstermek ve kalite durumunu raporlamaktır. Yazılım testleri için ne kadar zaman ve kaynak ayrılırsa ayrılsın yazılımı tüm girdi ve buna karşılık gelen tüm çıktıları ile test etmek projedeki zaman ve bütçe kısıtları nedeniyle mümkün değildir(NASA,2004).

Yazılım içeren sistemler tümleştikten sonra yapılan kabul testleri tüm olası durumları test etmede yetersiz kalabilmektedirler. Bu sebeple yazılım sistemlerini oluşturan yazılım bileşenleri kendi içlerinde çok iyi test edilmelidirler. Bir yazılım bileşeninin çok iyi test edilebilmesi için bileşenin test edilebilirliğinin yüksek olması gereklidir. Yazılım test edilebilirliği basitçe yazılımın test faaliyetlerini ne kadar desteklediğidir. Yazılım test edilebilirliği doğrudan ölçülebilen bir yazılım niteliği değildir. Test edilebilirlik, yazılım kalitesi altında değerlendirilir. Kaliteli yazılımda, test edilebilirlik yüksek olacağı için hata çıkma olasılığı düşüktür ve yazılım isteklerin değişmesi veya yeni isteklerin eklenmesi gibi durumlar test edilebilirliği çok etkilemez. Bunun aksine kalite nitelikleri düşük olan yazılım hatalara açıktır, test edilebilirliği düşüktür ve gelen değişiklik talepleri yazılımı sürdürülebilir olmaktan çıkarabilir. Yazılım kalitesinin ölçülmesi amacıyla yazılım ve test metrikleri kullanılır (Özçelik,2015).

Yazılım ürünleri için ne kadar iyi test senaryoları hazırlanır ve çalıştırılırsa kalite durumu da o kadar iyi ortaya konulabilir. Bu senaryolar içerisinde sürekli tekrar eden temel adımları içeren metriklerin otomatize edilerek karmaşık senaryolar için insan kaynaklarının kullanılması daha verimli çalışma için uygundur. Böylelikle daha çok ve daha karmaşık senaryolar ile test metrikleri daha iyi ölçümlenerek daha kaliteli yazılım ürünleri ortaya koyulabilir.

Bu tez çalışmasında test senaryolarının otomatize edilebildiği insan kaynaklarının daha efektif çalışabileceği İş Gereksinimi Odaklı Test Senaryoları Üretimi(İGOT) Modeli tasarlanmıştır.

2. KAYNAK ARAŞTIRMASI

2.1. Yazılım Yaşam Döngüsü

Yazılımın ürününün hem üretim hem de müşterideki kullanım süreci boyunca geçirdiği tüm aşamalar yazılım geliştirme yaşam döngüsü olarak adlandırılır. Yazılım geliştirme süreci, zamanlamaya dayalı ve içerik olarak bölünmüş aşamalardan oluşmaktadır. Bu sayede yazılım planlı bir şekilde geliştirilmektedir. İGOT modeli uygulanırken yazılım yaşam döngüsü adımları dikkate alınır. Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu aşamalar sürekli bir döngü biçiminde ele alınır. Döngü içerisinde her hangi bir aşamada geriye dönmek ve tekrar ilerlemek söz konusudur. Temel yazılım geliştirme aşamaları aşağıdaki gibidir:

Planlama: Yazılım yaşam döngüsünün ilk aşamasıdır. Temel gereksinimler belirlenir, proje için fizibilite çalışmaları yapılır (maliyetlerin ve sistemin yararlarının tanımlanması) ve proje planlaması gerçekleştirilir.

Analiz: Yazılım yaşam döngüsünün en önemli aşamalarından biri olan analiz sürecinde projenin tüm işlevleri detaylı olarak belirlenir. Bu belirtimlere bağlı olarak sistem gereksinimleri netleşir ve buna bağlı talepler hazırlanır. Kısaca, analiz sürecinde projenin tüm detayları ortaya çıkartılır(Kızmaz,2012). Bu çalışma müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi gibi rollerin bir araya geldiği gruplar tarafından yapılabilir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanım dokümanları ve test plan dokümanları da oluşturulabilir.

Tasarım: Yazılımımızın veya sistemimizin tasarımları yapılır. Projeleri çizilir. İnşaat projeleri gibi düşünülebilir. Planlama ve tanımlaya göre bir tasarım çizilir. Kararlar verilir, seçimler yapılır, örneğin yazılımın ekranları, ekranlarda neler bulunacağı, hangi ekranlara nasıl geçileceği, fonksiyonel olarak hangi adımların oluşturulacağı, hatta yazılımın bileşenleri, modülleri bu aşamada tasarlanır. Bir sonraki adım olan uygulama/geliştirmeye bütün kararlar verilmiş olarak geçilmesi beklenir. Geliştirme aşamasında herhangi bir soru veya karar bırakılmaz(Seker,2015).

Yazılım tasarımında kullanılan en önemli tekniklerden birisi Soyutlama (Abstraction) dır. Soyutlama, problemlerin çözümlerini kolaylaştırmak için nesnelerin, olayların ve durumların bazı özelliklerin görmezden gelinmesidir. Problemi basitleştirerek en önemli kısımlarına odaklanmayı sağlar. Modelleme ise temel tasarım

aracı olup statik ve dinamik modellerden bahsetmekten mümkündür. Statik model, programın çalışması sırasında değişmeyen yönlerini ifade etmek için kullanılırken (sınıf ve nesne modelleri), dinamik model programın çalışması sırasındaki işleyişi ifade etmek için kullanılır (durum ve sıra diyagramları) (Kılınç,2013).

Gerçekleştirim (Kodlama ve Test): Bu aşama, yazılım projeleri için kodlama olarak düşünülebilir veya sistemin yaşatılmaya başlandığı ilk örneklerinin çıkmaya başlandığı aşama olarak düşünülebilir. Daha önceden tasarım aşamasında karar verilen ortama uygun olarak, yine tasarım aşamasında verilen kararlar doğrultusunda projenin gerçekleştirilmesine başlanır. Bir anlamda daha önceden mimari projesi tamamlanmış inşaatın gerçekten yapılmaya başladığı aşamadır, yazılım projelerinin kodlandığı aşamadır(Seker,2015).

Kodlama süresince ve kodlama sonrasında yapılan diğer önemli aşama testtir. Erken test et yaklaşımı ile (early testing) hareket edip, analiz aşamasından itibaren test bakış açısına sahip olmamız hata yapma oranımızı ve maliyetleri (zaman, para, prestij vb.) düşürecektir. Birim testleri, duman testleri, yanlış değer testleri, kabul testleri, kullanım senaryo testleri, yük testleri, kullanıcı kabul testi, yoldan geçen adam testi, test otomasyonu gibi sürece ve duruma göre uygulanabilecek çok farklı kategoride ve derinlikte test türü bulunmaktadır(Kılınç,2013).

Teslim ve Bakım: Tüm test aşamaları tamamlandıktan sonra yazılım ürünün sahaya teslim edilebilir bir versiyonu çıkartılır ve teslim aşaması gerçekleştirilir. Proje yayına alındıktan sonra oluşabilecek hataların giderilmesi, yazılımın iyileştirilmesi ve yeni işlevlerin eklenmesi bakım süreçleridir. Bu süreç zarfında kullanıcılardan gelen bilgiler doğrultusunda bu istekler gerçekleştirilmektedir(Kızmaz,2012).

Projenin teslimi ile birlikte bakım aşaması da başlar. Hata giderici, önleyici, altyapıyı iyileştirici, ürüne yeni özellikler ekletici gibi farklı bakım faaliyetleri mevcuttur.

2.1.1. Yazılım Belirtim Yöntemleri (Specifications)

Bir çekirdek sürece ilişkin fonksiyonları yerine getirmek amacıyla kullanılan yöntemlerdir.

Süreç Akışı İçin Kullanılan Belirtim Yöntemleri: Süreçler arası ilişkilerin ve iletişimin gösterildiği yöntemlerdir (Veri Akış Şemaları, Yapısal Şemalar, Nesne/Sınıf Şemaları).

Süreç Tanımlama Yöntemleri: Süreçlerin iç işleyişini göstermek için kullanılan yöntemlerdir (Düz Metin, Algoritma, Karar Tabloları, Karar Ağaçları, Anlatım Dili).

Veri Tanımlama Yöntemleri: Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemlerdir (Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü) (Kılınç,2013).

Belirtilen bu yöntemler İGOT modelinde kod üretiminde kaynak amaçlı kullanılabilir.

2.1.2. Yazılım Süreç Modelleri

Yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, uygulanma yöntemini ve sıralamasını tanımlayan modellerdir. Karmaşıklığı azaltıp olası sorunların önüne geçmede fayda sağlar. Ürünlerin beklenilen kalitede olması süreçlerin kontrol edilmesine bağlıdır. Belli başlı yazılım süreç modelleri aşağıdaki gibidir;

Şelale Modeli (Waterfall Model)

V Modeli (V-shaped Model)

Evrimsel Geliştirme (Evolutionary Development)

Prototipleme (Prototyping)

Spiral Model

Yeniden kullanıma yönelik geliştirme (Re-use based development)

Çevik Modeller (Agile models: XP, Scrum).

Bu tezde anlatılan İGOT modeli birçok yazılım süreç modelinde uygulanabilir. Özellikle yazılımı küçük parçalar halinde gerçeklenen çevik modelin kullanıldığı projelerde İGOT modeli büyük kolaylık sağlamaktadır. V modelinin kullanıldığı projelerde de sürecin içerdiği adımların gerçekleştirilmesinde kolaylık sağlamaktadır. Prototipleme modeli için de İGOT modelinde ekranların tanımlanması süreç için kaynak oluşturmaktadır.

2.1.3. Testin Yazılım Yaşam Döngüsündeki Yeri

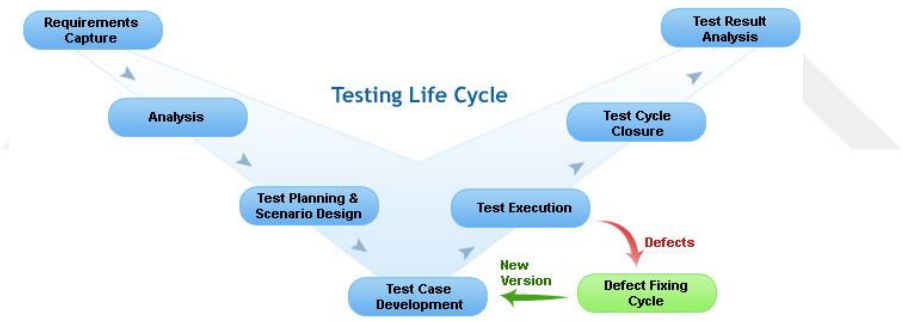
Yazılım testi; yazılım geliştirmenin her aşamada ayrıca ele alınmalıdır. Sadece kodlamadan sonraki test kısmında değil; analiz kısmında statik test tekniklerinden, kodlama kısmında dinamik test tekniklerinden faydalanılmalıdır.

Test işleminin de kendi içerisinde bir yaşam döngüsü vardır. Bu süreç yazılım projelerinin her aşamasında uygulanabilmektedir.

2.1.3.1. Yazılım Test Yaşam Döngüsü

Testin işlemininde standartları ve optimizasyonu vardır. Bu süreç STLC(Software Testing Life Cycle) olarak isimlendirilir. Yazılım Test Yaşam Döngüsü kalite hedefleri yerine getirilmesi için süreçteki bu adımları sırası ile yerine getirebiliyor olmak gerekir. İGOT modelinde yazılım test yaşam döngüsü yazılım projesinin ilk aşamalarından itibaren başlatılmalı ve tüm süreçlere dahil edilmelidir. Örneğin yazılım yaşam döngüsünün planlama aşamasında test politikası ve stratejileri de belirlenebilir.

STLC işlemlerinde, her bir faaliyet planlı ve sistemli bir şekilde ifade edilir. Her aşamanın farklı hedefleri ve çıktıları vardır. Bu aşamalar Şekil 2.1.'de verilmiştir ve aşağıda incelenmiştir.



Şekil 2.1. Yazılım Test Yaşam Döngüsü

Gereksinim Aşaması: Gereksinim Analizi Yazılım Test Yaşam Döngüsünün (STLC) ilk adımıdır . Bu aşamada, Kalite Güvencesi (QA) ekibi, test edeceğimiz ve test edilebilir gereksinimleri anlamaya yönelik gereksinimi anlıyor. Herhangi bir çatışma, eksik veya gerekçenin anlaşılamaması durumunda, QA ekibi, gereksinim detayını daha iyi anlamak için İş Analisti, Sistem Mimarisi, Müşteri, Teknik Müdür gibi çeşitli paydaşları takip eder.

İlk aşamadan itibaren QA, STLC'in teste tabi tutulan Yazılımın içine giriş kusurlarını önlemeye yardımcı olduğu yerlerde yer alır. Gereksinimler Fonksiyonel veya İşlevsiz, Performans, Güvenlik testi olabilir. Projenin ihtiyaç ve otomasyon fizibilitesi de bu aşamada (varsa) yapılabilir(STC,2013).

Planlama Aşaması: Pratik senaryolarından sonra , test planlama test sürecinin ilk adımıdır. Bu aşamada test hedeflerini karşılamak için kullanılacak yardımcı faaliyetler ve kaynakların tespiti yapılır. Planlama sırasında ölçümleri ve bu ölçümleri nasıl izlenebileceğinin yöntemi belirlenmeye çalışılır(Yener,2015).

Sadece gereksinimleri karşılayacak şekilde test planı yapmak yeterli değildir. Test planlaması etkileyen diğer 2 çok önemli faktörler vardır(Yener,2015). Bunlar:

- Test stratejisi organizasyonu : Yani hangi uygulama yada ürün test süreci içerisinde nerede nasıl test edilecek bunun iyi kestirilmesi ve planlanması gerekmektedir. Test stratejisinde bu durum çok önemlidir ve sürecin sağlıklı işletilmesinde belirleyici bir rol oynar.
- Risk analizi / Risk Yönetimi ve hafifletme : Planın sağlıklı şekilde işleyebilmesi için olası riskler hafifletilmeli ve riskler doğru eşleştirilmelidir.

Analiz Aşaması: Analiz aşamasında en doğru soru test edilecek "NE" sorusudur. Temelde gereksinimleri dokümanlar ile belirtilir. Ürüne ait riskleri ve diğer test temel test koşulları belirlemek gerekir. Test gereksinimi izlenebilir ve geri bildirimi olan bir yöntem olmalıdır. Test koşullarının belirlenmesinde bazı gereksinimler ve koşullar vardır.

- Test Seviyeleri ve Test Derinliğinin Belirlenmesi
- Ürünün Karmaşıklık
- Ürün ve Proje Riskleri
- Yazılım Geliştirme Yaşam Döngüsü Çıktısı
- Test Yönetimi
- Ekip Beceri ve Bilgisi
- Paydaşların Durumu(Yener,2015)

Tasarım Aşaması: Bu aşamada testin hangi yöntemler izlenerek uygulanacağı belirlenir. Aşağıdaki aşamalar incelenir:

- Test koşum durumunu göz önünde bulundurmak.
- Yapılacak testi ve test datalarını belirlemek
- Test gereksinimlerini yapabilmek için test ortamlarını kurmak
- Gereksinimi izlenebilmek için ölçümler oluşturmak
- Test kapsamı ölçümleri oluşturmak(Yener,2015)

Uygulama ve Yürütme Aşaması: STLC aşamasında önemli görevi ayrıntılı test olguların yaratılmasıdır. Test koşumu için gerekli olan test regresyon test setinizin olduğuna emin olmanız gerekmektedir. Test durumlarının doğruluğu için kontrol etmek-gözden geçirmek önemlidir. Projelerinizde test otomasyon gereksinimi varsa testlerinizi otomatize edilmesi önemlidir(Yener,2015).

Test Durum Geliştirme ve Test Ortamı kurulumu hazırlandıktan sonra test yürütme aşaması başlatılabilir. Bu aşama test ekibi hazır test planlaması ve önceki aşamadaki hazır test durumlarına dayalı test durumlarını çalıştırmaya başlar.

Test senaryosu bir kez geçtiğinde, aynı durum 'Geçti' olarak işaretlenebilir. Herhangi bir test durumu başarısız olursa, ilgili hata, hata izleme sistemi vasıtasıyla geliştirici ekibe rapor edilebilir ve hata, daha ileri analiz için ilgili test durumu için bağlanabilir. İdeal olarak başarısız olan her test durumu en azından tek hata ile ilişkilendirilmelidir. Bu bağlantıyı kullanarak başarısız olan test durumunu, hatayla ilişkili olarak alabilirsiniz. Geliştirme ekibi tarafından düzeltilen hata sonra, test planlamanıza dayalı olarak aynı test durumu yürütülebilir.

Test durumlarının herhangi biri herhangi bir kusur nedeniyle engellendiyse, bu tür test kılıfları 'Engellendi' olarak işaretlenebilir, dolayısıyla kaç test kalıbında, başarısızlığa, engelleneceğine veya çalışmadığına neden olduğu hakkında rapor alınabilir. Kusurlar düzeltildikten sonra, aynı Başarısız veya Engellenmiş test durumları, işlevselliği yeniden test etmek için tekrar çalıştırılabilir(STC,2013).

Sonuç Aşaması: STLC aşamasının çıkış kriterleri ve raporlama faaliyetlerinin incelendiği aşamadır. Proje ve paydaşlar seçimine bağlı olarak, raporlar (DSR - Günlük Durum Raporu (Daily Status Report) WSR - Haftalık Durum Raporu (Weekly Status Report) gibi türleri vardır(Yener,2015).

Proje yönetiminde test raporları çok önemli bir yer tutar. Yayınlanacak raporda ne kadar test durumunun olduğu, ne kadarının hatalı, ne kadarının doğru, yada senaryolarının çalıştırılma durumu gibi çeşitli test durumlarına ait detaylarına yer verilmelidir. Ayrıca tespit edilen hatalar, test senaryolarının koşumuna engel durumlarda raporlanmalıdır. Test raporları ürün yada projenin anlık durumu ve kullanılabilirlik yüzdesini böylece ortaya koyar. Test raporu ne kadar önemsenirse projenin başarısı o kadar ortaya konulabilir.

Kapatma Aşaması: Test ekibi üyesi toplantısı yapılır ve Test kapsamı, Kalite, Maliyet, Zaman, Kritik İş Hedefleri ve Yazılım'a dayalı döngü tamamlama ölçütlerini değerlendirilir. Tüm adımların durumu tartışılır, hangi alan iyileştirilmeli ve STLC sürecindeki darboğazın iyileştirilmesine yardımcı olacak yaklaşmakta olan test döngülerine girdi olarak mevcut STLC'den ders alınmalıdır. Test durumu ve hata raporu, tür ve şiddete göre kusur dağılımını bulmak için analiz edilir. Test döngüsünü tamamladıktan sonra test kapanış raporu & Test metrikleri hazırlanacaktır. Kusur dağılımını türüne ve ciddiyetine göre test sonuç analizi hazırlanır(STC,2013).

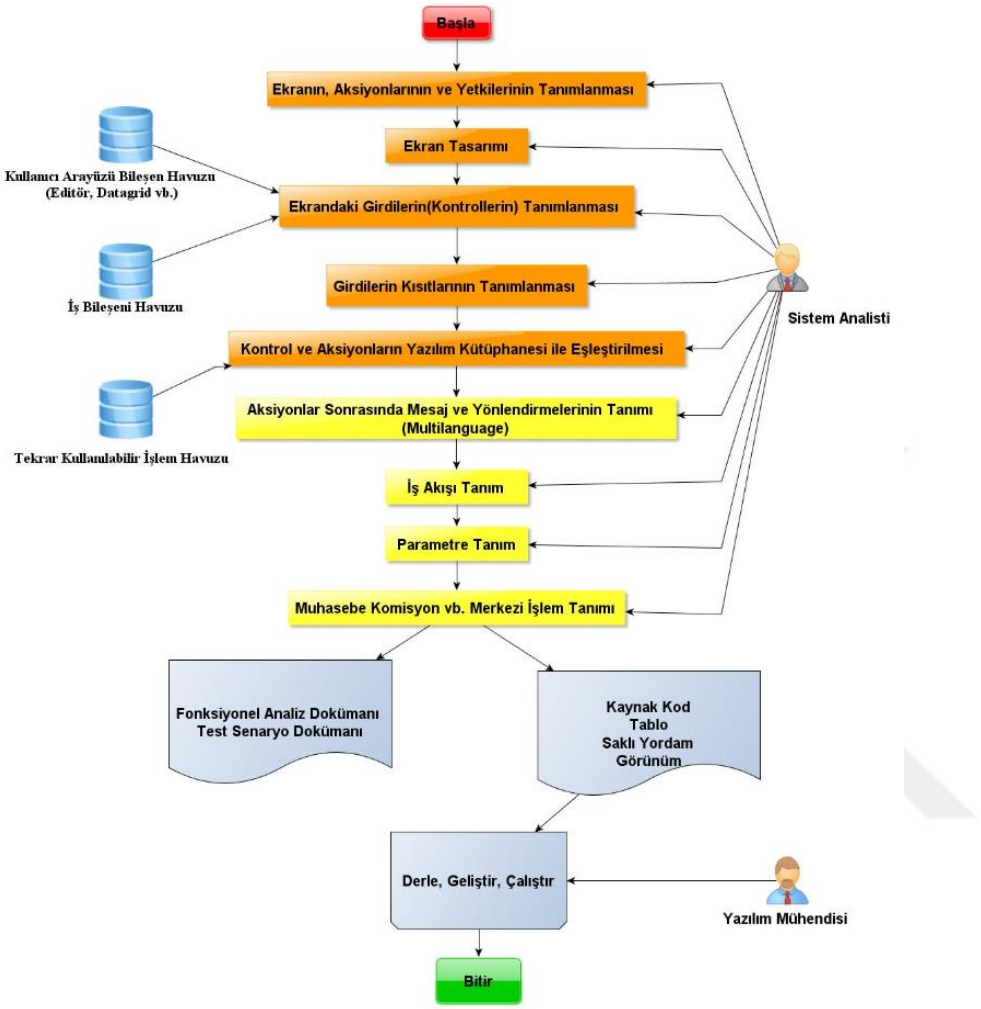
2.2. İş Gereksinimi Odaklı Kaynak Kod Üretimi

Balçiçek ve Altıparmak(2014) ile çalıştığımız bu modelde iş gereksinimlerini ekran tanımı, tasarımı, ekrandan alınacak veri tipleri, iş akışı, parametre tanımları ve merkezi sistem entegrasyon tanımlarının yapılabileceği tanım tabanlı bir araç aracılığı ile alınabileceği ifade edilmiştir. Önerilen sistem, iş gereksinimleri tanımları referans alınarak, gerekli uygulama sunucusu ve kullanıcı arayüzü kaynak kodları, veritabanı nesneleri, fonksiyonel analiz dokümanı ve test senaryo dokümanı üretebilmektedir.
Modelin gerçekleştirilebilmesi için yazılım üretim bandının olması, dolayısı ile yazılım geliştirme standartlarının ve tasarım kalıplarının oluşturulmuş ve oturmuş olması oldukça önemlidir. Böylelikle daha efektif, kontrolü ve yönetilebilirliği kolay kod parçalarının kod üretici tarafından üretilebilmesi mümkün olmaktadır.

Sistem analistinin iş gereksinimlerini toparlayarak, var olan sistemi analiz ederek ve yeni modülü kurgulayarak bu bilgileri sisteme girdi olarak uygulama aracılığı ile tanıtabilmelidir. Analistler bu araç ile ekran tasarımı, ekrandan alınacak veri bilgileri ve kullanıcı ara yüz kontrolleri, aksiyonlar, aksiyon sonrasındaki yönlendirme ve mesajlar, muhasebe ve komisyon gibi merkezi işlemler ve son olarak da kullanıcılar arası onay ve işlem akışını içeren iş akışı gibi bütün tanımlamaları yapabilmelidir. Bu tanımlamaların yapılmasından sonra sistemden çıktı olarak; veritabanı tablosu, veritabanı görünümü, saklı yordamlar, orta katman uygulama kodları, kullanıcı arayüzü kodları, fonksiyonel analiz ve fonksiyonel test dokümanları verilebilmelidir.

İş gereksinimlerinin kodsal karşılığını adresleyebilmek, daha önce çözümlenmiş gereksinim kodlarını tekrar kullanabilmek adına yazılım nesne ve fonksiyonların yer aldığı bir yazılım kütüphanesi oluşturulması modelin başarımını artıracaktır. Sistem analisti tasarım anında kullanıcı bileşenleri ve aksiyonlar ile bu kütüphanedeki

fonksiyon ve metotları eşleştirmek sureti ile yazılım mühendisine ilgili fonksiyonları hazır ve bağlı halde sunabilecektir.

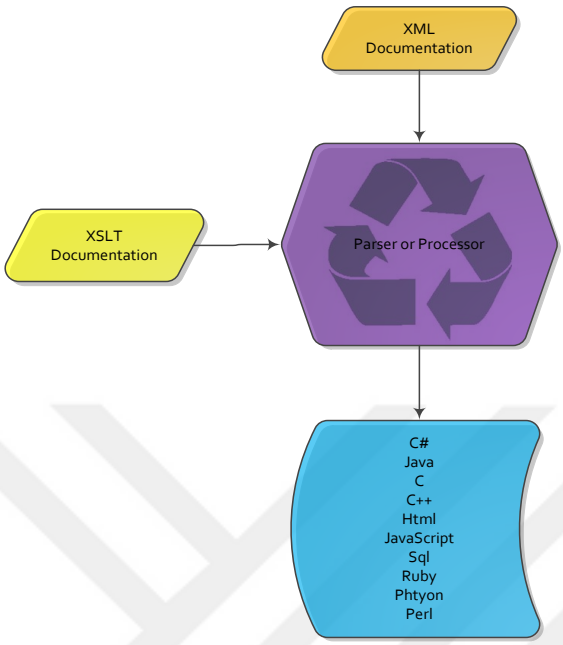


Şekil 2.2. Kod Üretim Modeli Süreci

Uygulamanın, fiziksel ve sanal yazılım katmanları için üretmiş olduğu kodlar kurum içerisinde kullanılan uygulama geliştirme ortamı ve uygulama geliştirme standartları dahilinde üretilmiş olarak, derlenebilir bir halde yazılım mühendisine aktarılır. Yazılım mühendisi bu kod ve dokümanları temel alarak otomatize edilemeyen talepleri karşılamak amacı ile kendisine aktarılan özelleştirmelerini yaparak geliştirme işlemini tamamlar. Tüm süreç Şekil 2.2. ‘de gösterilmektedir.

Balçiçek ve Altıparmak(2013) ile anlattığımız Kaynak Kod Üretimi modelinde kullanılan Xslt tekniği İGOT modeli için de uygulanabilir.

Şekil 2.3.'de ifade edildiği gibi Xslt herhangi bir Xml içeriğini farklı bir Xml, Html, Csv (Comma Seperated Values) veya text formatına dönüştürme işlemi ile ilgili materyalleri sağlayan bir işaretleme dilidir(Balçiçek,Altıparmak, Tokgöz,2013).



Şekil 2.3. XML'den XSLT 'ye dönüşüm şeması

Kullanıcıdan alınan verilere göre xslt teknolojili şablonlar ile gerekli kodlar oluşturulmaktadır. En önemli husus ortadaki parser ve XSLTransform işlemidir. Aşağıdaki Şekil 2.4.'de xslt uzantılı şablon dosyasının içeriği verilmiştir(Balçiçek, Altıparmak, Tokgöz,2013).

```
<?xml version="1.0" encoding="utf-8"?>
<!-- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"></xsl:output>
  <xsl:template match="/Tablo"> -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-prefixes="msxsl">
    <xsl:output method="text" indent="yes"/>

  <xsl:include href="GeneratorInformation.xslt"/>
  <xsl:include href="BusinessObjectForTable.xslt"/>

    <xsl:template match="/Table">

  <xsl:variable name="FileName"><xsl:value-of select="@ClassName"/>.cs</xsl:variable>

<xsl:call-template name="GeneratorInformation">
  <xsl:with-param name="FileName" select="$FileName"></xsl:with-param>
  <xsl:with-param name="SqlOrCs" select="'Cs'"></xsl:with-param>
</xsl:call-template>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using BOA.Common.Types;
using BOA.Base;
using BOA.Base.Data;
using BOA.Types.<xsl:value-of select="@ProjectName"/>;

namespace BOA.Business.<xsl:value-of select="@ProjectName"/>
{
  /// <summary>
    /// Auto generated Business Object class of
    "<xsl:value-of select="@Schema"/>.<xsl:value-of select="@Name"/>" table.
    /// </summary>
  <xsl:call-template name="BusinessObjectClassForTable">
  </xsl:call-template>
}
  </xsl:template>
</xsl:stylesheet>
```

Şekil 2.4. Xslt uzantılı şablon dosyasının içeriği

Anlattığımız bu modelde test senaryolarının oluşturulması, çalıştırılması çözümlerinin kodsal olarak üretimi test otomasyonu ile entegrasyonu bulunmamaktadır. İGOT modelinde iş gereksinimleri odaklı test senaryoları temel alınarak yazılımın üretilmesi söz konusudur.

2.3. Gereksinimlere Dayalı Otomatik Test Durumu Üretimi

Kara kutu testi, test durumlarının uygulamanın iç yapısına bakılmaksızın gereksinimlerden türetildiği bir tekniktir. Mevcut uygulamada kara kutu test durumları gerekliliklerden elle türetilmiştir. Gereksinimlerden elle test durumları çıkartmak maliyetli ve zaman alan bir işlemdir. Rajan(2006) bu modelde, kara kutu test durumlarının gereksinimlerden otomatik olarak oluştuğu ve dramatik zamana ve maliyet tasarrufuna neden olabileceği fikrini sunmaktadır. Bunu başarmak için zamansal lojik özellikler olarak biçimlendirilen şartları kullanıyor. Kapsama ölçümlerini doğrudan biçimlendirilmiş gereksinimlerin yapısı üzerinde tanımlıyor ve istenen kriterleri karşılayan resmi gerekliliklerden test durumları üretmek için model denetleyicisi gibi otomatikleştirilmiş bir test kutusu oluşturma aracı kullanıyor. Bu şekilde üretilen kara

kutu test takımlarının etkinliğini değerlendirmek için, test takımları tarafından gerçekleştirilen uygulama kapsamını ve bunların arıza bulma etkinliklerini ölçmekte(Rajan,2006).

Bu modelde İGOT modelinden farklı olan bir çok yanı bulunmaktadır. Gömülü sistemlerde uygulanan bu model birleşik donanımsal aygıtlarda çalışan yazılımların arızaları tespit edebilmekte ve sadece giriş değerleri ile çıkış değerleri ölçüm olarak alınmaktadır. Yazılım yaşam döngüsünün sadece test aşamasında test durumu üretmektedir.

2.4. Otomatik Test Durumlarının Üretimi

Otomatik olmayan test zor ve zaman alıcıdır ve belki de büyük sistemlerde imkansız veya testte hatalar oluşturur. Yazılım testi kalkınma faaliyetlerinin artan maliyetidir. Yazılım ve test durumunun oluşturulması önemli bir faaliyettir. Dolayısıyla otomatikleştirmek için yapılan araştırmalar otomatik test kutusu üretimi gibi yapılar üzeredir(Motlagh,2012).

Motlagh(2012) yaptığı bu incelemede test durumu oluşturmak için yapılan son araştırmalar üzerine bir araştırma yayınladı. UML tabanlı, biçimsel yöntemler, web uygulaması, web servisi, kombin ve grafiklerden sunulan verilerle test durumlarının oluşturulmasını incelemiştir.

Bu yöntemler İGOT modelinde kod üretimi aşaması, veritabanı öğelerinin oluşturulması işlemi, web gereksinimlerinin oluşturulması gibi işlemlerin sonrasında kullanılabilmektedir.

2.5. Test Otomasyonu

Yazılım test etmede, test otomasyonu önceden tahmin edilmiş sonuçlarla gerçek sonuçların karşılaştırılması ve testlerin koşulmasını kontrol etmek için(test edilmiş yazılımdan farklı olan) belirli yazılımın kullanılmasıdır. Test otomasyonu tekrar eden fakat çoktan test etme süreçlerinde yer almış gerekli testlerin otomatikleştirebilir veya manuel olarak koşulmasının zor olacağı testleri de içerebilir. Test otomasyonları sürekli paket dağıtımı veya sürekli test etme için kritik öneme sahiptir(Anonim,2011). Kısaca elle(manuel) yapılan yazılım testlerinin, komut dizisi(script) veya bir araç(tool) aracılığıyla otomatik olarak yapılması olarak tanımlanır.

Test otomasyonu test ile ilgili olsa da, aslında bir yazılım geliştirme işidir. Yazılım geliştirmenin erken safhalarında otomasyon çalışmalarına başlamak gerekir. Başarılı test otomasyonu kararlılık, test uzmanları ve geliştiriciler arasında takım çalışması gerektirir. Yazılım otomasyonu da bir yazılım geliştirme projesi olarak ele alınmalı, bu iş için de özel olarak atanmış test uzmanları bulunmalı, otomasyon fazının da normalde işleyen yazılım projelerinde olduğu gibi gereksinimlerinin çıkarılması, tasarımının yapılması, hata yönetimi yapılması ve testlerinin yapılması gerekmektedir(Yıldız,2014).

2.5.1. Otomasyonun Avantajları

• Bir organizasyondaki Test Otomasyonu prosedürleri manuel testlerdeki insana bağlı uygulamayı azaltarak, sistem testlerinin daha kaliteli olmasını sağlar.

• Daha fazla hatayı daha erken teşhis ederek yazılım test sürecinde etkinlik ve verimliliğinin artırılması sağlar.

• Sürekli tekrarlanan testlerin otomatize edilmesi test maliyetini azaltır.

• Test mühendisi otomasyon sayesinde testlerini daha detaylı yapmak için ekstra vakit kazanır (Keşif Testi – Exploratory Testing ve Kullanılabilirlik Testleri – Usability Testing).

• Altyapısal değişiklikte Regresyon Testinin (Regression Testing) hızlı bir şekilde tamamlanmasında önemli bir rol oynar.

• Testlerin yeniden kullanımını kolaylaştırır.

• Testlerle kapsanan kod yüzdesini artırır (Code Coverage).

• Testler 7×24 çalışabilir.

• Testlerin raporlama kalitesinin arttırılmasını sağlar.

• Geliştirilen ürünün kalitesini arttırır(Yıldız,2014).

2.5.2. Manuel ve Otomasyon İncelemesi

Testlerin manuel ve otomasyon(Automated) ile yapılması ile ortaya çıkan durumlar aşağıdaki Çizelge 2.1.'de verilmiştir.

Çizelge 2.1. Manuel ve otomasyon incelemesi(Yıldız,2014)

Manual	Automated
Gerçek Tecrübe	Kesin
Daha Az Başlangıç Maliyeti	Hızlı
Detaylı	Verimli, Etkin
	Devamlı

2.5.3. Otomasyonun Kullanım Alanları

Test etme araçları ürün kurulumu, test veri oluşturmasını, GUI etkileşimini, problem tespit etme gibi A dan Z ye tüm testlerin otomatikleştirilmesini gerektirmeyen durumlarda bile otomatikleştirmeye yardımcı olabilir.

Test otomasyon düşünüldüğünde aşağıdaki popüler gereksinimlerin yerine getirilmesi gerekir(Anonim,2011):

- Platform ve işletim sistemi bağımsızlığı
- Veri sürdürülebilirlik yeteneği(girdi verisi, çıktı verisi, Metadata)
- İyileştirilebilir raporlama(DB Access very tabanı, Cyristal Reports)
- Kolay debug ve loglama.
- Kullanıcı dostu versiyon kontrolü- minimal binary dosyalar.
- Genişletilebilir ve uyarlanabilir(Diğer araçlarla entegre olabilen açık kaynak API ler.)
- Ortak sürücü(mesala java geliştirme ekosistemi, ANT veya Maven ve popular geliştirme ortamları.) Bu testlerin geliştirici iş akışlarıyla entegre edilmesini mümkün kılar.

Otomasyon işleri kolaylaştırmak için yapılır. Bazen en önemli hatalar Ad Hoc testler ile bulunur, yani testçi kendisini müşterinin yerine koyar ve çeşitli senaryolar dener. Böylece önemli hataları tespit eder. Ancak bilinmelidir ki, otomasyon manuel testlerin yerine konan bir yöntem değildir. Çizelge 2.2.'de testlerin otomatize edilebilirliği belirtilmiştir(Yıldız,2014).

Çizelge 2.2. Testlerin Otomatize Edilebilirliği

	Testler Otomatize Edilebilir Mi ?			
Test Çalıştırma Periyodu		Mümkün Değil	Pahalıya Mal Olur	Ucuza Mal Olur
	Ne Zaman Mümkünse	X	?	+
	Devamlı	X	?	+
	Ara Sıra	X	X	X

2.5.4. Test Otomasyon Tipleri

Otomasyon geliştirilen yazılımın içerdiği nesneleri yada simgeleri kullanarak test işlemini gerçekleştirebilmektedir. Çizelge 2.3.'de test otomasyonu için nesne ve simge bazlı tanımlamaların özellikleri incelenmiştir.

Çizelge 2.3. Test Otomasyon Tipleri(Yıldız,2014)

Obje Bazlı Tanımlama	
Avantajları	***Dezavantajları***
Çok Esnek Kontroller	Platforma Bağlı
Derin (detaylı) Otomasyon Şansları	Birleşik Kontrol Engeli
Olgun Uygulamalar İçin Hassas Olmaması	Düşük Kaliteli Kod İçin Bakım Sorunları

İmaj Bazlı Tanımlama	
Avantajları	***Dezavantajları***
Ekranda Görünen Her Şeyi Otomatize Edebilir	Non-GUI Yazılımları Otomatize Edemez
Platforma Bağlı Değil	Ekran Özelliklerine Gereğinden Fazla Duyarlı
Kullanılması Kolay	Obje Bazlı Tanımlamaya Göre Daha Az Esnek

2.5.5. Test Otomasyon Araçları

Test otomasyonunda en önemli başarı faktörlerinden biri doğru test aracını seçmektir, test yapılan sisteme en uygun test otomasyon aracı kullanılmalıdır. Otomasyon yazılırken bakım ve güncelleme maliyeti göz önüne alınmalı ve otomasyonlar belli bir standarda uygun oluşturulmalıdır. Yazılım değişiklikleri ve otomasyon ilişkisi sürekli takip edilerek, otomasyonlar güncel tutulmalıdır. Tüm manuel testlerin otomasyona geçirilmesi mümkün olmayabilmektedir. Çok sayıda ticari ve ücretsiz açık kaynak kodlu aracın olduğu test otomasyon pazarında son birkaç senede en çok kullanılan araçlardan birkaç tanesi UFT (HP), RFT (IBM) ve Selenium'dur. Bu araçların avantajları ve dezavantajları Çizelge 2.4.'de belirtilmiştir(Yıldız,2014).

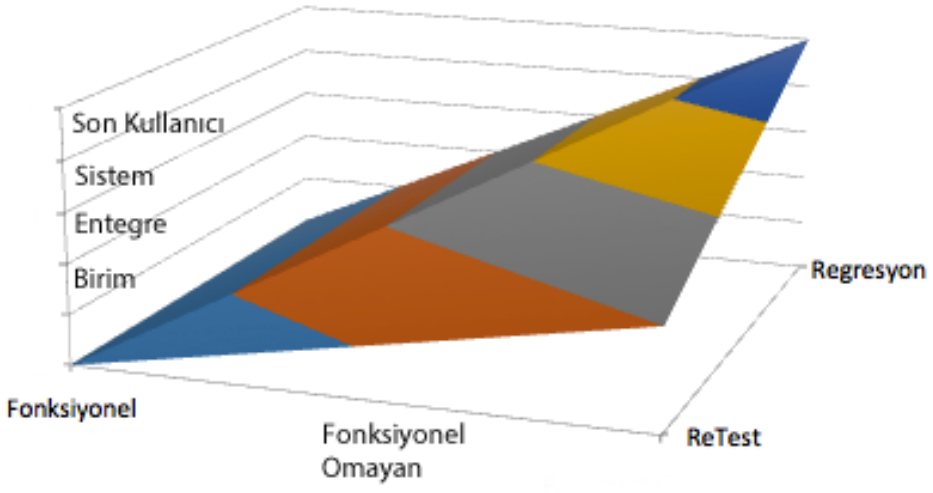
Çizelge 2.4. Test Otomasyon Araçları

Tools	Avantajları	Dezavantajları
HP/Unified Functional Testing (UFT)	• Web 2.0, Java ve .NET uygulamalarını destekler • Full GUI Object Map ambarı • Quality Center/ALM sorunsuz entegrasyon	• Visual Basic scripting sınırlı • Lisanslı ürün
IBM/Rational Functional Tester (RFT)	• Eclipse plug-in olarak built edililir ve full IDE ve Java desteği vardır • Web 2.0, Java ve .NET uygulamalarını destekler • Full GUI Object Map ambarı	• Tarayıcı desteği yetersiz • Lisanslı ürün
Selenium RC & IDE	• İyi tarayıcı desteği • İyi programlama desteği (Java, Ruby, C#) • Kolaylıkla jUnit suit'ine dönüştürülebilir • Açık-kaynaklı	• GUI Object ambarı yok • Sadece web uygulama desteği

İGOT modelinde test otomasyonu ile birleştirilmek istenilirse belirtilen bu özellikler dikkate alınarak seçim yapılmalıdır.

2.6. Test Seviyeleri

Yazılım yaşam döngüsünün içinde yazılım ürününün küçük biriminden büyük sisteme kadar olan yapıların test edilmesinde seviyeler yer alır. Şekil 2.5.'de bu seviyeler belirtilmiştir.



Şekil 2.5. Test Seviyeleri ve Test Çeşitleri

Test seviyelerinin her biri için şunlar belirlenebilir: genel hedefler, test senaryoları türetmek için referans alınan çalışma ürünleri(örn. test esası), test nesnesi (örn. test edilen şey), bulunacak genel hatalar ve arızalar, test kuluçkası gereksinimleri, araç desteği, özel yaklaşımlar ve sorumluluklar(ISTQB,2011).

2.6.1. Bileşen (Birim) Testi

Birim test uygulamanın kaynak kodunun belli bölümlerinin doğru biçimde çalıştığını ve kullanıma uygun olup olmadığını anlamak için kullanılan bir test tekniğidir. Birim bir yazılımın test edilebilir en küçük parçasıdır. Birim yapısal programlamada bir fonksiyona, nesneye yönelik programlamada ise sınıfa karşılık gelebilir(Tozlu,2010).

Bileşen testi; fonksiyonalite testini, fonksiyonel olmayan testleri, yapısal testleri veya sağlamlık testlerinden oluşabilir. Test senaryoları, bileşenin gereksinimi, yazılım tasarımı veya veri modeli gibi test esaslarından türetilebilir. Bileşen testinde genellikle test edilen koda erişim söz konusudur. Aynı zamanda, birim testi çerçevesi ya da hata ayıklama aracı gibi bir geliştirme ortamının desteğiyle gerçekleştirilir. Uygulamada, bileşen testi genellikle kodu yazan programcı tarafından yapılır. Bileşen testlerinde çoğunlukla hatalar bulundukları anda kayıt altına alınmadan düzeltilir(ISTQB,2011).

Bileşen testine yönelik yaklaşımlardan biri, kodlamadan önce test senaryolarını hazırlamak ve otomasyona geçirmektir. Bu yaklaşıma test öncelikli yaklaşım veya test güdümlü geliştirme adı verilir ve oldukça döngüseldir. Test senaryolarını geliştirme, ardından küçük kod parçalarını oluşturma ve entegre etme ardından da tüm sorunları düzelterek ve başarılı olana kadar yineleyerek bileşen testlerini yürütme döngüsüne dayanır(ISTQB,2011). Tam da bu yaklaşıma dayanarak İGOT modelinde yazılım bileşenlerinin tanımlanmasıyla senaryolar sistem tarafından oluşturularak yapılabilmesi durumunda doğru kodu oluşturmalı yada yapılamadığı durumlarda senaryoların içerildiği test projeleri oluşturulmalıdır. Her iki durumunda gerçekleştirilememesi durumunda da dokümante edilerek manuel test için raporlanması İGOT modeli olarak tasarlanmıştır.

2.6.2. Entegrasyon Testi

IEEE'nin Standart Yazılım Mühendisliği Sözlüğü'ne göre entegrasyon testi, yazılım bileşenlerinin, donanım bileşenlerinin veya her ikisinin bir bütün halinde ele alınarak aralarındaki etkileşimlerin test edilmesidir(IEEE,1990).

Birden fazla entegrasyon testi seviyesi olabilir ve aşağıdaki gibi çeşitli boyutlardaki test nesneleri üzerinde gerçekleştirilebilir(ISTQB,2011):

1. Bileşen entegrasyon testi, yazılımın bileşenleri arasındaki etkileşimleri test eder ve bileşen testinden sonra yapılır.

2. Sistem entegrasyon testi, farklı sistemler veya donanım ve yazılım arasındaki etkileşimleri test eder ve sistem testinden sonra yapılabilir.

Sistematik entegrasyon stratejileri, sistem mimarisine (yukarıdan aşağıya veya aşağıdan yukarıya gibi), fonksiyonel görevlere, işlemleri ele alma sıralarına veya sistem ya da bileşenlerle ilgili diğer bazı kapsamlara dayanabilir. Kusur izolasyonunu kolaylaştırmak ve hataları erken saptamak için entegrasyonun "büyük patlama" yerine aşamalı olması gerekir(ISTQB,2011).

Entegrasyon testi birbirleri ile bağlı bir şekilde çalışan bir kaç modülün bir araya gelerek çıkardıkları sonuçla ilgilenirken, birim testleri ise sadece birim başı testlerle ilgilenir. Eğer Entegrasyon testine yönlenirse Test Güdümlü Yazılım geliştirme yaparken, metot bazlı doğruluk sağlanmamış olur(Gökalp,2015).

Her bir entegrasyon aşamasında test uzmanları yalnızca birleşime odaklanır. Örneğin, modül X'i modül Y ile birleştirirken, her bir modülün fonksiyonalitesini değil sadece bu modüller arasındaki iletişimi test etmelidirler. İGOT modelinde bu modüllere ait özellikler kullanılarak test senaryosu üretilmesi gerekmektedir.

İdeal yaklaşım, mimari özelliklerin de dahil edilerek entegrasyon testi kurgusunun planlanmasıdır. İGOT modelinde modüllerin ve mimari özelliklerin aynı anda ele alınarak senaryoların oluşturulması önerilmektedir.

2.6.3. Sistem Testi

Sistem testi bir yazılımın (donanımlar için de bu test uygulanabilir) gereksinimlerinin tam ve doğru bir biçimde karşılanıp karşılanmadığını belirlemek için yapılır. Bütünleştirme testinin ardından yapılan bu test, bütünleştirme testi sonrası ortaya çıkan bütünleşik sistemi girdi olarak alır ve testlerini bunun üzerinde uygular.

Sonuçta ortaya gereksinimleri yazılım gereksinimleri tanım belgesi ile tanımlanmış ve burada tanımlı bütün gereksinimleri eksiksiz ve doğru biçimde gerçekleştirmiş bir sistem çıkar(Tozlu,2010).

Sistem testi, risklere ve/veya gereksinimlere, iş süreçlerine, kullanım senaryolarına veya sistem davranışını tanımlayan metinlere ya da modellere, işletim sistemiyle etkileşimlere ve sistem kaynaklarına dayanabilir. Sistemin fonksiyonel ve fonksiyonel olmayan gereksinimlerini ve veri kalitesini sorgulamalıdır. Bu seviyede test uzmanlarının tamamlanmamış gereksinimlerle de ilgilenmesi gerekir. Fonksiyonel gereksinimlerle ilgili sistem testi, test edilecek sistem için en uygun gereksinim bazlı (kara kutu) teknikler kullanılarak başlar. Örneğin, iş kurallarında tanımlanan etkilerin kombinasyonları için karar tablosu test tekniği kullanılabilir. Ardından, yapısal testlere (beyaz kutu) geçilerek gereksinim bazlı testlerin yakalayamadığı hatalar yakalanabilir. Örneğin menü yapısı ve web sayfası navigasyonunu test nesnesi olarak ele alan yapısal testler(ISTQB,2011).

İGOT modelince sistem testinde yapay zeka teknikleri kullanılarak sistemin öğrendikleri ile test senaryoları oluşturması tavsiye edilir. Mümkün olmadığı durumlarda sadece temel fonksiyonaliteye bağlı işlemlerin gerçekleşmesi yada fonksiyonel olmayan performans, yük testi gibi testlerin senaryolarının oluşturulması sağlanabilmelidir.

2.6.4. Kabul Testi

Kabul testi genellikle bir yazılımın müşterilerinin veya kullanıcılarının sorumluluğundadır; bu seviyedeki testlere diğer paydaşlar da dahil olabilir. Kabul testinin amacı, sisteme, sistemin parçalarına veya sistemin fonksiyonel olmayan gereksinimlerine karşı güven oluşturmaktır. Kabul testinde ana odak hataları bulmak değildir, sistemin canlıya hazır olduğunu göstermektir. Kabul testinin son test seviyesi olmadığı durumlar olabilir buna rağmen kabul testinde sistemin canlıya alınmaya ve kullanıma hazır olup olmadığı denetlenebilir(ISTQB,2011).

İGOT modelinde kabul testleri için dokümantasyon olarak test senaryoları çıktısı ve test raporları verilmelidir.

2.7. Test Çeşitleri

Özel bir nedene veya test hedefine dayanarak yazılımın (veya yazılımın bir bölümünün) testi yapılabilir.

Test çeşidi belirli bir test hedefine odaklanır, bu hedef aşağıdakilerden herhangi biri olabilir:

- Yazılımın gerçekleştireceği bir fonksiyon
- Güvenilirlik veya kullanılabilirlik gibi fonksiyonel olmayan gereksinimler
- Yazılımın veya sistemin yapısı ya da mimarisi
- Değişiklik ile ilgili. Örn. hataların düzeltildiğini onaylama (onaylama testi) ve istenmeyen değişiklikleri arama (regresyon)

Yapısal test (örn. kontrol akışı modeli veya menü yapısı modeli), fonksiyonel olmayan test (örn. performans modeli, kullanılabilirlik modeli, güvenlik tehdidi modellemesi) ve fonksiyonel test için (örn. süreç akış modeli, durum geçişi modeli) yazılıma benzeyen bir model geliştirilebilir ve/veya kullanılabilir(ISTQB,2011).

2.7.1 Fonksiyonu Test Etme (Fonksiyonel Test)

Fonksiyonel test, donanım, yazılım, harici veya dahili uygulamanın gerekli işlevlerini nasıl yürüttüğünü inceler ve yayınlanan yazılımın kalitesini garanti altına almak için hayati öneme sahiptir. Kullanıcı komutlarını, veri işlemeyi, aramaları, iş süreçlerini, kullanıcı ekranlarını vb test etmeyi içerir(Belatrix, 2015).

Bu, doğru ve yanlış girilen verilerin geniş bir yelpazesini kullanarak, ürün davranışını, özelliklerine göre doğrulayan bir dizi test halinde yürütülür. Bu, ürünün kullanıcı arabiriminin, veritabanının, güvenlik, kurulum, ağın vb. test edilmesini içerebilir.

Kullanıcı arabirimi düzeyinde işlevsel sınamanın yapılması çok önemlidir, çünkü bir kaynak kodu incelemesi yapılırken hemen görünmeyen birçok eksikliği ortaya çıkarabilir. Birincil öncelik, uygulamanın dahili çalışmalarının karmaşıklığı yerine, uygulamanın kullanılabilirliğini test etmektir(Belatrix, 2015).

Yazılımın fonksiyonalitesinden gereksinim bazlı test teknikleri kullanılarak test koşulları ve test senaryoları türetilebilir. Fonksiyonel testlerde genellikle yazılımın harici davranışları, girdi ve çıktılar dikkate alınır (kara kutu testi). Bir çeşit fonksiyonel test olan güvenlik testi, kötü amaçlı dış kaynaklardan gelen tehditlerin algılanması ile

ilgili fonksiyonları ele alır (örn. güvenlik duvarı). Fonksiyonel testin diğer bir çeşidi olan birlikte çalışabilirlik testi, yazılımın belirtilen bir veya daha fazla bileşenle veya sistemle etkileşim kurma yeteneğini değerlendirir(ISTQB,2011).

2.7.2 Fonksiyonel Olmayan Gereksinimleri Test Etme (Fonksiyonel Olmayan Test)

Fonksiyonel olmayan testler, işlevsel olmayan gerekliliklerle ilgilidir ve özellikle, bir sistemin fonksiyonel test kapsamına girmeyen çeşitli ölçütlere göre hazır olup olmadığını değerlendirmek üzere tasarlanmıştır. Örneğin, fonksiyonel testte, bir veri tabanındaki bir hücre kümesine veri girme işlevi çalıştığını ancak kullanılabilirlik testinin (İşlevsizlik Denetimi'nin bir parçası) işe yaradığı belgenin bir sürümünün kaydedilmesinin 2 dakika, (Ya da beklemek için çok uzun sürecek) olarak değerlendirilecektir(Eriksson,2015).

Esasen fonksiyonel olmayan test, yazılım sistemlerinin işlevsel olmayan özelliklerinin sonuçlarının test edilmesidir. Örneğin, uygulamayı veya sistemi müşterinin gereksinimine veya bir performans gereksinimine karşı test ederek ölçüp karşılaştırmamızı sağlar. Temel olarak, işlevsel olmayan test, ürünün ne yaptığı yerine, ürünün ne kadar iyi davrandığını gösterir .

Fonksiyonel olmayan test, tüm test seviyelerinde gerçekleştirilebilir. Fonksiyonel olmayan test terimi, yazılımın performans testindeki yanıt süreleri gibi değişen bir skalada ölçülebilen karakteristiklerini ölçmek için gereken testleri tanımlar. Bu testler, "Yazılım Mühendisliği – Yazılım Kalitesi" (ISO 9126) kapsamında tanımlanan model gibi bir kalite modelini referans alabilir(ISTQB,2011).

2.7.3 Yazılım Yapısını/Mimarisini Test Etme (Yapısal Testler)

Yapısal (beyaz kutu) testler, tüm test seviyelerinde gerçekleştirilebilir. Yapısal tekniklerin test kapsamını tamamlamaya yardımcı olması amacıyla gereksinim bazlı tekniklerden sonra kullanılması tavsiye edilir.

Test kapsamı, yapının bir test grubu tarafından çalıştırılma derecesidir ve kapsanan öğelerin yüzdesi olarak ifade edilir. Kapsam %100 değilse, kapsamı artırmak amacıyla test edilmeyen öğeleri test etmek için daha fazla test tasarlanabilir(ISTQB,2011).

Bileşen testi ve bileşen entegrasyon testi başta olmak üzere tüm test seviyelerinde, komut ve karar öğelerinin kod kapsamını ölçmek için araçlar kullanılabilir. Yapısal testler, bir çağırma hiyerarşisi gibi sistem mimarisine dayanabilir. Yapısal test yaklaşımları ayrıca sistem, sistem entegrasyonu veya kabul testi seviyelerinde uygulanabilir (örn. Menü yapıları) (ISTQB,2011).

2.7.4 Değişiklikleri Test Etme: Tekrar Testi ve Regresyon

Bir hata tespit edildikten ve düzeltildikten sonra bulunan hatanın başarılı şekilde ortadan kaldırıldığını onaylamak için yazılım yeniden test edilmelidir. Bu işleme onay adı verilir. Hata ayıklama (bir hatayı bulma ve düzeltme) bir geliştirme işlemidir, test işlemi değildir(ISTQB,2011).

Regresyon testi, yeni bir program veya kod değişikliğinin mevcut özellikleri olumsuz etkilemediğini doğrulamak için bir yazılım testi türü olarak tanımlanır. Regresyon testi, halihazırda işlevselliklerin iyi çalıştığından emin olmak için yürütülmüş olan test durumlarının tam veya kısmen seçilmesinden başka bir şey değildir. Bu test, yeni kod değişikliklerinin mevcut işlevler üzerinde yan etkilere sahip olmamasından emin olmak için yapılır. Yeni kod değişiklikleri yapıldıktan sonra eski kodun hala çalışmasını sağlar(Anonim,2015).

Tekrar test etme, kodun sabitlendiğinden emin olmak için işlevsellik veya hata testini tekrar denemek demektir. Sabit değilse, hata bildiriminin yeniden açılması gerekiyor. Düzeltilirse, bildirim kapanır.

Regresyon ve tekrar test, tüm test seviyelerinde gerçekleştirilebilir ve fonksiyonel, fonksiyonel olmayan ve yapısal testleri içerir. Regresyon test grupları birçok kez çalıştırılır ve genellikle yavaş bir şekilde ilerler; bu nedenle regresyon, otomasyon için güçlü bir adaydır(ISTQB,2011).

2.8. Bakım Testi

Bir yazılım sistemi piyasa sürüldükten sonra yıllarca kullanılır. Bu süre içerisinde sistem ve çevresi düzenlenebilir ya da değişebilir. Bakım testi, var olan bir işletim sistemi üzerinde, yapılan değişikliklerden (modification) sonra ya da taşıma/göç (migration) işleminden sonra yapılır. Değişiklikler, sürüm tabanla (release-based) olarak

yapılan değişiklikler ya da acil bir biçimde yapılacak olan değişiklikler olabilir(Önder,2013).

Yazılımın dağıtılması ve kullanıma başlanmasından sonra yazılımda yapılacak değişiklikler yazılımın bakımı olarak adlandırılır. Bu değişiklikler basit kodlama hatalarının düzeltilmesi şeklinde olabileceği gibi tasarımdan kaynaklanan hataların giderilmesi gibi daha kapsamlı değişiklikler şeklinde de olabilir. Yazılımın bakımı aslında yazılımın evrimleşmesidir. Yazılımın yaşamına devam edebilmesi için gerekli değişikliklerin uygulanmasıdır(Önder, 2013).

Taşıma (bir platformdan diğerine) için bakım testi, değiştirilen yazılımda olduğu kadar yeni ortamda da gerçekleştirilmesi gereken operasyonel testleri içermelidir. Taşıma testi (dönüşüm testi), başka bir uygulamadaki veriler bakımı yapılan sisteme taşınacağı zaman da gereklidir(ISTQB,2011).

Bir sistemin kullanımdan kaldırılmasına yönelik bakım testi, veri taşıma testini ve uzun süreli veri saklama dönemleri gerekmesi durumunda arşivlenebilir. Değişikliklerle ilgili testlere ek olarak bakım testi, değiştirilmeyen sistem bölümlerinde gerçekleştirilen regresyonu da kapsar.

Bakım testinin kapsamı, değişiklik riskine, var olan sistemin boyutuna ve değişikliğin boyutuna göre değişir. Değişikliklere bağlı olarak bakım testi, tüm test seviyelerinde ve tüm test çeşitleri için gerçekleştirilebilir. Var olan sistemin değişikliklerden ne şekilde etkileneceğini belirlemeye etki analizi adı verilir ve bu analiz ne kadar regresyonun yapılacağına karar vermede kullanılır. Etki analizi, regresyon test grubunu belirlemek için kullanılabilir(ISTQB,2011).

2.9. Statik Teknikler

Yazılımın yürütülmesini gerektiren dinamik testin aksine statik test teknikleri, kod yürütülmeden kodun veya diğer proje dokümanlarının manuel olarak incelenmesine (gözden geçirme) ve otomatik şekilde analiz edilmesine (statik analiz) dayanır(ISTQB,2011).

Gözden geçirme, yazılımı (kod dahil) test etmenin bir yöntemidir ve dinamik testler yapılmadan önce gerçekleştirilebilir. Yazılım geliştirme yaşam döngüsünün ilk adımlarında gözden geçirmeler sırasında tespit edilen hataları düzeltmek (örn. Gereksinimlerde veya analizde bulunan hatalar) genellikle yürütülen kod üzerinde çalıştırılan testlerle tespit edilen hataları düzeltmekten daha düşük maliyetlidir.

Gözden geçirme, tamamen otomatikleştirmeden yapılan bir işlem olarak uygulanabilir, ayrıca araç desteği de bulunur. Asıl yapılan işlem, bir dokümanı incelemek ve bu doküman hakkında yorumlar yapmaktır. Gereksinimler, tasarımlar, kod, test planları, test gereksinimleri, test senaryoları, test komut dosyaları, kullanım kılavuzları veya web sayfaları gibi tüm yazılım ürünleri gözden geçirilebilir.

Statik testlerde, dinamik testlere göre daha kolay bulunan genel hatalar şunlardır: standartlardan uyumsuzluk, gereksinim hataları, tasarım hataları, eksik sürdürülebilirlik ve yanlış ara yüz gereksinimleri.

2.10. Test Tasarım Teknikleri

Test tasarımı sırasında test senaryoları ve test verisi oluşturulur ve belirlenir. Test senaryosu, belirli test hedeflerini veya test koşullarını kapsayacak şekilde belirlenmiş bir dizi girdi değeri, yürütme önkoşulu, beklenen sonuç ve yürütme artkoşulu içerir. "Yazılım Testi Dokümantasyonu Standardı" (IEEE STD 829-1998), test tasarımlarını (test koşullarını içeren) ve test senaryolarının içeriğini tanımlar. Beklenen sonuçlar, bir test senaryosunun parçası olarak üretilir ve çıktıları, veri ve durumlardaki değişiklikleri ve testin diğer sonuçlarını içerir. Beklenen sonuçlar tanımlanmamışsa uygun gözüken fakat hatalı olan bir sonuç doğru sonuç olarak yorumlanabilir. İdeal olarak beklenen sonuçlar test yürütmeden önce tanımlanmalıdır(ISTQB,2011).

Test uyarlama sırasında test senaryoları ve test prosedürü geliştirilir, uyarlanır, önceliklendirilir ve organize edilir (IEEE STD 829- 1998). Test prosedürü testin yürütüleceği işlem sırasını belirler. Testler, test yürütme aracı kullanılarak yürütülüyorsa işlemlerin sıralaması test komut dosyasında belirlenir (buna otomatik test prosedürü denir). Çeşitli test prosedürleri ve test komut dosyaları test yürütme çizelgesine dönüşür. Bu çizelge çeşitli test prosedürlerinin ve test komut dosyalarının hangi sırayla yürütüleceğini belirler.

Test tasarım tekniklerinin amacı test şartlarını, test senaryolarını ve test verilerini tanımlamaktır. Test teknikleri kara kutu veya beyaz kutu olarak en temel ayrımla belirtilir.

Kara kutu test tasarım teknikleri (özellik bazlı teknikler adı da verilir); test koşullarını, test senaryolarını veya test verisini çoğaltarak üretmek için test esası dokümanlarının analizini temel alan bir yöntemdir. Fonksiyonel ve fonksiyonel

olmayan testleri içerir. Kara kutu testi, test edilecek bileşenin veya sistemin iç yapısı ile ilgilenmez.

Beyaz kutu test tasarım teknikleri (yapısal veya yapı bazlı teknikler adı da verilir), bileşen veya sistemin iç yapısının incelenmesine ve yorumlanmasına dayanır. Test edilmesi gereken yapılara karar vermeleri için yazılım ekibi paydaşlarının tecrübelerini kullanmak amacıyla kara kutu ve beyaz kutu testi, tecrübeye dayalı tekniklerle de bir arada kullanılabilir.

Bazı teknikler net bir şekilde tek kategoriye sahip olurken diğerleri birden fazla kategori öğesine sahip olabilir. Bu yazı gereksinim bazlı test tasarım tekniklerini kara kutu teknikleri ve yapı bazlı test tasarım tekniklerini beyaz kutu teknikleri olarak ele almaktadır. Ayrıca tecrübeye dayalı test tasarım teknikleri de kapsam içindedir.

Gereksinim bazlı test tasarım tekniklerinin bilinen özellikleri şöyledir:

- Modeller, çözülecek problemin gereksinimlerini belirlemek için kullanılır
- Test senaryoları bu modellerden sistematik olarak türetilebilir

Yapı bazlı test tasarım tekniklerinin bilinen özellikleri şöyledir:

- Yazılımın iç çalışma mantığı ile ilgili bilgiler test senaryoları türetmek için kullanılır (kod ve detaylı tasarım bilgileri)
- Mevcut test senaryolarıyla yakalanan kapsam derecesi ölçülerek kapsamı artırmak için daha fazla test senaryosu yazılabilir.

Tecrübeye dayalı test tasarım tekniklerinin bilinen özellikleri şöyledir:

- Test senaryoları türetmek için kişilerin bilgisi ve tecrübesi kullanılır
- Yazılım, yazılımın kullanımı ve ortamı hakkında test uzmanlarının, yazılımcıların, kullanıcıların ve diğer paydaşların bilgi birikimi, bilgi kaynaklarından biridir
- Olası hatalar ve bu hataların dağılımı da diğer bir bilgi kaynağıdır(ISTQB,2011).

2.10.1. Spesifikasyon Bazlı veya Kara Kutu Teknikleri

Denklik Paylarına Ayırma : Denklik paylarına ayırmada yazılımın girdileri aynı davranışı göstermesi beklenen gruplara ayrılır, böylece aynı şekilde ele alınabilirler. Denklik payları (veya sınıflar), hem geçerli veriler (kabul edilmesi gereken değerler) hem de geçersiz veriler (reddedilmesi gereken değerler) için oluşturulabilir.

Paylar aynı zamanda çıktılar, dahili değerler, zamanla ilgili değerler (bir olaydan önce veya sonra) ve arayüz parametreleri (entegrasyon testi sırasında test edilen entegre edilmiş bileşenler) için de tanımlanabilir(ISTQB,2011).

Testler tüm sağlayacak ve sağlamayacak payları kapsayacak şekilde tasarlanabilir. Denklik paylarına ayırma tüm test seviyelerinde uygulanabilir. Denklik paylarına ayırma, girdi ve çıktı kapsam hedeflerine ulaşmak için kullanılabilir. Tüm veri giriş yöntemlerince kullanılabilir.

Sınır Değer Analizi: Denklik paylarının uç noktalarındaki girdilerin hataya sebep olma olasılığı daha yüksek olduğu için bu alanların daha yoğunlukla test edilmesi gerekmektedir. Bu teknik kullanılarak bu sınır değerlerinin bulunması amaçlanmaktadır. Bir payın maksimum ve minimum değerleri, sınır değerleridir. Geçerli bir payın sınır değeri, geçerli sınır değeridir; geçersiz bir payın sınırı ise geçersiz sınır değeridir. Testler geçerli ve geçersiz sınır değerlerini kapsayacak şekilde tasarlanabilir(ISTQB,2011).

Test senaryoları tasarım aşamasında tüm sınır değerleri için bir test seçilir. Sınır değer analizi tüm test seviyelerinde uygulanabilir. Uygulaması kolay ve hata bulma oranı yüksektir. Spesifik gereksinimler farklı sınırları belirlemede etkin olabilir. Bu teknik genellikle denklik paylarına ayırma veya diğer kara kutu test tasarım tekniklerinin bir diğer yolu olarak düşünülebilir. Ekranda kullanıcı girdisi için denklik sınıfında veya zaman aralıklarında (zaman aşımı, işlem hızı gereksinimleri) veya tablo aralıklarında (örn. tablo boyutu 128*128) olduğu gibi aralıklı veriler içeren yapılar için kullanılabilir.

Karar Tablosu Testi: Mantıksal koşulları içeren gereksinimleri yakalamak ve yazılım iç tasarım mantığını dokümante etmek için karar tabloları iyi bir tekniktir. Bir yazılımdaki karmaşık iş kurallarını kaydetmek için de kullanılabilirler. Karar tabloları oluştururken gereksinimler analiz edilir ve yazılımın koşulları ve eylemleri belirlenir. Girdi koşulları ve eylemler sıklıkla doğru veya yanlış (Boolean) olacakları şekilde ifade edilir(ISTQB,2011).

Karar tablosu testi, etki koşullarını, genellikle tüm girdi koşulları için doğru ve yanlış kombinasyonlarını ve her bir koşul kombinasyonu için ortaya çıkan eylemleri, sonuçları içerir. Karar tablosunun her bir sütunu, farklı bir koşul kombinasyonunu içeren bir iş kuralına karşılık gelir ve bu kuralla bağlantılı senaryonun yürütülmesi ile sonuçlanır. Karar tablosu testinde genellikle kullanılan test kapsamında amaç tabloda yer alan sütunlardan her biri için en az bir test senaryosunun yürütülmesi şeklindedir.

Karar tablosu testinin en önemli özelliği, test yürütme sırasında gözden kaçabilecek koşul kombinasyonlarının net bir şekilde listelenmesidir. Yazılım davranışının birden fazla mantıksal karara bağlı olduğu tüm durumlarda uygulanabilir.

Durum Geçişi Testi: Yazılımın davranışı mevcut veya geçmişteki durumuna göre değişiklik gösterebilir. Bu tür davranışlar sergileyen yazılımlar bir durum geçiş diyagramı ile gösterilebilir. Durum geçiş diyagramları test uzmanının yazılımın alabileceği durumları, durumlar arasındaki geçişleri, durum değişikliklerini (geçişleri) tetikleyen girdileri veya olayları ve bu geçişler sonucunda oluşabilecek eylemleri görüntülemesini sağlar(ISTQB,2011).

Test edilen sistemin veya bileşenin durumları farklıdır, tanımlanabilir ve belirlenebilir sayıdadır. Durum tablosu, durumlar ve girdiler arasındaki ilişkiyi gösterir ve geçersiz olan olası senaryoları ortaya koyabilir. Her durumu ve durumların genel sıralamasını kapsayacak, her eylemi deneyecek, belirlenmiş eylem sıralamalarını deneyecek veya geçersiz eylemleri test edecek test senaryoları tasarlanabilir. Durum geçişi testi çoğunlukla gömülü yazılımlarda ve teknik otomasyonda kullanılır. Ayrıca bu teknikle nesnelerin modellenmesi veya ekran-diyalog akışının test edilmesi (örn. İnternet uygulamaları veya iş senaryoları için) mümkündür.

Kullanım Senaryosu Testi: Test uzmanları kullanım senaryolarını kullanarak testler türetilebilir. Kullanım senaryosu, aktörler ve sistem arasındaki etkileşimleri tanımlayarak; bu etkileşimler sonucunda üretilen değeri gösterir. Kullanım senaryoları soyut seviyede (iş kullanım senaryosu, teknolojiden bağımsız, iş süreç seviyesi) veya sistem seviyesinde (sistem fonksiyonalite seviyesinde sistem kullanım senaryosu) tanımlanabilir. Her bir kullanım senaryosunda, kullanım senaryosunun başarılı bir şekilde çalışması için karşılanması gereken önkoşullar bulunur. Her bir kullanım senaryosu, kullanım senaryosu tamamlandıktan sonra gözlemlenebilir sonuçları ve sistemin son durumunu içeren art koşullarla sona erer. Kullanım senaryosunda genellikle bir ana (en olası) senaryo ve alternatif senaryolar bulunur(ISTQB,2011).

Kullanım senaryoları, gerçek kullanım eylemlerine dayanarak sistem boyunca işlem basamaklarının akışını tanımlar, bu nedenle kullanım senaryolarından türetilen test senaryoları, sistemin gerçek dünyada kullanımı sırasında işlem basamakları akışlarında hataları ortaya çıkarmanın kolay bir yoludur. Kullanım senaryoları, müşteri/kullanıcı katılımı ile kabul testleri tasarlamada da çok kullanışlıdır. Ayrıca, bağımsız bileşen testlerinin göremeyeceği şekilde, farklı bileşenlerin etkileşiminin neden olduğu bütünleşme hatalarının ortaya çıkarılmasını sağlar.

2.10.2. Yapı Bazlı veya Beyaz Kutu Teknikleri

Yapı bazlı veya beyaz kutu testi, aşağıdaki örneklerde görülebileceği üzere yazılımın iç çalışma mantığına dayanır:

- Bileşen seviyesi: bir yazılım bileşeninin yapısı, örn. komutlar, kararlar, dallar ve yollar
- Entegrasyon seviyesi: Test için ele alınan yapı bileşenlerin birbirlerini nasıl çağırdığını gösteren bir çağrı ağacı olabilir (örnek modüllerin diğer modülleri çağırdığı bir diyagram)
- Sistem seviyesi: Yapı; menü yapısı, iş süreci veya web sayfası yapısı olabilir Bu bölümde kod yapısıyla ilgili üç çeşit test tekniğinden bahsedilecektir: komut, karar ve dal test teknikleri(ISTQB,2011).

Komut Testi ve Kapsam: Komut testinde kapsam çalıştırılan komutların yüzdesi ele alınarak belirlenir. Komut kapsamı yüzdesi, test senaryoları tarafından kapsanan (tasarlanmış veya yürütülmüş) yürütülebilir komut sayısının test edilen koddaki tüm yürütülebilir komutların sayısına bölünmesiyle elde edilir(ISTQB,2011).

Karar Testi ve Kapsam: Dal testi ile ilgili olan karar kapsamı, bir test senaryo grubu tarafından oluşturulmuş karar çıktılarını (örn. bir IF komutunun doğru ve yanlış sonuç üreten seçenekleri) dikkate alır Karar kapsamı, test edilen kodda test grubu tarafından oluşturulmuş karar çıktılarının koddaki tüm karar çıktılarına bölünmesiyle elde edilir. Karar testinde karar noktaları boyunca bir kontrol akışını izlendiği için karar testi bir çeşit kontrol akış testi biçimidir. Karar kapsamı, komut kapsamından daha güçlüdür; %100 karar kapsamı %100 komut kapsamını sağlar, ancak bunun tersi olmaz(ISTQB,2011).

Diğer Yapı Bazlı Teknikler: Karar kapsamının ötesinde daha güçlü yapısal kapsam seviyeleri vardır; örn. koşul kapsamı ve çoklu koşul kapsamı. Kapsam kavramı diğer test seviyelerinde de uygulanabilir. Örneğin, entegrasyon seviyesinde, bir test senaryo grubu tarafından denenmiş modüllerin, bileşenlerin veya sınıfların yüzdesi, modül, bileşen veya sınıf kapsamı olarak ifade edilebilir. Kodun yapısal testi için araç desteği oldukça yararlıdır(ISTQB,2011).

2.10.3. Tecrübeye Dayalı Teknikler

Tecrübeye dayalı testte, testler benzer uygulamalar ve teknolojilerle daha önce çalışmış test uzmanının becerisine, sezgilerine ve tecrübesine dayanılarak türetilir. Sistematik teknikleri artırmak için kullanıldığında bu teknikler, resmi teknikler tarafından kolayca yakalanamayan özel testleri belirlemek için kullanılabilir. Fakat bu tekniğin etkisi test uzmanının tecrübesine bağlı olarak çeşitlilik gösterecektir(ISTQB,2011).

Genellikle kullanılan tecrübeye dayalı tekniklerden birisi de hata tahmin etmektir. Genellikle test uzmanları tecrübelerine dayanarak hataları tahmin eder. Hata tahmin etme tekniğine yönelik bir yaklaşım, olası hataların listesini çıkarmak ve bu hataları hedef alan testler tasarlamaktır. Bu sistematik yaklaşıma kusur ortaya çıkarmaya yönelik saldırı adı verilir. Hata listeleri, tecrübeye, mevcut hata ve arıza verilerine ve yazılımın neden başarısız olduğu ile ilgili yaygın bilgilere dayanarak oluşturulur. FMEA maddelerinin oluşturulması sırasında bu teknikten faydalanılabilir.

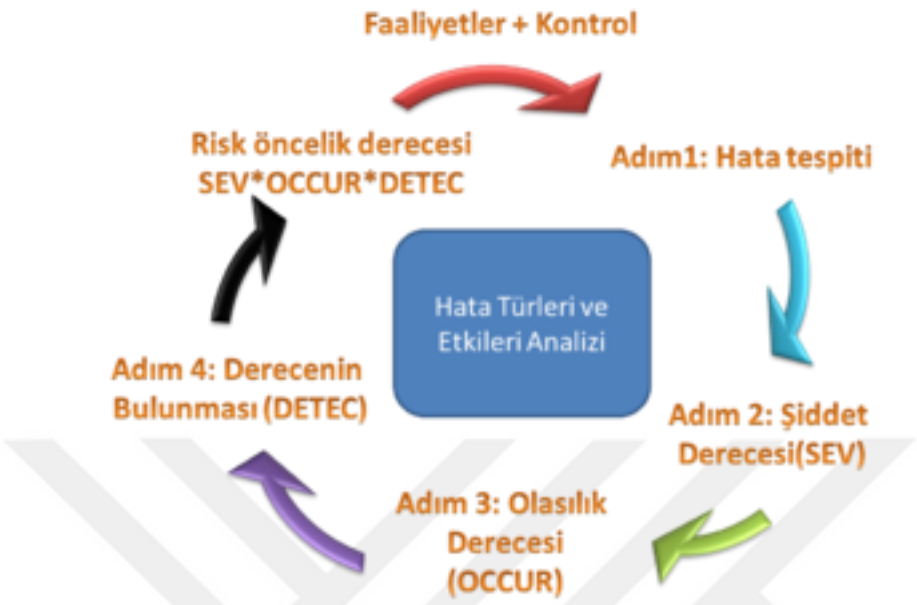
Keşif testi, test uzmanının daha iyi ve daha yeni testleri tasarlamak için test yaparken bilgi dağarcığını kullanarak ve bu testleri yürüterek esasen test tasarımını kontrol altına aldığı gayri resmi test tasarım tekniğidir. Bu teknik, eksik gereksinimler ve ciddi zaman sınırlaması olan durumlarda ya da daha resmi testleri artırmak için oldukça kullanışlıdır. Test sürecinde genel kontrol olarak kullanılabilir ve bilgi durumuna göre en ciddi hataların bulunmasını sağlayabilir.

Yapay zeka teknikleri kullanılarak önceden yapılan kullanıcı testleri, test uzmanı testleri gibi kişilerce manuel yapılmış senaryolar sisteme öğretilerek İGOT modelince bu tekniklerin kullanılması uygundur.

2.11. Failure Mode Effect Analysis(FMEA)

Hata türleri ve etkileri analizi; bir sistemin potansiyel hata türlerini analiz etmek için hataları olasılıklarına ve benzerliklerine göre sınıflandıran bir ürün geliştirme ve operasyon yönetim prosedürüdür. Başarılı bir hata türü analizi işi, benzer ürünlerin veya proseslerin geçmiş deneyimlerine dayanarak hata türlerinin tanımlanmasına yardımcı olur, bu hataların sistemden minimum kaynak kullanımı ve çabayla atılmasını sağlar ve bununla beraber geliştirme zamanını ve maliyetini düşürür. Genellikle üretim sektöründe ürünlerin çeşitli aşamalarında kullanılmakla beraber hizmet sektöründe de

kullanım alanı artmıştır(Anonim,2016). Şekil 2.6.'de FMEA'in temel süreci görülmektedir.



Şekil 2.6. FMEA Döngüsü (Pekşen, 2011)

FMEA'nin temel fikri hatayı sonradan bulmak ve düzeltmek (hata yönetimi) yerine hataları erkenden tanıyarak tedbirli bir şekilde önlemek ve tasarım aşamasından itibaren hataların olası nedenlerinin değerlendirilmesidir. Böylece aksi halde üretim aşamasında çıkan kontrol ve hata maliyetlerden ve hatta müşteri maliyetlerinden (müşteri) kaçınılabilir ve toplam maliyetleri azaltılabilir. Sistematik bir yaklaşımla ve bu yaklaşımdan gelen bilgilerle ayrıca tasarım hatalarının yeni ürün ve süreçlerde tekrarlanması önlenir(Namenlos,2011).

Bir sürecin, üretime hazır hale gelmesinin ardından veya üretime geçmiş bir proseste, önemli olan sürecin veya ürünün güvenilirliğini sağlamaktır. Güvenilirlik ürünlerin veya proseslerin önemli bir özelliğidir. Aynı zamanda müşteri tatminini sağlamakta etkisi çok fazla olan bir faktördür. Müşteriler kullandıkları ürünün hizmet süresinin uzun ve aynı zamanda sorunsuz bir proses olmasını istemektedirler. Bu nedenle ürünün veya sürecin güvenilirliğini sağlamak için atılacak adım, ortaya çıkabilecek olan hataların türlerini ve bunların ürün ya da sürece etkilerini belirleyebilecek bir risk analizinin yapılması ve kurulacak veya kurulmuş olan bir sürecin güvenilirliğinin kontrol altına alınmasıdır(Anonim,2000).

FMEA tecrübeye dayalı test tekniği olarak İGOT modelinde kullanımı gerçekleştirilebilir.

2.12. Yazılım Kalite Metrikleri

Metrikler yazılım kalitesinin belirlenmesi ve iyileştirilmesi çalışmalarında etkin biçimde kullanılmaktadır. Bu çalışmalar sonucunda aşırı bağımlı, karmaşık ve hataya eğilimli modüllerin belirlenmesi gibi önemli bilgiler elde edilebilmektedir. Bu bilgiler yazılım kalitesinin iyileştirilmesinde, daha sonra hangi kısımların öncelikli olarak test edileceğine karar verilmesinde, bakım için gereken bütçe ve zaman analizlerinde kullanılabilmektedir. Ancak tüm bu çalışmaların başarımı büyük ölçüde doğru metriklerin tanımlanmasına, bu metriklerin doğru biçimde ölçülmesine ve sonunda doğru yorumlanmasına bağlıdır(Erdemir, Tekin, Buzluca,2008).

Yazılımın iç özellikleri gruplandırılarak projenin gidişatı hakkında bilgi sahibi olunması amaçlanarak; yazılımın esnekliği, yeniden kullanılabilirliği ve bakım kolaylığını etkileyen yazılım iç özellikleri belirlenmiştir. Bu özelliklerin belirlenmesinde McCall ve ISO/IEC 9126 kalite modellerindenyararlanılmıştır. Tanımlanan yazılım iç özelliklerin yazılımda etkilediği kalite özellikleri Çizelge 2.5.'de verilmektedir(Canbaz,Buzluca,2012).

Çizelge 2.5. Yazılım Kalite Değişkenleri

	Kalite Değişkenleri		
	Yeniden Kullanılabilirlik	Bakım Kolaylığı	Esneklik
Bağımlılık	*	*	
Uyumluluk	*	*	
Kalıtım	*		*
Karmaşıklık	*		*

Kod kaydetme politikaları konularak kodsal dönüşümler sırasında oluşabilecek hataların giderilmesini İGOT modeli savunmaktadır. Sonsuz döngülerin oluşması gibi kullanıcı testinde tespiti çok zor olan senaryoların yazılım aşamasında çözülmesi kalitenin ve projenin verimli sürdürülmesini sağlayacaktır.

3. MATERYAL VE YÖNTEM

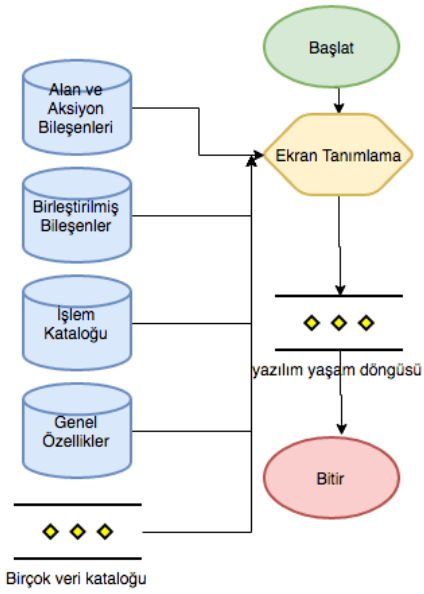
Modelin uygulanması sırasında oluşturulacak olan sistem için öncelikle yazılım yaşam döngüsünün planlama adımında sistemin tasarlanması ve projenin alt yapısı ile birlikte hazırlanması gerekmektedir. Bu sistemin hazırlanmasında test yaşam döngüsü de devreye alınarak erken testlerin başlaması da kolaylaştırılmalıdır. Ayrıca bu şekilde senaryoların daha da iyi gereksinim odaklı olmaları sağlanabilir.

Test yaşam döngüsü planlama adımında belirlenen stratejiler modelin uygulandığı sistemin geliştirilen proje için daha verimli çalışmasında önemli rol oynamaktadır.

İGOT modelinin uygulandığı sistem geliştirmelere açık olmalıdır. Yapılan her yeni tanımlama sistemin kütüphanesinde tutulmalı ve öğrenim oranını artırmalıdır. Tanım tabanlı bu sistem ve modelin diğer özellikleri alt başlıklarda anlatılmıştır.

3.1. Tanım Tabanlı Sistem

Yazılım projelerinin süreçlerinde modele tanımlama işlemleri yapılarak ürün ve özellikleri sisteme tanıtılmış olmalıdır. Ürünün analizinde ortaya çıkan özellikler başlangıç olarak sisteme girişi yapılmalıdır. Şekil 3.1.'de de görüldüğü gibi veri katalogları oluşturulmalı ve üretilen her ürün de tanımlanarak tekrar kullanılabilir olmalıdır.



Şekil 3.1. Ekranların Oluşturulmasında Sistemdeki Tanımlamalar

Projedeki ekranlar, ekrandaki alanlar ve özellikleri, ekrandaki aksiyonlar ve özellikleri bilgi olarak modele girilmelidir. Tanımlama işlemi kullanılacak veritabanı tablolarının girilmesiyle de sağlanabilir. Böylelikle kodlamada kullanılacak olan birçok bileşen modele tanımlanarak gerekli kodların ve test senaryosu aksiyonlarını içeren kodların üretimi için bilgiler modele tanıtılmış olur.

Tüm bu tanımlamalar modelde daha önceden yer alan bilgiler ile ilişkilendirilerek yapılmalıdır. Bir aksiyon için kaydetme işlemi yapacağı tanımlanacaksa kaydetme işlevini içeren tanım modelin içinde yer almış olmalıdır. Aynı şekilde bir özellik sınırlar içeriyorsa bu sınırı betimleyen yapı modelin içinde yer almış olmalıdır. Böylelikle girilecek veriler ile ilişkilendirilmek üzere tanımlı bu yapılar arka planda üretilecek olan kodların, test senaryolarının yapı taşı olarak kullanılabilmelidirler.

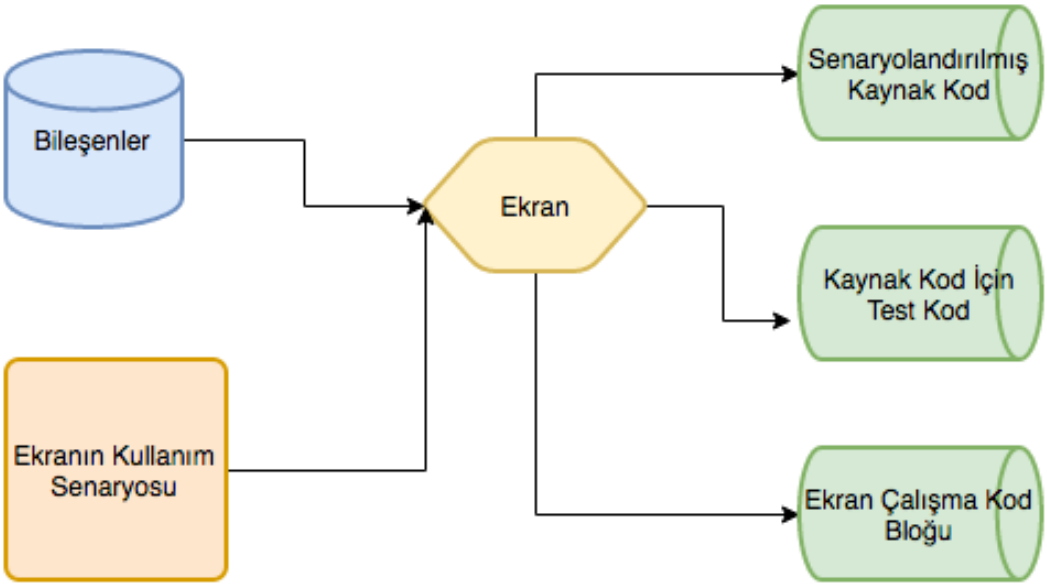
3.2. İş Gereksinimi Odaklı Test Senaryolarının Oluşturulması

Ekrandaki veya veritabanındaki bileşenlerin tanımlanmasıyla modele tanıtılan özellikler test tekniklerinin kullanılmasıyla senaryolara ve senaryoları sağlayan kodlara dönüştürülür. Alt başlıklarda test seviyelerine göre senaryo üretimi de incelenmiştir.

3.2.1. Birim Seviyesi Test Senaryolarının Üretimi

Birim seviyesi test senaryoları model içinde tanımlanan alanlar aksiyonlar gibi yazılımsal özelliklerin verilerini kullanarak bu özelliklerin tanımlanan alan tarafından sağlanıp sağlanamayacağının kontrolünü yapan test kodlarının üretilmesi sağlanmalıdır. Bu durumun devamında test senaryosunun çözüm aksiyonunun kodsal karşılığı üretilebilmesi durumunda kod üretimi sağlanmalıdır.

Statik test tekniklerinden olan kodların test edilmesi kısmı bu seviyede uygulanmalı ve test projeleri de kod üretiminde olduğu gibi üretilerek çalıştırılması sağlanmalıdır. Şekil 3.2.'de de anlatıldığı üzere ekranlara yapılan veri girişleri sonrasında bileşen özelliklerine göre test senaryolarının aksiyonlarının yer aldığı senaryolandırılmış kaynak kod, senaryolandırılamayan kodların çalışma durumlarının kontrolü için test senaryolarını içeren test projelerinin kaynak kodu ve otomasyon araçlarıyla ekranın kullanım senaryosunun oynatılmasıyla yazılımın gerçekleşmesi sonrasında kullanılabilecek ekran çalışma kod bloğu çıktı olarak alınmalıdır.

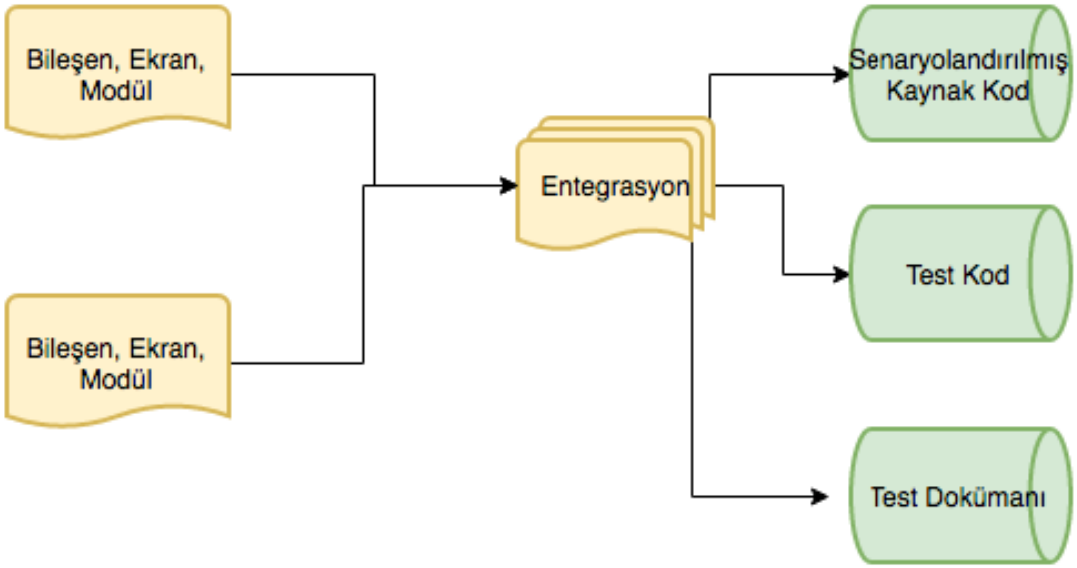


Şekil 3.2. Ekran Verileri ve Çıktıları

Ekranın oluşması sonrasında koda dönüştürülemeyen senaryolar için çıktı olarak test dokümanı da üretilebilir.

3.2.2. Entegrasyon Seviyesi Test Senaryolarının Üretimi

Entegrasyon seviyesi testlerde birim testlerine göre daha büyük yazılım parçaları ele alındığından modelde tanımlı olan bileşen, ekran özellikleri gibi tanımlamalar kullanılarak kodsal test senaryoları yazılabileceği gibi daha büyük yazılım parçaları için tanımlı özelliklere bağlı olarak dokümante edilebilecek test senaryoları çıktı olarak sağlanmalıdır. Şekil 3.3.'de çıktılar görülmektedir.



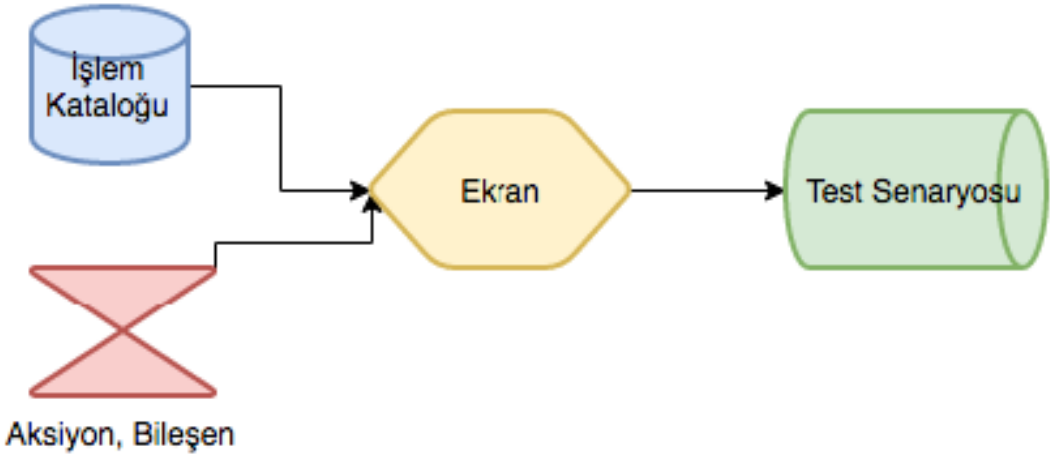
Şekil 3.3. Entegrasyon Verileri ve Çıktıları

3.2.3. Sistem Seviyesi Test Senaryolarının Üretimi

Modele tanımlanan yazılımın bütüne bakan sistemsel bilgi, altyapı bilgisi, kullanılan teknoloji bilgisi gibi temel özellikleri ele alınarak kodsal test senaryoları yazılabileceği gibi sistem için tanımlı özelliklere bağlı olarak dokümante edilebilecek test senaryoları çıktı olarak sağlanmalıdır.

3.2.4. Fonksiyonel ve Fonksiyonel Olmayan Test Senaryolarının Üretimi

Modelde tanımlı yazılımın fonksiyonlarının kullanılmasıyla fonksiyonel test senaryoları üretilebilmelidir. Aksiyonların fonksiyonlarını gerçekleştirmesi gibi temel düzeydeki test senaryoları başlıca yer almalıdır. Özellikle bu fonksiyonlar aksiyonlar ile eşleştirilmek üzere önceden modelin yapısında tanımlanmış işlevler ile ilişkilendirilmiş olmalıdır. Şekil 3.4.'de aksiyonların veya bileşenlerin tanımlı işlemlerle eşleştirilmesi ile işlem sonucu baz alınarak fonksiyonel başarımın test edilmesi için test senaryosunun oluşması şemalandırılmıştır.



Şekil 3.4. Bileşenlerin İşlem Kataloğu ile Eşleşmesi

Fonksiyonel olmayan testler için üretilecek senaryolar için test otomasyonu entegresi kullanılabileceği gibi, sistemin gereksinimlerine dayanarak; performans, bütünlük gibi testlerin senaryolarının çalıştırılması için da sistemde ayrı bir modül olarak senaryoların çalıştırılacağı motorlar tanımlanabilir. Böylelikle İGOT modelinde eş zamanlı çalışan test senaryoları yer alabilmelidir. Bu durum ayrıca büyük ölçekli firmaların büyük ölçekli yazılım ürünleri için gereksinimin boyutuna göre donanımsal ihtiyaçlarda gerektirebilir.

Fonksiyonel olmayan test için gereksinimlerin adreslendiği senaryo havuzu oluşturularak tanımlanmalıdır.

3.2.5. Test Senaryolarının Oluşturulmasında Test Tekniklerinin Kullanımı

Test teknikleri hakkında üstteki bölümlerde bilgilendirme yapılmıştır. Statik ve dinamik test tekniklerinin model içinde kullanımı aşağıda incelenmiştir.

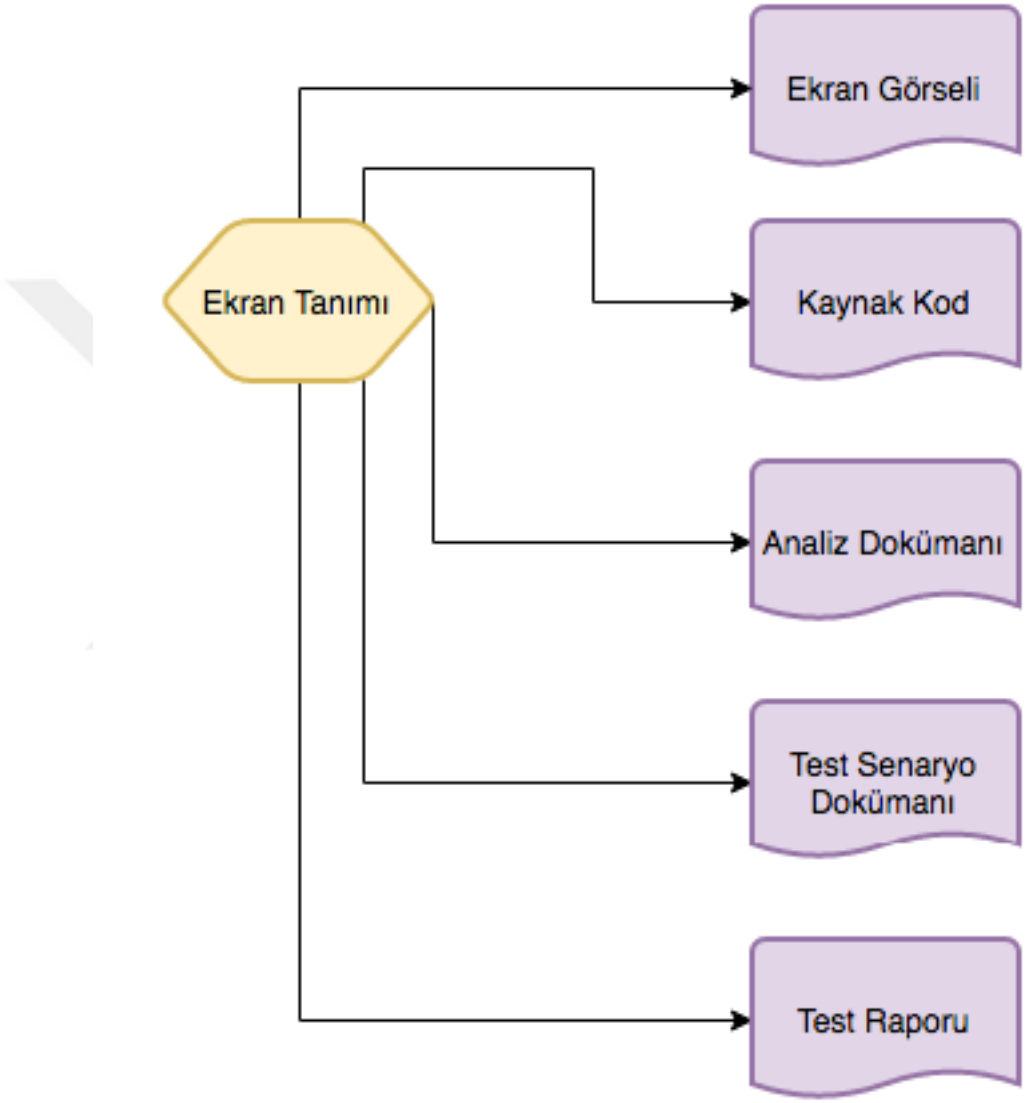
3.2.5.1. Statik Test Tekniklerinin Kullanımı

Model ekranların tasarımlarının oluşturulmasıyla gözden geçirme sürecinde etkin rol oynar. Teknik gözden geçirmelerde de yönetim aracı olarak kullanılabilir.

Kodların üretimi sırasında mümkünlüğe bağlı olarak üretilen senaryolara göre doğru çalışan kodlar üretilebilir yada test senaryolarını içeren test projeleri üretilebilir.

Modelin tanımlama kısmından sonra ortaya çıkan ekranlar da prototip ekran olarak kullanılabilir ve gözden geçirme tekniği için kaynak oluşturabilmektedir.

Şekil 3.5.'de İGOT modeline ekran verilerinin girilmesi sonucu oluşan dokümantasyon çıktıları gösterilmektedir. Bu dokümanlar gözden geçirme için kullanılabilmektedir.



Şekil 3.5. Statik Test için Gözden Geçirme Kaynakları

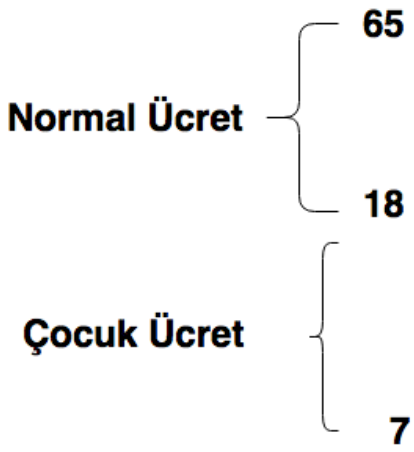
3.2.5.2. Dinamik Test Tekniklerinin Kullanımı

Dinamik test tekniklerinin kapsadığı birçok kara kutu ve beyaz kutu test tekniği vardır. Bunlardan bazılarının model yapısında kullanımları aşağıda incelenmiştir.

3.2.5.2.1. Kara Kutu Test Tekniklerinin Kullanımı

Kara kutu testlerinde yapısal incelemeler yer almadığı gibi ekranların veri girişindeki bilgiler direk olarak kullanılabilmelidir. Bu bilgilerden bileşenlere giriş ve çıkış değerleri baz alınır.

Denklik Paylarına Ayırma: Bileşenler için seçilen girdiler aynı davranışı göstermesi beklenen gruplara ayrılır, böylece aynı şekilde ele alınabilirler. Geçerli veya geçersiz değerler için elde edilmesi gereken sonucu içeren kod bloğu yazılır. Burada bileşene bilgi olarak girilebilecek değerlerin kontrolünü içeren kod bloğu kaynak kod olarak oluşturulabilir yada bu değerlerin verilerek kaynak kodun test edildiği bir test projesi oluşturulabilir. Eğer ki bu şekilde otomatize edilebilecek bir değer aralığı yok ise bu değer aralıklarının belirtildiği aksiyon sonuç içeren test senaryosu dokümanı çıktısı oluşturulmalıdır.



Şekil 3.6. Denklik Payları

Şekil 3.6.'da görüleceği gibi değerlerin sınıflandırılması sonucunda hangi girdinin hangi sonuca ulaştırılacağı test senaryosuna dönüştürülmelidir. Belirtilmeyen sınırlar dışında kalan değerlerin belirlenerek aksiyon sonucu belirtilen işlemin gerçekleşmemesini sağlayan kod yazımı yada bu durumu test eden test projesi yada test senaryosu dokümanı oluşturulmalıdır.

Bu teknik için modelde sadece sağlayan değerlerin veri olarak girilmesi arta kalan değerler için işlem yapılmaması yada belirtilen uyarıların verilmesi sağlanabilir. Örneğin, ekrandan 6 değerinin girilmesi durumunda işlem yapılmayabilir. İGOT modeli için oluşturulan uyarı kataloğuna "Bu değer için işlem yapılmamaktadır." Tanımlaması

yapıldığında ve bileşen ile eşleştirildiğinde bileşen için sınıflandırmanın dışındaki tüm değerler için otomatik olarak bu uyarının verilmesi senaryolaştırılmış kod tarafından sağlanır.

Sınır Değer Analizi: Denklik paylarına ayırma tekniğinden farklı olarak bu teknikte tam sınırdaki değerler için senaryo yazılır. Şekil 3.6.'da görülen 18 sınır değeri için yapılması gereken işlemi test eden senaryo yazılmalıdır. Ayrıca 18,2 gibi farklı bir tipteki değerin işlem sonucunun doğruluğunun testi gibi durumlar için de senaryo üretimi sağlanabilir.

Karar Tablosu: Bu teknik ile belirtilen özelliklerin kombinasyonu sonucunda ortaya çıkacak olan sonucun test edildiği senaryolar üretilmelidir.

Çizelge 3.1. Şifre Karar Tablosu

Veri Tipi	Aralığı	Girdi-Sonuç
Doğal sayı	1-10	0-olumsuz
Tam sayı	10-100	0,2-olumsuz
Türkçe Yazı	10-25 karakter	Büşra-olumlu
Yazı	3-10 karakter	Mi-olumsuz

Çizelge 3.1.'de görülen karar tablosu bir şifre giriş senaryosu için Karar tablosunu göstermektedir. Buradaki değerler iç içe koşullar ifade edilerek yada daha verimli bir kodlama işlemi ile kontrol edilerek test senaryosu oluşturulabilir. Bütün tekniklerde kullanılacak özellikler ne kadar çok belirtilirse oranla daha fazla test senaryoları oluşturulabilir. Burada belirtilen tekniklerin dışındaki birçok kara kutu test tekniği de yine bu modelde uygulanarak kullanılabilir.

3.2.5.2.2. Beyaz Kutu Test Tekniklerinin Kullanımı

Beyaz kutu testlerinde yapısal incelemeler üzerine test senaryoları yer aldığından ekran veri girişindeki veriler yorumlanarak kullanılabilir. Bu durumdaki yorumlama

bilgilerin koda dönüşümündeki strateji, kullanılan yöntem ve araçlar, seçilen akışları içerebilmelidir.

Komut Testi: Her bir durum en az bir kere değer verilip sonuçlar alınacak şekilde test senaryosu üretilir. Öncelikli olarak kod parçasında bir yol seçilir ve her seferinde değişiklik yapılarak test senaryosu oluşturulur.

Komut testi tekniği kullanılarak yazılan test senaryolarında tüm durumların en az bir kere test edilmesine dikkat edilmelidir.

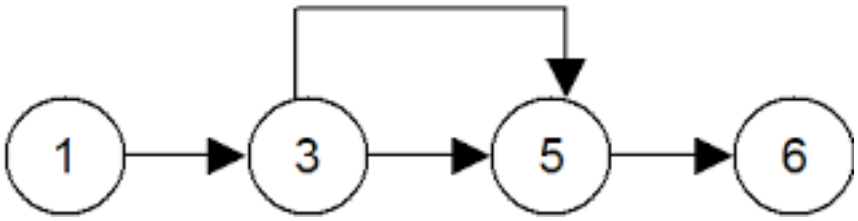
Karar Testi: Eğer bir karar verme durumu, doğru yada yanlış gibi bir sonuç varsa karar testi tekniği kullanılarak test senaryosu oluşturulmalıdır. Karar testinde de komut testinde olduğu gibi kod parçasında bir yol seçilerek ve türevleri oluşturularak test senaryoları oluşturulur.

Yazılımın geliştirimi sırasında ortaya çıkan hataların %65'lik kısmı birim testler ile bulunmaktadır. Birim testler içerisinde en çok kullanılanı akış kontrol testidir. Araştırmalar göstermiştir ki hataların %50'lik kısmı akış kontrol testleri sayesinde bulunmaktadır(YazılımcılarDünyası,2017). Bu test tekniği; yapısal programlama dillerinde yapısal olmayan dillere göre daha etkilidir.

Yol(Path) Araçları: Akış kontrol testi ile yazılımlar içerisindeki hataların önemli bir kısmı bulunabilir. Koşul ve durum testleri beraber kullanılması başarıyı arttıracaktır. Bu nedenle 2 test yöntemi de bir arada kullanılmalıdır. Test işlemini yaparken tek bir yol üzerine bağlı kalmayıp olabilecek diğer yolları da test etmek gereklidir.

Aşağıda tam sayı olarak girilen verinin pozitif tam sayı olarak çıktısını veren bir fonksiyon örneği verilmiştir.

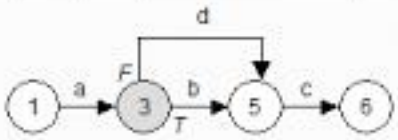
```
/* ABS
   This program function returns the absolute value of the integer
   passed to the function as a parameter.
    INPUT: An integer.
    OUTPUT: The absolute value if the input integer.
*/
1               int ABS(int x)
2               {
3               if (x < 0)
4                   x = -x;
5               return x;
6               }
```



Şekil 3.7. Fonksiyon İçin İki Yol Seçeneği

Şekil 3.7.'de fonksiyonun çalıştırılması durumunda iki tane yol izleyebileceği gösterilmektedir. Her bir yol karar komut test tekniği uygulanarak test senaryosu üretilmelidir.

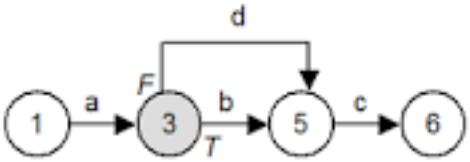
PATHS	PROCESS LINKS				TEST CASES	
	a	b	c	d	INPUT	OUTPUT
abc	√	√	√		A Negative Integer, x	-x
adc	√		√	√	A Positive Integer, x	x



Şekil 3.8. Test Senaryosunda İzlenen Yol ve Durumları

Şekil 3.8.'de test senaryosu için yol seçenekleri ve bu yol için giriş ve çıkış değerleri gösterilmiştir. Şekil 3.9.'da da benzer olarak T ve F kararları koşulun doğru(T) yada yanlış(F) olması durumunda izleyeceği yolu belirtmektedir.

PATHS	DECISIONS	TEST CASES	
		INPUT	OUTPUT
abc	T	A Negative Integer, x	-x
adc	F	A Positive Integer, x	x



Şekil 3.9. Test Senaryosunda Karara Bağlı İzlenen Yol

3.3. Test Otomasyonunun Entegrasyonu

Açık kaynak kodlu olan test otomasyonlarının gerekli kodlamalar yapılarak modelin içinde kullanılması sağlanabilir. Modelde belirtilen sisteme yapılan tanımlamaların kullanılması ve otomasyonun sonuçlarının modele çıktı verileri olarak bilgi sunması sağlanarak entegrasyon sağlanabilir.

Modele tanımlanan prototip ekranların kullanımı hakkındaki tanımlamalar da otomasyon için çalıştırma kod blokları oluşturulmuş olur. Bu kod blokları kullanıcının yazılımı kullanım senaryolarını veya birim testlerini de içerebilir. Böylelikle sistemde basit düzeyde kullanıcı kullanım senaryosu da test senaryosu olarak oluşturulmuş olur. Birim testlerinde de modelde anlatılan gereksinim odaklı senaryoların dışındaki temel basit senaryoların üretimi sağlanabilir.

Model için seçilen test otomasyonunun özelliklerine göre test sürecinin yönetimi ve raporlanması için de kullanılabilir.

3.3.1. Regresyon ve Tekrar Test Senaryolarının Çalıştırılması

Modelde oluşturulan senaryoların tekrar çalıştırılması ile tekrar testleri gerçekleştirilmiş olur. Bu testler genellikle hataların düzeltilmesi sonrasında

oluşabilecek yeni hatalara karşı aynı senaryoların çalıştırılmasıyla önlem alınmasını sağlar.

Regresyon testlerinde de tekrar testlerinde de üretilen yeni kodlar için test senaryosu oluşturma motoru ayrı olarak sistemde yer almalı ve değişiklikleri tespit edebilmelidir. Değişiklikler kaynak kodların karşılaştırılmasıyla tespit edilebilir. Böylelikle oluşturulan yeni kodlar için birim testi senaryoları oluşturulabilir. Regresyon testlerinin kapsadığı kodların büyük olması durumu bakım ve gereksinim değişikliklerine sebep olacağından diğer başlık altında ayrıca incelenmiştir. Örneğin ekran tasarımında bir değişiklik yapılarak regresyon testinin istenmesi gereksinim değişikliğini ilgilendirir. Ancak bir fonksiyonun çalışması için yeni bir koşulun eklenmesi regresyon testini ilgilendirir. Bu değişikliğin başka bileşenleri etkilediğinin kontrolü ise tekrar testlerini ilgilendirir. Bu şekildeki sınıflandırmanın sonucunda tekrar testlerinde önceki senaryolar kullanılabilir ancak regresyon testleri için eklenen kodlar için yeni senaryoların oluşturulması ve çıktılara yansıtılması gereklidir.

Test Otomasyonunun verisi olması için oluşturulan kod bloklarının özellikle yapılan değişikliklerin sisteme olan etkisini ölçmek için yapılan tekrar testlerinde ve regresyon testlerinde kullanılması amaçlanır. Böylelikle otomasyon kod bloklarını kullanarak yazılımda yapılan değişikliklerin hatalara sebep olup olmadığını ölçümleyerek bildirir. Yapılan en ufak değişikliklerde bile bu senaryolar otomasyon tarafından çalıştırılarak hata oranları düşürülebilir. Ayrıca bu senaryolar hataların düzeltilmesinden sonra yeniden yazılan kodların hataya sebebiyet verip vermediğinin kontrolü için de kullanılmalıdır. Tekrar test işlemi de bu senaryolar çalıştırılarak gerçekleştirilmiş olur.

3.4. Bakım ve Olası Gereksinim Değişikliklerinde Modelin Tavrı

Birçok yazılım ürünü ne yazık ki ilk tasarlandığı şekilde tamamlanamamakta ve yeniden tasarlanmaktadır. Bu değişiklikler kimi zaman sadece bir bileşen üzerinde olabileceği gibi yazılımın alt yapısını değiştirecek düzeyde de olabilmektedir. Bu gibi durumlarda model tekrar tasarlanan bileşenleri tespit edip değişikliklerin olduğu özellikleri güncellemeli senaryolarda eski özelliklere ait aksiyonları çıkarmalı ve yerine yeni aksiyonları ve çözümleri koymalıdır. Ayrıca senaryolarda üretildiği veya güncellendiği tarih bilgileri de tutulmalıdır. Değişiklikler çıktılara versiyon bilgileriyle yansıtılmalıdır.

Fonksiyonel olmayan bakım yada değişiklik isteklerinde de sistem veri güncellemesi yapılarak test senaryoları tekrar oluşturulmalıdır. Fonksiyonel olmayan test senaryolarının üretilmesi yanı sıra fonksiyonel test senaryolarında da değişikliğe uğrayan kısımların senaryolarının üretimi sağlanmalıdır.

3.5. FMEA Kullanımı

Tanımlamalarla oluşturulan katalog bilgilerinin FMEA maddeleriyle ilişkilendirilmesi ile bileşenlerin kullanıldığı alanlardaki hata modları tespit edilebilir ve maddelere bağlı test durumları da senaryo olarak üretilebilir. Ayrıca hata önceliklerinin oranına göre değerlendirilmesiyle senaryoların mümkünlüğüne göre otomatize edilmesi yada manuel yürütülmesi, senaryoların üretilmesi yada üretilmemesi, senaryoların çalıştırılması yada çalıştırılmaması da seçime bağlanabilir.

Klasik risk analizden farklı olarak olasılık ve ağırlık faktörünün yanında saptanabilirlik çarpanının eklemesi önemli bir farkıdır. FMEA çalışmasında belirlenen bütün hatalar için olasılık, ağırlık ve saptanabilirlik tahmini yapılmaktadır. Hata önceliklerini belirlemede üç ana faktör vardır:

• Ortaya çıkma, (Olasılık) (O)

• Ağırlık, (Şiddet) (A)

• Saptama (Farkedilebilirlik-Saptanabilirlik) (S)

FMEA için RÖS; Olasılık (O), Ağırlık (A) ve Saptama (S) değerlerinin çarpılması ile bulunur.

RÖS = O * A * S

RÖS değerine göre de hangi hatanın daha öncelikli olduğunu ve bunların üzerinde daha çok durulması gerektiğini söyleyerek zamandan ve imkânlardan tasarruf etmeyi sağlar(Kahraman,Demirer,2010).

3.6. Kod Üretimi

İGOT modeline girişi gerçekleştirilen verilerin html olarak yorumlanması, kaynak kod üretimi ve test senaryosu dönüşümleri(üretimi, kullanımı, test projeleri kaynak kodları vb.) xslt tekniği kullanılarak sağlanır. Xslt ile kod üretimi 2. bölümde anlatılmıştır.

3.7. Yazılım Kalite Metriklerinin Uygulanması

Yazılım kalite metrikleri senaryolaştırılmış kaynak kod oluşturulurken, kaynak kod için test senaryolarını içeren test projesi oluşturulurken kullanılması gerekmektedir. Bu durum test senaryolarının okunabilirliğini de kolaylaştırmaktadır.

Modelin kullanılacağı sistemin, kaynak kodları merkezi saklama alanındaki saklama araçlarına, yöntemlerine göre kaydetme işlemi öncesinde kontrol işlemi yapılmalıdır. Çek edilecek dosyanın içeriği kaydetme öncesinde okunarak politika da belirlenen test senaryoları uygulanmalıdır. Bu işlem kod okuyucular(Code review işlemi) tarafından da yapılabilir.

4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA

İGOT modelinin üzerinde bir modülün ekranının verilerinin girilmesi ve test senaryolarının oluşturulması aşağıda incelenmiştir. Burada yazılım yaşam döngüsünün analiz safhasından başlanacak olunup sistem analisti rolünün verileri girmesi ve sırasıyla, kodlama safhasında yazılım uzmanının önüne gelen senaryolandırılmış kodlar ve test projeleri, test aşamasında da test uzmanının birçoğu çalıştırılmış ve düzenlenmiş test senaryoları ve kullanıcı gerektiren dokümante edilmiş test senaryoları ile karşılaşması anlatılmaktadır.

Uygulamadaki sistemde veri kataloglarının oluşturulmuş olduğu varsayılmaktadır. Ayrıca sistemde ekran bileşenleri, bileşenlere ait özelliklerin ve aksiyonların bilgilerinin de veri kataloglarında yer aldığı varsayılmıştır. Görsellerde yer alan program sadece diyagram programıdır uygulama amaçlı tekrar tasarlanarak gösterilmiştir. Modelin kullanıldığı bir program bulunmamaktadır. Modelin kullanılacağı sistemin program olarak geliştirilmiş olduğu varsayılmıştır.

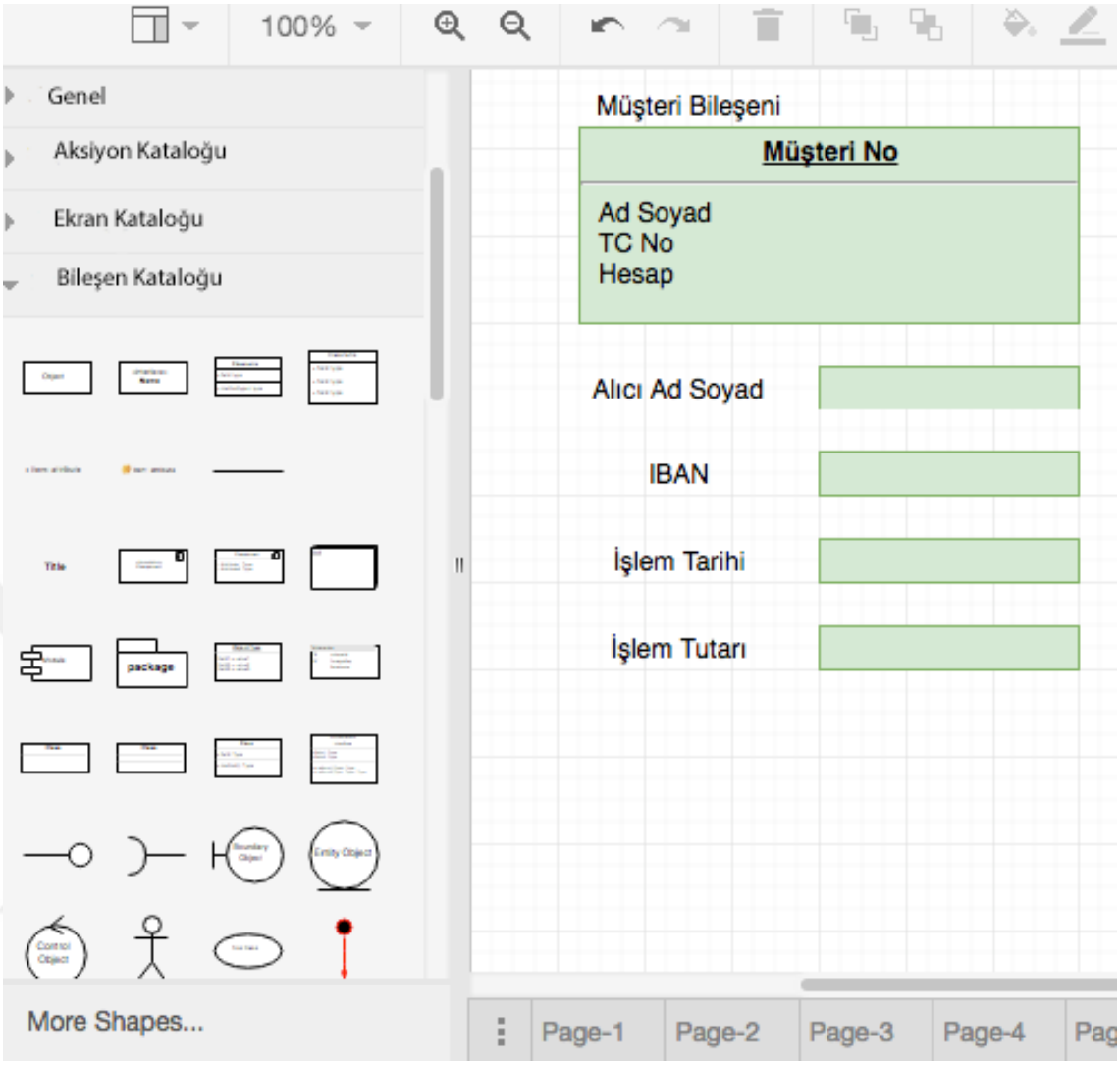
Bankacılık sektöründe kasa görevlisinin kullanımına yönelik masaüstü platformda EFT(Elektronik Fon Transferi) işlemi için bir ekran gereksinimi üzerine çalışılmıştır. Ekran için gereksinimler lehtar ve alıcı bilgileri ile işlem tutarı ve tarihi bilgilerinden oluşmaktadır.

4.1. Yazılımın Analiz ve Tasarım Safhasında İGOT Modeli Yaklaşımı

Gereksinimleri belirleyen analistin verileri sisteme girmesi ile test senaryoları belirlenmeye başlar. Aşağıda uygulama ile ilgili birkaç gereksinim belirlenmiştir. Gereksinimler modelin ihtiyaçlara göre çoğaltılabilir. Modelin uygulandığı sistemde analiz dokümanı oluşturulmak istenildiğinde gereksinimler bir formata bağlanarak tanımlanabilir.

Ekran gereksinimleri için bileşenler Şekil 4.1.'de gösterilmiş ve aşağıdaki gibidir:

- Lehtar bilgileri için müşteri bileşeni
- Alıcı bilgileri için ad soyadı alanı ve IBAN alanı
- İşlem tutarı için sayısal veri alanı
- İşlem tarihi için tarih alanı



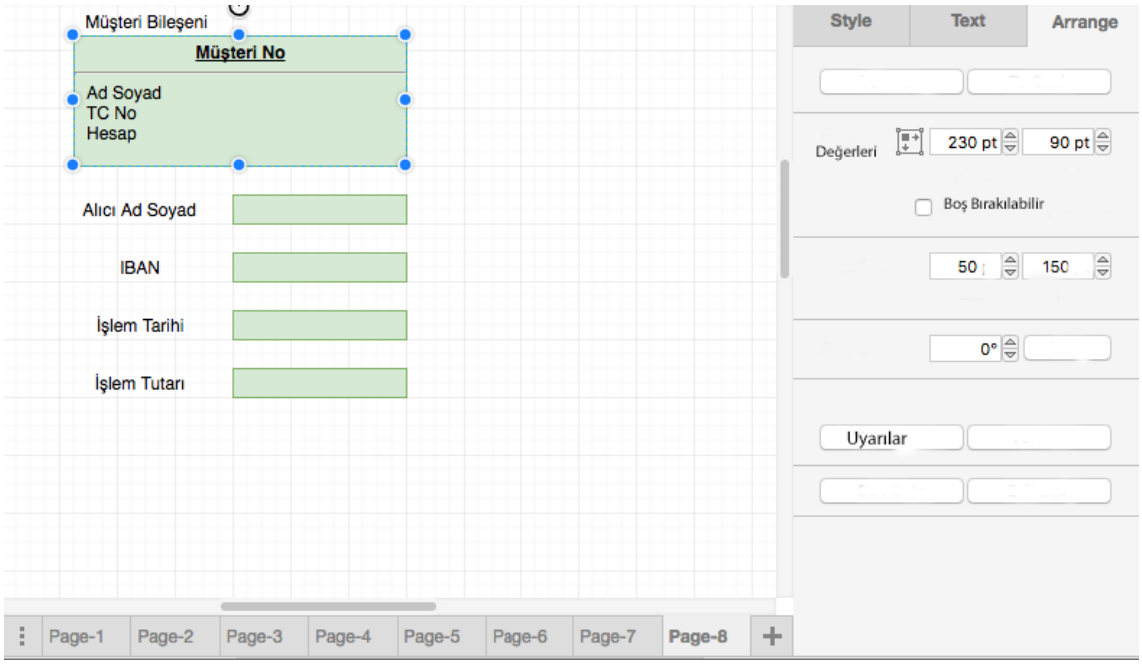
Şekil 4.1. Bileşenlerin Seçilmesiyle Ekran Tasarımı

Bileşen kataloğundan bileşenler seçilerek ekran tasarlanır. Bu kısımda prototip ekran tasarlanmış olup seçilen bileşenin tiplerine göre de basit düzeydeki test senaryoları gerçekleştirilmiş olur. Örneğin tarih bileşeni seçilerek alana sadece tarih verisinin girilmesi sağlanarak senaryolaştırılmış kodun oluşması için tasarım gerçekleştirilmiş olur.

Ekran gereksinimleri için bileşen özellikleri Şekil 4.2.'de gösterilmiş ve aşağıdaki gibidir:

- Müşteri bileşenine en az 6 haneli sayı girilmeli ve müşteri bilgileri gösterilmelidir.
- Ad soyad alanına sadece karakter girilebilmelidir.

- IBAN alanında sadece 2 harf ve 22 rakam girilebilmelidir ve IBAN kontrolü yapılmalıdır.
- İşlem tutarı alanına pozitif ve 10 ile 1000 arasında bir rakam girilebilmelidir.
- Tarih bileşeninde bugünden önce bir tarih seçilememeli ve sadece iş günleri seçilebilmelidir.



Şekil 4.2. Bileşenlerin özelliklerinin belirlenmesi

Bileşenler ekran üzerinden seçilerek her biri için özellikleri belirlenir. Alanın boş olup olamayacağı, sınır değerleri, hata vermesi durumundaki uyarı bilgilerinin katalogdan seçilmesi gibi bilgiler özellik olarak seçilir. Seçilen bu özellikler ekranın tamamlanması sonucunda senaryolara dönüştürülür. Örneğin işlem tutarı alanı için değerleri en az 10 ve en fazla 1000 olarak seçildiğinde;

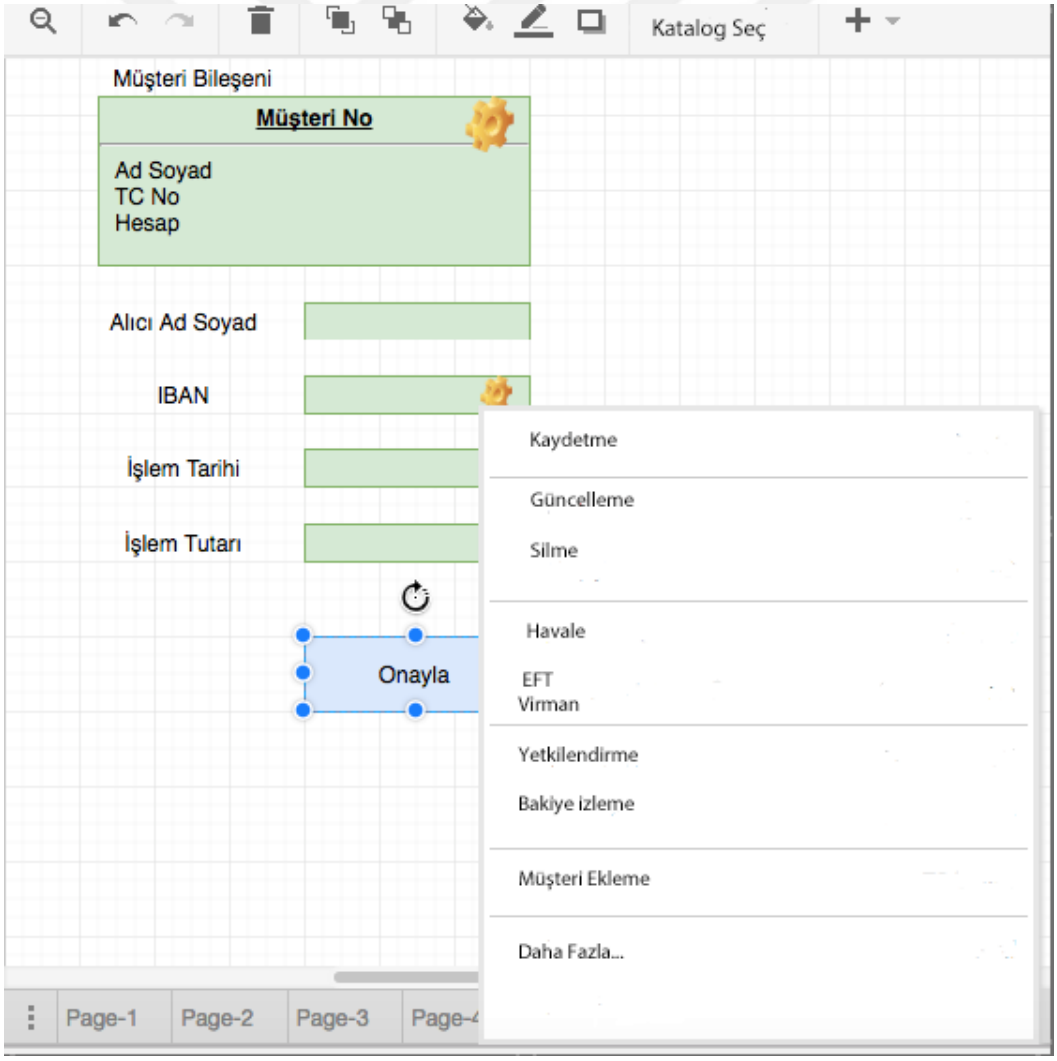
- Negatif rastgele bir sayının girildiği
- 10 ve 1000 sayılarının girildiği
- 10,05 reel sayının girildiği
- Karakter girildiği gibi test senaryoları oluşturulur.

Bu alanın rakamsal alan olduğunun belirtilmesi ile karakter kabul etmemesi test senaryosu senaryolaştırılmış kod olarak üretilmiş olur. 10 ve 1000 rakamsal sınır değerlerinin koşul olarak koda eklenerek kaynak kod oluşturulması gibi durumlarda

senaryolaştırılmış kodların oluşmasını sağlar. Özelliklerden senaryolaştırılmış kod olarak üretilebilecek olanları kaynak koda eklenerek üretilir. Sınır değerlerde bulunan değerler için test projelerinde birim testi senaryoları oluşturularak otomatik çalıştırılabilir test senaryoları oluşturulmuş olur. Alanın herhangi bir hatadan dolayı dolu gelmesi durumu da dokümante edilmiş test senaryosu olarak oluşturulabilir. Bu senaryolar sistemin test stratejisine göre çoğaltılabilir.

Ekran gereksinimleri için aksiyonlar Şekil 4.3.'de gösterilmiş ve aşağıdaki gibidir:

- Müşteri bilgisi getirme
- IBAN kontrol etme
- EFT Onay



Şekil 4.3. Aksiyonların Seçilmesi

Kaydetme, silme, güncelleme gibi aksiyonların işlem kataloğunda daha önceden tanımlandığı varsayılmıştı. Ayrıca bu üç işlemin farklı sıralar veya alanlarda farklı kombinasyonlar ile yapılarak yeni bir işlemin gerçekleşmesini sağlayan motor yapılarında bu işlem kataloğunda yer aldığı varsayılmıştır. EFT işlemi de böyle bir motordur ki müşteri hesabındaki paranın işlem tutarınca güncellenmesini ve diğer bankaya gönderilecek tutarın ve alıcının bilgisinin de diğer tabloya kaydedilmesini sağlar. EFT işlemi birçok platformda yada diğer modüllerce de kullanılabileceğinden motor olarak işlem kataloğunda yer alması verimliliği artırır.

Onayla aksiyonu ile EFT işlemi ilişkilendirilir. Böylelikle EFT işlemi için yapılacak fonksiyonlar özelliklerinde belirlenmesiyle test senaryosuna dönüştürülür. EFT işlemi için aşağıda belirtilen özellik dikkate alındığında EFT işlem kodu üretilmeden önce alanların dolu olması kontrol edilen senaryolaştırılmış kaynak kod oluşturulur, ardından EFT işleminin gerçekleştiğini kontrol eden test senaryosunu içerek test projesi oluşturulur.

Ekran gereksinimleri için aksiyon özellikleri aşağıdaki gibidir:

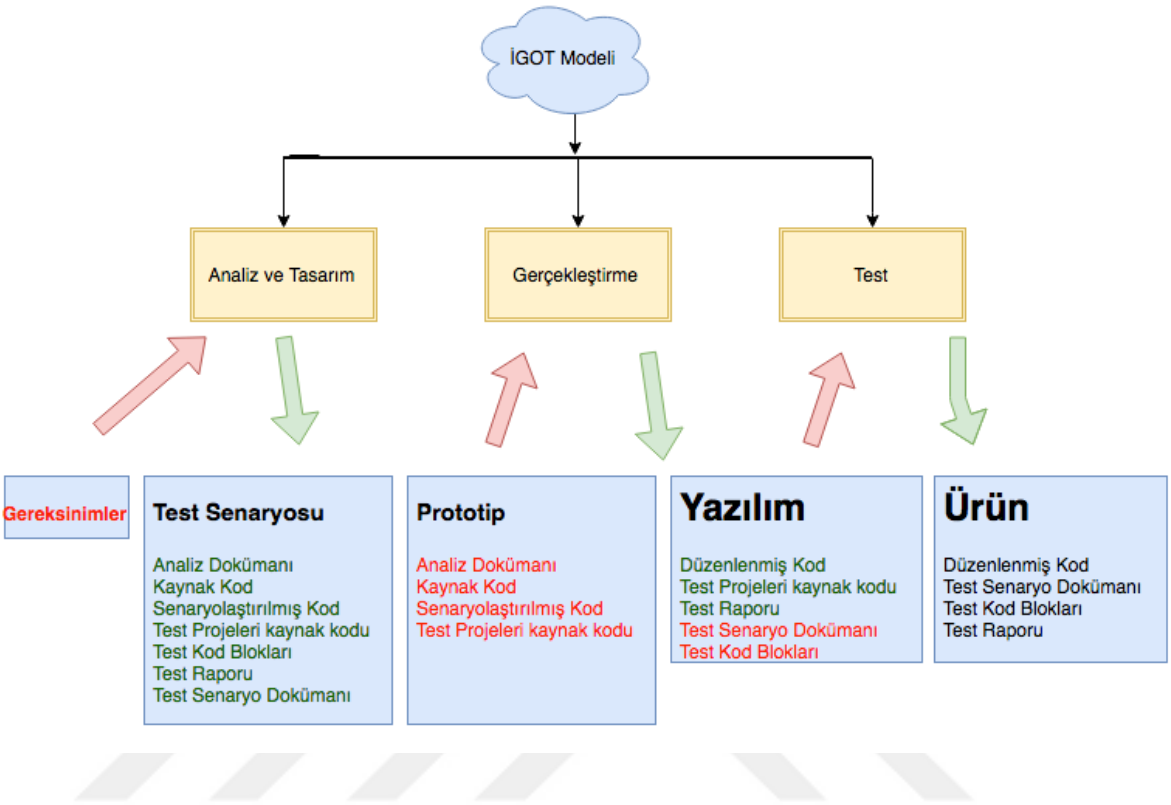
- Müşteri bilgisi getirmek için numara sorgulanmalı ve eşleşmelidir
- IBAN sorgulanması için tüm karakterler girilmelidir
- EFT onayı alınması için tüm alanlar doldurulmalıdır.

Belirtilen özellikler de aksiyon özellikleri ekranından seçilerek işlemlerin fonksiyonellikleri belirlenmiş olur. Böylelikle özelliklerde belirtildiği gibi müşteri bilgisinin getirilmesi aksiyonu bilgi getir işlemi ile ilişkilendirildikten sonra getirilen bilginin boş yada doğru olduğu senaryolaştırılmış kaynak kod olarak üretilir. Sonrasında gereksinime bağlı olarak bu şekilde birden fazla kaydın olması durumunu test eden test projesi üretilebilir. Ayrıca müşterinin hesabında para olmaması gibi test senaryoları da test senaryosu dokümanı olarak üretilebilir. Bu durum IBAN sorgulaması aksiyonu için de geçerlidir. İşlem kodunun eklenmesinden önce karakter kontrolü yapan bir senaryolaştırılmış kaynak kod eklenir.

Tüm aksiyonlar için İşlem kataloğundan seçilmiş işlemlerle ilgili olarak da fonksiyonlarını sağladıklarını kontrol eden test senaryosunu içeren test projeleri oluşturulur. Ayrıca senaryolaştırılmış kaynak kodların doğru çalışmasını kontrol eden test senaryolarını içeren test projeleri de oluşturulabilir.

Yazılımın analiz ve tasarım aşamasının sonunda, test otomasyonu eklentisinin olduğu varsayılarak, ekranın analist tarafından kullanımı sağlanarak test kod bloğunun oluşması sağlanır. Tüm bu işlemlerin ardından çıktı olarak analiz dokümanı, kaynak

kod, senaryolaştırılmış kod, test senaryolarını içeren test projesi, test kod bloğu test raporu ve test senaryosu dokümanı elde edilir. Şekil 4.4.'de gösterilmiştir.



Şekil 4.4. İGOT Modeli Çıktıları

4.2. Yazılımın Geliştirme Safhasında İGOT Modeli Yaklaşımı

Bu safhada yazılımcı ortaya çıkan analiz dokümanını, kaynak kodu, senaryolaştırılmış kaynak kodu ve test projeleri kaynak kodunu alarak yazılım projesini gerçekleştirir. Kaynak kodların projeye dönüştürülmesini sağlar. Düzenlemesi veya analizde belirtilen eklenmesi gereken gereksinimleri kod olarak ekler. Örneğin EFT işlemi sırasında kara listedeki kişilerin 500 TL tutarından fazla gönderememesi durumu için kod eklemesi yapabilir. Bu durumun senaryosu için de test projesine kod eklemesi yapabilir yada modeli uygulayan ekibin test politikalarına göre test uzmanına da bırakabilir. Yazılım kodlarının kaydedilmesi esnasında çek etme politikasınca yazılım kalite metrikleri de dikkate alınır.

Test senaryolarını içeren test projesini çalıştırarak birim testini gerçekleştirir. Sonuçlarına göre kod düzenlemesi yapılır. Birim testlerinin uygulamasını teknik test uzmanları da gerçekleştirebilirler. Teknik test uzmanları bu durumda kod okuyan bir

sistem geliştirerek kodlamada yer alan koşullar, döngüler gibi yapılar için test senaryosu dönüştüren xslt dönüşüm aracı geliştirebilirler. Böylelikle yazılımın içine eklenen sistem tarafından fark edilemeyen kod parçaları için de senaryoların oluşması sağlanmış olur.

Proje ekibince planlanan test stratejisine göre test otomasyonundan faydalanılarak düzenlenmiş kod üzerinde birim testi tekrarlanabilir. Test otomasyonu ile test kod bloğunu çalıştırarak kullanım testini de gerçekleştirebilir.

4.3. Yazılımın Test Safhasında İGOT Modeli Yaklaşımı

Test safhasında ortaya bir ürün öncesi yazılım çıkmış olur. İlk olarak test raporlarında senaryolaştırılmış kodlar, test senaryosu içeren test projeleri, oluşturulan test kod blokları, birim testi sonuçları ve test dokümantasyonu incelenir. Mevcut duruma bağlı olarak çalıştırılabilecek test senaryoları çalıştırılır. Test dokümantasyonu düzenlenir. Gerekli senaryo eklemeleri yapılabilir. Manuel testler gerçekleştirilir. Test kataloglarına eklenebilecek veriler eklenir. Hataların düzenlenmesi sağlanır. Hataların düzenlenmesi sonucu regresyon ve tekrar testleri gerçekleştirilir. Tekrar testleri sırasında test kod blokları kullanılır. Test sonuç raporu oluşturularak test kapanışı sağlanır.

Çizelge 4.1.'de test uzmanının incelemesi gereken Test Senaryo Raporu verilmiştir. Böylelikle oluşan test senaryolarının başarım durumu takip edilir.

Çizelge 4.1. Test Senaryo Raporu

Test Senaryosu	Gerçekleşmesi Gereken Durum	Oluşturma Alanı	Çalıştırılma Durumu
İşlem Tarihi geçmiş bir tarih seçilir.	Seçime izin verilmez.	Senaryolaştırılmış Kod	Başarılı
EFT işlemi sırasında bağlantı kopar.	Uyarı verilir ve ekran kapanır.	Test Dokümanı	Çalıştırılamadı
Kara listedeki müşteri için 520 TL tutar girilir.	İşlem yapılmaz uyarı verilir.	Test Projesi Kaynak Kodu	Test Uzmanı
IBAN alanına rastgele sayılar girilir.	Uyarı verilir.	Test Projesi Kaynak Kodu	Başarılı

Test yaşam döngüsünün bu süreci içinde test uzmanı test senaryo kataloğu ve FMEA risk kataloğu oluşturarak sistem özelliklerine bağlı otomatik test senaryosu

oluşturulan bir yapay zeka uygulaması ile de çalışabilir. Daha basit bir uygulama olarak da analiz ve tasarım safhasında sistem özellikleri ile test senaryosu kataloğundaki eşleşmeler sayesinde test senaryosu üretimi sağlanabilir. Örneğin mobil platformda EFT işlemi için gerekli güvenlik testi tasarım safhasında mobil güvenlik testi senaryosu ile ilişkilendirilerek test senaryolarının oluşturulması sağlanabilir.

Test senaryo katalogları oluşturulurken dinamik test senaryoları yazılmaya dikkat edilmelidir. Örneğin müşterinin sisteme dahil olma(login) işlemi sırasındaki kontrollerin test edildiği senaryolar için test kataloğu oluşturulduğunda web ve mobil gibi platformlar için ortak bir çok senaryo elde edilebilir.

4.4. Bakım ve Olası Gereksinim Değişikliği

EFT işleminin sonraki versiyon güncellemesinde tekrar test senaryoları oluşturulmaya gerek kalmayabilir yada daha küçük değişiklikler ile senaryolar düzenlenebilir. Bu durum değişikliğin etkisine bağlıdır. Örneğin müşterinin 10 000 TL ye kadar EFT yapabilmesi durumu güncellendiğinde ekran tasarımında tekrar aralık değeri 10 000 olacak şekilde seçilir kodu ve test senaryosu oluşturulur. Modelin yaşam döngüsü devam ettirilir.

Sadece kod olarak değişiklik yapılması durumunda kod dosyaları okutularak değişikliğe uğrayan kısımlar için birim testi senaryoları oluşturulur ve test uzmanı test dokümantasyonunu oluşturur.

İGOT modelinin uygulanması ile oluşturulan test senaryoları yazılım versiyonlarında tekrar test senaryosu üretme oranını azaltır sadece değişiklik yapılan kısımlar için test senaryosu üretimi sağlar. Değişiklik yapılmayan kısımlar için sistemde var olan test senaryoları test otomasyonun kullanılmasıyla da büyük ölçüde kaynak tasarrufu sağlar.

4.5. FMEA Uygulaması

Bankacılık sektöründe mobil platformda EFT işlemi için, sisteme giriş yapan kullanıcılar, banka bünyesindeki hesaplarından diğer bankaların hesaplarına para aktarımı yapabileceklerdir.

Gereksinimlerin sağlanması için EFT Bilgileri, Onay ve Güvenlik Doğrulama ekranları gerçeklenmiştir. Lehtar ve alıcı bilgileri, işlem tutarı ve güvenlik doğrulama bilgileri geliştirilen sistem tarafından kullanılabilir durumda bulunmaktadır.

FMEA süzgecinden geçmiş örnek bir Mobil EFT ekranlarına ait RÖS değerleri çizelge 4.2.'de gösterilmiştir.

Çizelge 4.2. FMEA ile olası tehlikelere olasılık, ağırlık ve saptama değerlerinin verilmesi

Olası Hatalar (EFT Yapılamama potansiyeli)	FMEA Süzgeci (Olasılık-Ağırlık-Saptama)				SONUÇ (Hatanın Oluşması-EFT Yapılamaması)
				ÖS	**EFT İşleminin Yapılamaması**
Alanların kayarak görüntülenememesi(Scroll olmaması durumu)				88	
Hesap numarası alanının yanlış eşleştirilmesi (Yanlış alan adına)				0	
Hesap numarası alanına veri girilememesi (Yanlış veri tipinde tanımlanması)				0	
Tutar bilgisinin virgüllü değişken tipiyle ayrılamaması				48	
İşlem sırasında ağ bağlantısının kopması	0	0		00	
Lehtar bilgilerinde başka müşterinin numarasının girilebilmesi		0		50	
Gönderim bilgilerinin ağ üzerinde değiştirilmesi	0	0		00	
Ekranda yer alan imajların yüklenememesi				2	
Numerik alanlara karakter girilmesi sebebiyle verinin dönüştürülememesi				8	
Güvenlik şifresinin kullanıcıya gönderilmemesi				0	

Olası hata etkilerinin, nedenlerinin ve mevcut kontrollerin belirlenmesi; Olasılık, ağırlık, saptama ve RÖS değerlerinin belirlenmesi (Çizelge 4.3.-4.4.-4.5.) RÖS' e göre hataların sıralanması, alınacak önlemlerin belirlenmesi (Çizelge 4.6) FMEA in en önemli başlıklarındandır.

Çizelge 4.3. Hatanın Ortaya Çıkma Sıklığı ve Derecesi (Olasılık-O) (Kahraman,Demirer,2010)

HATANIN OLUŞMA SIKLIĞI	HATANIN OLASILIĞI	DERECE
Çok Yüksek: Kaçınılmaz Hata	1/2 'den fazla	10
	1/3	9
Yüksek: Tekrar Tekrar Hata	1/8	8
	1/20	7
Orta: Ara Sıra Olan Hata	1/80	6
	1/400	5
Düşük: Nispeten Az Olan Hata	1/2000	4
	1/15000	3
Pek Az: Olası Olmayan Hata	1/150000	2
	1/150000' den düşük	1

Çizelge 4.4. Ağırlığın Etkisinin Sınıflandırılması (A) (Kahraman,Demirer,2010)

ETKİ	AĞIRLIĞIN (SIDDETIN) ETKISI	DERECE
Uyarısız Gelen Yüksek Tehlike	Felakete yol açabilecek etkiye sahip ve uyarısız gelen potansiyel hata	10
Uyarısız Gelen Tehlike	Yüksek hasara ve toplu ölümlere yol açabilecek etkiye sahip ve uyarısız gelen potansiyel hata	9
Çok Yüksek	Sistemin tamamen hasar görmesini sağlayan yıkıcı etkiye sahip ağır yaralanmalara, 3. derece yanık, akut ölüm vb. etkiye sahip hata türü	8
Yüksek	Ekipmanın tamamen hasar görmesine neden olan ve ölüme, zehirlenme, 3. derece yanık, akut ölüm vb. etkiye sahip hata türü	7
Orta	Sistemin performansını etkileyen, uzuv ve organ kaybı, ağır yaralanma, kanser vb. yol açan hata	6
Düşük	Kırık, kalıcı küçük iş görmezlik, 2. derece yanık, beyin sarsıntısı vb. etkiye sahip olan hata	5
Çok Düşük	İncinme, küçük kesik ve sıyrıklar, ezilmeler vb. hafif yaralanmalar ile kısa süreli rahatsızlıklara neden olan hata	4
Küçük	Sistemin çalışmasını yavaşlatan hata	3
Çok Küçük	Sistemin çalışmasında karşaşaya yol açan hata	2
Yok	Etki yok	1

Çizelge 4.5. Saptanabilirlik (S) (Kahraman,Demirer,2010)

SAPTANABİLİRLİK	SAPTANABİLİRLİK OLASILIĞI	DERECE
Fark Edilemez	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **mümkün değil**	10
Çok Az	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **çok uzak**	9
Az	Potansiyel hatanın nedeninin saptanabilirliği **uzak**	8
Çok Düşük	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **çok düşük**	7
Düşük	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **düşük**	6
Orta	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **orta**	5
Yüksek Ortalama	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **yüksek ortalama**	4
Yüksek	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **yüksek**	3
Çok Yüksek	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **çok yüksek**	2
Hemen Hemen Kesin	Potansiyel hatanın nedeninin ve takip eden hatanın saptanabilirliği **hemen hemen kesin**	1

Çizelge 4.6. Risk Öncelik Sayısı (RÖS) Değerlendirme(Kahraman,Demirer,2010)

RÖS DEĞERİ	ÖNLEM
RÖS <40	Önlem almaya gerek yok.
40≤RÖS≤100	Önlem alınabilir.
RÖS >100	Önlem alınması gereklidir.

4.5.1. FMEA Formları

Çizelge 4.7.-4.8.-4.9.'da yer alan FMEA formlarında örnek senaryolar yer almaktadır.

Çizelge 4.7. EFT Bilgileri için FMEA

Ref-No	Component	Fault	Cause	Failure Effect	Safety
1	Hesap Bilgileri	Hesap bulunamadı.	Hesap bilgileri arama alanına alfanumerik karakterler girilir.	Müşteri hesap numaraları numerik olduğundan eşleştirilemezler.	Bilgi getirilemez.
		Başka müşteriye ait hesaptan para gönderilir.	Paranın gönderileceği hesap bilgisi alanı sadece görüntülenebilir modda olmadığında başka hesap numarası girilir.	Başka müşteriye ait hesaptan para çekilir.	Müşteri bilgilerinin başkaları tarafından kullanılması nedeniyle %100 oranında güvenlik açığı oluşur.
2	Ekran Gridi	Alanlar görüntülenemez.	Ekrana scroll konulmaz, ekran hareketi sağlanmaz.	Alanlar görüntülenemediğinden veri girişi yapılamaz.	Bilgi eksikliği sebebiyle işlem gerçekleştirilemez.
3	Img file	Ekran stilini doğru görüntülenemez.	Dosya verileri büyüktür.	Veriler ekrana yüklenemez.	Ekran stilinin bozuk olması nedeniyle kullanım zorlaşır.

Çizelge 4.8. Onay Ekranı için FMEA

Ref-No	Component	Fault	Cause	Failure Effect	Safety
1	Hesap Bilgileri	Para seçilmeyen ek hesaptan gönderilir.	EFT bilgileri ekranında seçilen ek hesap bilgisi seçili gelmez.	Kullanıcının seçmediği farklı bir hesaptan para çıkışı yapılır.	Müşteri hesaptaki para akışını kontrol edemez.
2	Gönderilecek Para Miktarı	Gönderilen para miktarı yanlışdır.	Alan ile ilişkilendirilen değişken tipi farklıdır ve virgül ile ayrın yapılmamıştır.	Müşteri hesabından fazla para gönderimi gerçekleşir.	Müşteri verileri doğru olarak kullanılamadığından %100 tutarlılık hatası oluşturur.
3	Session Controller	Oturum sonlandığından işlem referansı oluşturulamaz.	İşlem doğrulama sırasında ağa bağlantı kesilir.	İşlemin gerçekleşmesi doğrulanamaz. Uygulama kapanır.	Veri tutarsızlığı oluşur.

Çizelge 4.9. Güvenlik Doğrulama için FMEA

Ref-No	Component	Fault	Cause	Failure Effect	Safety
1	SMS Güvenlik Şifresi	Güvenlik doğrulanamaz.	Müşteriye gönderilen kod ile kontrol edilen kod tutarsız girilir.	İşlem gerçekleştirilip sonlandırılamaz.	Müşteri hesabı blokelenir ve tutarsızlık açığı oluşur.
2	Criptografy processor	Bilgiler şifrelenemez.	Veriler parametre eksikliği ile gönderilirler.	Veriler şifrelenemez.	Veri gizliliği sağlanamaz.

Bu çalışmanın sonuçlarının değerlendirilmesinde en önemli ölçüt Risk Öncelik Sayısı (RÖS) olmuştur. Tek başına bakıldığında her bir faktörde yüksek çıkan değerler (O,A,S) olsa da önemli olan RÖS değerinin yüksekliğidir. Belirlenen değerlere ve adımlara göre önlemler alınarak süreç optimizasyonu daha kolay sağlandığı görülür.

FMEA de belirlenen adımların test edilmemesi sonucu gerçek ortamda 10 hata ile karşılaşılabilmektedir. Adımların uygulanması sonucu gerçek ortamda karşılaşılan hata sayısı 2 ye düşmüştür.

FMEA çalışması sonunda önerilen iyileştirmelerin kazaların engellenmesine ne oranda katkı sağladığını tespit edebilmek için dönemlik(yazılım projesi süresinde) hata sıklık ve ağırlık oranlarına bakmamız gereklidir. Dönemlik çalışma sonunda projede tespit edilen Kaza Sıklık Oranı(KSO) ve Kaza Ağırlık Oranı(KAO) ise aşağıda çıkarılmıştır. Burada kaza olarak belirtilen durum yazılım projesinin çalışmaması olarak

değerlendirilmelidir. Kahraman ve Demirer'in kullandığı formül için aşağıdaki uygulama gerçekleştirilmiştir.

Projede ortalama personel sayısı (iort) = 30

Oluşan Hata Sayısı (n) = 2

Kaybedilen işgünü sayısı = 1000

Bir projedeki işgünü sayısı = 100

Kayıp işgünü sayısı (Kaza ve meslek hast.) (k) = 30

Günlük çalışma süresini= 8 saat

$$KSO = \frac{n}{[(iort*100*8)-(1000*8)]} * 10000$$

$$KSO = \frac{2}{[(30*100*8)-(1000*8)]} * 1000000 = 125$$

$$KAO = \frac{k}{[(iort*100*8)-(1000*8)]} * 1000$$

$$KAO = \frac{30}{[(30*100*8)-(1000*8)]} * 1000 = 1,875$$

Çalışmanın sonunda ortaya çıkan veriler doğrultusunda FMEA test senaryo katalogları oluşturulabilir, belirli bir özelliğe dayanılarak adresleme(index) işlemi yapılabilir ve yazılımın yeni versiyonlarında veya benzer uygulamalarda da kullanımı sağlanabilir.

5. SONUÇLAR VE ÖNERİLER

5.1. Sonuçlar

Bu tezde analiz ve kaynak kod üretiminde iş gereksinimi odaklı test senaryolarını kazandırmanın nasıl olması gerektiğinden ve çok katmanlı yazılım mimarileri için sistem analisti, yazılımcı ve test mühendisleri pozisyonlarına sağladığı avantajlardan bahsedilmiştir. Önerilen bu sistem, analiz aşamasında tasarım ve dokümante kolaylığı sağlayarak, ortaya çıkabilecek analiz uygulama uyuşmazlığını da minimuma indirdiği gibi erken test aksiyonları ve senaryoların önceden oluşturulması ile hata oranlarının da azaldığı görülecektir.

Modelin büyük ölçekli kurumsal firmalarda veri katmanları, bileşen katalogları ve işlem katalogları oluşturarak kullanılması ayrıca bir standardizasyon ve beraberinde bakım kolaylığı sağlayacaktır. Ayrıca bu standardizasyon hataların tekrarlanmasını yada basit hataların üretimini de azaltacaktır. Standartlar ne kadar iyi belirlenirse ve kataloglar dinamik oluşturulursa oranla daha iyi verim alınacaktır.

Kurumun tüm uygulamalarını kapsayabilecek fonksiyonel bir değişiklik için işlem kataloğunda gerçekleştirilebilecek ufak bir değişiklik ile dinamik yapının sağlamış olduğu avantajlardan yararlanılabilir bağlı olduğu test senaryoları da güncellenerek test başarısı artırılabilir.

Analiz ve tasarımda belirtilmesi gereken gereksinimlerin, oluşturulması gereken test senaryolarının önemsenmeyerek atlanma riskinin azaldığı ve daha çok takip edilebilirliği olduğu görülecektir. Senaryolaştırılmış test kodlarıyla üretilebilecek hataların en baştan yok edildiği ve otomatize edilebilen senaryolar ile de bir çok kaynaktan tasarruf edildiği görülecektir.

Ortaya çıkan ürünlerden en önemlisi olan iş gereksinimlerini de içeren senaryolaştırılmış kaynak kodların, kaynak kod dizininde derlenip çalıştırılarak yazılımcının, test senaryolarını içeren kaynak kod ve test projelerinin çalıştırılarak hataların çözülmesiyle test uzmanının standartlaşmış gereksinimler için daha az ve efektif gereksinimler için daha verimli iş gücü harcamasına katkı sağlayarak oluşturulmasıdır.

5.2. Öneriler

İGOT modelinin özellikle tekrarlayan iş odaklarınca kullanılması tavsiye edilmektedir. Yazılım uzmanının test sonucunda tekrar kod yazması, sistem analistinin tanımlamayı unuttuğu birçok özellik ve test uzmanının senaryosunu yoğunluk, unutkanlık vb. durumlardan dolayı oluşturamayacağı senaryolar hepsi küçük yada büyük etkili olacak şekilde yazılım ürününde hatalar bırakmaktadır. Aynı alt yapı üzerine birçok modül geliştiren, yazılım ürünlerinin versiyonlarını üreten, büyük ölçekte benzer alt yapıları kullanarak farklı yazılım ürünleri üreten birimler bu modeli kullanabilirler.

Modelin tasarlanması sırasında kullanılacak teknolojiler üretim altyapısına uygun olmalıdır.

Birçok test tekniği kullanılarak modelin uygulaması daha zengin hale getirilebilir.

Modelin bilgi kaynaklarının kullanılarak yapay zeka entegrasyonunun sağlanması insan zekasıyla üretilebilecek test senaryolarının üretimi sağlanabilir.

Model dinamik, değiştirilebilir ve geliştirilebilir alt yapı üzerine kurulmalı ve gelişim süreci izlenmelidir.

KAYNAKLAR

Anonim, 2000, Hata Türü ve Etki Analizi , Dokuz Eylül Üniversitesi Sosyal Bilimler Enstitüsü Dergisi 2 (4): 136. 2000.

Anonim, 2011, *Test Otomasyon* [online], İstanbul, Wikipedia Özgür Ansiklopedi, https://tr.wikipedia.org/wiki/Test_otomasyon [Ziyaret Tarihi: 20 Ocak 2017].

Anonim, 2015, *Regresyon Testi Nedir* [online], Guru99, http://www.guru99.com/regression-testing.html [Ziyaret Tarihi: 31 Ocak 2017].

Anonim, 2016, *Hata Türleri ve Etkileri Analizi* [online], İstanbul, Wikipedia Özgür Ansiklopedi, https://tr.wikipedia.org/wiki/Hata_t%C3%BCrleri_ve_etkileri_analizi [Ziyaret Tarihi: 19 Ocak 2017].

Anonymous, 2010, *Application Life Cycle Management* [Online], United States, PPM Studio Solutions, http://www.ppmstudio.com/Solutions_ApplicationLifecycleManagement.aspx , [Ziyaret Tarihi: 31 Ocak 2017].

Balçiçek, Ö. E., Altıparmak, H. Ç., Tokgöz, B., 2013, Source Code Generation for Large Scale Application, *IEEE International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, Konya, 404 - 410 .

Balçiçek, Ö. E., Altıparmak, H. Ç., Tokgöz, B., 2014, İş Gereksinimi Odaklı Kaynak Kod Üretme Sistemi, *Akdemik Bilişim Konferansı,* Mersin,16.

Belatrix, 2015, *Functional Testing Best Practis* [Online], United States, Belatrix Software, http://www.belatrixsf.com/whitepapers/functional-testing-best-practices/ [Ziyaret Tarihi: 25 Ocak 2017].

Eriksson, U., 2015, *Fonksiyonel Testler ve Fonksiyonel Olmayan Testler* [Online], İsveç, ReQtest, http://reqtest.com/testing-blog/functional-vs-non-functional-testing/ [Ziyaret Tarihi: 25 Ocak 2017].

Erdemir, U., Tekin, U., Buzluca, F., 2008, Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi, *Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu.*

Gökalp, G., 2015, Entegrasyon (Integration) Testi Nedir ve Tipleri Nelerdir [online], İstanbul, Gökhan Gökalp, http://www.gokhan-gokalp.com/entegrasyon-integration-testi-nedir-ve-tipleri-nelerdir/ [Ziyaret Tarihi: 21 Ocak 2017].

IEEE, 1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE Standards Board.

IEEE Std 829™, 1998, IEEE Standard for Software Test Documentation, IEEE Std 829-1998.

International Software Testing Qualifications Board(ISTQB), Turkish Testing Board(TTB), 2014, Sertifikalı Test Uzmanı Temel Seviye Ders Programı, Turkce-ISTQB-Foundation-Level-Syllabus, *Yazılım Test ve Kalite Derneği*, Vol 2011.

ISO "ISO,IEC 9126",http://www.iso.org,2006

Kahraman Ö., Demirer A., 2010, OHSAS Kapsamında FMEA Uygulaması, *Teknolojik Araştırmalar, Makine Teknolojileri Elektronik Dergisi*, e-ISSN:1304- 4141, Cilt:7, No:1(5368)

Kılınç, D., 2013, *Yazılım Yaşam Döngüsü Temel Aşamaları* [online], İzmir, Yrd. Doç. Dr. Deniz KILINÇ Blog Sitesi, https://denizkilinc.com/2013/07/08/yazilim-yasam-dongusu-temel-asamalari-software-development-life-cycle-core-processes/ [Ziyaret Tarihi: 19 Ocak 2017].

Kızmaz, V.U., 2012, *Yazılım Yaşam Döngüsü Nedir* [online], Ankara, Veysel Uğur Kızmaz Blog, http://www.ugurkizmaz.com/YazilimMakale-1825-Yazilim-Yasam-Dongusu-Nedir.aspx [Ziyaret Tarihi: 14 Kasım 2016].

Küçük, A.S., 2016, *Yazılım Testi Metodolojileri* [online], İzmir, İztim Bilişim, http://www.iztim.com/Blog/YazilimTeknolojisi/YAZILIM-METODOLOJI [Ziyaret Tarihi: 14 Kasım 2016].

McCall,J.A.,Richards,P.K., Walters, G.F.,1997,Factors in Software Quality,*Nat'l Tech.Information Service*,no. Vol. 1,2 and 3.

Motlagh, E. S., 2012, A Review of Automatic Test Cases Generation, *International Journal of Computer Applications* (0975 – 8887), Volume 57– No.13.

Namenlos, 2011, FMEA [online], Wikipedia, http://de.wikipedia.org/wiki/FMEA [Ziyaret Tarihi: 19 Ocak 2017].

NASA Software Safety Guidebook, 2004, National Aeronautics and Space Administration, Vaşington, NASA-GB-8719.13.

Önder, 2013, *Bakım Testi* [online], Yazılım Testi Blog, http://yazilimitestedelim.blogspot.com.tr/2013/11/bakm-testi-maintenance-testing-1.html [Ziyaret Tarihi: 28 Ocak 2017].

Özçelik, O., 2015, Emniyet Kritik Yazılım Test Edilebilirliğinin İyileştirilmesi, Yüksek Lisans Tezi, *İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü*, İstanbul, 10-15.

Pekşen, Ö., 2016, *Hata Türleri ve Etkileri Analizi* [online], İstanbul, Wikipedia Özgür Ansiklopedi, https://tr.wikipedia.org/w/index.php?title=Dosya:Hatat%C3%BCrleriveetkilerianalizigrafi%C4%9Fi.png&filetimestamp=20111229184654&, [Ziyaret Tarihi: 31 Ocak 2017].

Rajan, A., 2006, Automated requirements-based test case generation, ACM SIGSOFT Software Engineering Notes, New York, USA, Volume 31 Issue 6,Pages 1-2.

Seker, S.E., 2015, Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü, YBS Ansiklopedi, İstanbul Medeniyet Üniversitesi, Cilt 2, Sayı 3.

STC, 2013, *Yazılım Testi Yaşam Döngüsü STLC* [online], Software Test Class, http://www.softwaretestingclass.com/software-testing-life-cycle-stlc/ [Ziyaret Tarihi: 14 Kasım 2016].

Tozlu, S., 2010, *Yazılım Test Aşamaları* [online], Amerika, Seçkin Tozlu, http://www.seckintozlu.com/414-yazilim-testi-nedir.html [Ziyaret Tarihi: 21 Ocak 2017].

Yener O., 2015, *Software Testing Life Cycle* [online], İstanbul, Linkedin, https://www.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-test-s%C3%BCre%C3%A7leri-stlc-software-testing-life-cycle-orhan-yener [Ziyaret Tarihi: 19 Ocak 2017].

Yıldız, G., 2014, *Yazılım Test Otomasyonu* [online], İstanbul, Gokyhome Blog, https://gokyhome.com/2014/07/22/yazilim-test-otomasyonu/ [Ziyaret Tarihi: 13 Aralık 2016].

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Büşra ÇÖLLÜ
Uyruğu : Türkiye Cumhuriyeti
Doğum Yeri ve Tarihi : Selçuklu, 1992
Telefon : 05535432994
Faks : yok
e-mail : Busratokgoz32@gmail.com

EĞİTİM

Derece		Adı, İlçe, İl	Bitirme Yılı
Lise	:	Yalvaç Anadolu Lisesi, Yalvaç, Isparta	2009
Üniversite	:	Selçuk Üniversitesi, Selçuklu, Konya	2013
Yüksek Lisans	:	Necmettin Erbakan Üniversitesi, Meram, Konya	

İŞ DENEYİMLERİ

Yıl	Kurum	Görevi
2012-2013	KuveytTürk Katılım Bankası AŞ	Yazılım Mühendisi
2013-2015	KuveytTürk Katılım Bankası AŞ	Test Mühendisi
2015-2016	KuveytTürk Katılım Bankası AŞ	Sistem Analisti

YABANCI DİLLER

İngilizce

YAYINLAR

1. "Source Code Generation for Large Scale Application", IEEE International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 404 - 410 (2013).
2. "İş Gereksinimi Odaklı Kaynak Kod Üretme Sistemi", Akdemik Bilişim, Mersin,2014

MESLEKİ SERTİFİKA VE EĞİTİMLER

1. ISTQB Foundation Level Uluslararası Yazılım Test Uzmanı Sertifikası (25.10.2014)
2. ISTQB Advance Level Test Yöneticisi Eğitimi (22.08.2015)
3. User Experience Design Eğitimi (29.10.2013)