EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

TEST YÖNTEMLERİ İLE YAZILIMIN KALİTESİNİN BELİRLENMESİ VE İYİLEŞTİRİLMESİ

Gül DELİORMAN

Tez Danışmanı: Prof. Dr. Aylin KANTARCI

Bilgisayar Mühendisliği Anabilim Dalı

Sunuş Tarihi: 26.10.2015

Bornova-İZMİR

2015

Gül DELİORMAN tarafından **Yüksek Lisans** tezi olarak sunulan **"TEST YÖNTEMLERİ İLE YAZILIMIN KALİTESİNİN BELİRLENMESİ VE İYİLEŞTİRİLMESİ"** başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Ensittüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 26.10.2015 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza:**
Jüri Başkanı	**: Prof. Dr. Aylin KANTARCI**	**.....**
Raportör Üye	**: Prof Dr. Yasemin TOPALOĞLU**	**.....**
Üye	**: Doç. Dr. Ayşegül ALAYBEYOĞLU**	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarında Yüksek Lisans Tezi / Doktora Tezi olarak sunduğum **"TEST YÖNTEMLERİ İLE YAZILIMIN KALİTESİNİN BELİRLENMESİ VE İYİLEŞTİRİLMESİ"** başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilemeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

26/10/2015

İmzası

Adı – Soyadı

ÖZET

TEST YÖNTEMLERİ İLE YAZILIM KALİTESİNİN BELİRLENMESİ VE İYİLEŞTİRİLMESİ

DELİORMAN, Gül

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Aylin KANTARCI

Ekim 2015, 42 sayfa

Her geçen gün artarak artan yazılım geliştirme ihtiyacı tüm sektörlerde olmazsa olmaz gereksinimler arasında yerini almıştır. Bu ihtiyaçlara cevap verebilmek için çeşitli yazılım geliştirme yaşam döngüsü modelleri uygulanmakta ve gereksinimlere uygun model seçilerek geliştirim amacı ile projelere başlanmaktadır. Zaman kısıtı, maliyet kısıtı gibi çeşitli etkenlerden dolayı proje adımları içerisinde test adımına gereken önem verilememektedir. Bu durum kalite yoksunu yazılımların gerçek ortamda kullanılmasına ve hataların kullanıcı deneyimleri ile tespit edilmesine neden olmaktadır. Kullanıcı deneyimleri ile tespit edilen hatalar, önemli ölçüde para kaybına neden olabildiği gibi; projenin üretildiği sektöre bağlı olarak can kaybına dahi neden olabilmektedir.

Bu tez projesinde, test sürecinin yazılım geliştirme modelleri içerisindeki yeri ve uygulanmaya başlanması ile yazılımlardaki kalite artışının gözlemlenmesinin sağlanmasıdır ve test yöntemleri ile yazılımın kalitesinin belirlenmesi ve iyileştirilmesi amacıyla gerçekleştirilmiştir. Tez projesi için sağlık sektöründen bir uygulama seçilmiş, proje geliştirme yaşam döngüsünün baştan sonra test bakış açısı ile irdelenebilmesi için yazılım baştan geliştirilmiştir. Tez projesi içinde, karaciğer nakli ve sonrasındaki büyümenin tespiti için hacim ölçümü yapan bir uygulama gekiştirilmiştir. Testler için var olan bir yazılım kullanılmamış olup, tez projesi aynı zamanda sektörel bir ihtiyaca cevap veren kaliteli bir yazılım niteliğindedir.

Tez projesinde sağlık sektöründeki geliştirilen yazılım özelinde bazı tespitlerde bulunulmuş olsa da, tüm sektörlere ait yazılımların, kaliteli geliştirilmediği takdirde bazı risklere açık bulunduğu açıktır ve tez projesi kapsamında bu durumun test yöntemleri ile iyileştirilerek riskler en aza indirilebileceği anlatılmıştır.

Anahtar Sözcükler: Yazılımların Testi, Test Mühendisliği, Yazılım Mühendisliği, Yazılımda Kalite, Yazılımda Risk

ABSTRACT

DEFINE AND IMPROVE SOFTWARE QUALITY WITH TEST METHODOLOGIES

DELİORMAN, Gül

MSc in Computer Engineering

Supervisor: Prof. Dr. Aylin KANTARCI

October 2015, 42 pages

Software development needs from every industry becomes essential with increasingly growing expectations everyday. In order to response these demands form each different industry, suitable software development life cycle model shoud be selected at the begining of project thus this model must be appropriate for practicing requirements. Necessity of testing a software product may be ignored during milestones of projects due to limited time and budget. İgnoring essential testing process decreases quality of software where failures are detected by end user expriences. Software failures detected by end users with their own expriences causes financial loss, even loss of life according to industry of project such as safety critical systems.

Scope of this project focuses on importance of testing process within software development life cycle models , methodologies of testing which determines overall quality of software and observation of increasing quality of software after testing. For examination of sotware development life cylce from testing point of view, software from health sector is selected then its redeveloped . In this thesis a software is developed which measures growth of liver after a liver transplant surgery by means of its volume. There isn't any automated or predeveloped software testing tool is used. This thesis is also a kind of high quality software which may satisfy needs of health sector.

Although some observations are made for health sector's software, it's clearly certain that,software from other industries which have poor software qualty are vulnerable and risky. As a conclusion, testing of a software with correct methodologie minimizes risks of software product is covered in this thesis.

Keywords: Software Testing, Test Engineering, Software Engineering, Software Quality, Software Risk

TEŞEKKÜR

Eğitim hayatım ve tez proje çalışma sürecim boyunca verdiği destekten ve üzerimdeki emeklerinden dolayı değerli hocam Prof. Dr. Aylin Kantarcı'ya, çalışma sürecinde manevi desteğini esirgemeyen çok sevgili aileme teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ÖZET vii

ABSTRACT ix

TEŞEKKÜR xi

ŞEKİLLER DİZİNİ xv

ÇİZELGELER DİZİNİ xvii

SİMGELER DİZİNİ xix

KISALTMALAR DİZİNİ.....xix

1. GİRİŞ.....1

2. YAZILIMLARDA KALİTE VE YAZILIMLARIN TEST EDİLMESİ.....3

2.1 Yazılımların Ölçülmesi ve Kalite.....3

2.2 Yazılımların Testinin Avantajları.....4

2.3 Tipik Bir Test Süreci ve Test Türleri.....6

3. TEZ KAPSAMINDA GELİŞTİRİLEN YAZILIMDA KULLANILAN YAZILIM GELİŞTİRİM METOTLARI ve TEST TÜRLERİ.....9

3.1 Artırımlı Yazılım Geliştirme.....9

3.2 Gereksinim Tabanlı Yaşam Döngüsü.....10

3.3 Tez Kapsamında Kullanılan Test Türleri.....11

3.3.1 Birim testi.....11

3.3.2 Bileşen testi.....13

3.3.3 Duman testi.....15

İÇİNDEKİLER(devam)

Sayfa

3.3.4 Sistem testi (Uçtan uca test).....15

3.3.5 İlerleme testi.....16

3.3.6 Kullanılabilirlik testi.....16

4. VAKA ÖRNEĞİ: KARACİĞER VOLUMETRİ YAZILIMI TESTİ.....18

4.1 Gereksinimlerin Belirlenmesi ve Tasarım.....18

4.2 Gerçekleştirim Süreci.....20

4.2.1 Tasarım, geliştirim ve duman testleri – Versiyon 1.....20

4.2.2 Tasarım, geliştirim ve duman testleri – Versiyon 2.....22

4.3 Test Süreci.....24

4.3.1 Duman testleri.....31

4.3.2 Bileşen testleri.....32

4.3.3 Sistem testleri.....33

4.3.4 İlerleme (Regresyon) testleri.....34

4.3.5 Kullanılabilirlik testleri.....34

5. GENEL SONUÇLAR ve TARTIŞMA.....38

KAYNAKLAR DİZİNİ.....40

ÖZGEÇMİŞ.....42

EKLER.....

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

2.1 Test Sayısı ve Kalite Arasındaki İlişki (Patton, 2006).....5

3.1 İlk Örnek(Prototip) Yaşam Döngüsü (Jorgensen, 2013).....10

3.2 Test Seviyeleri ve Yazılım Yaşam Döngüsü İlişkisi.....11

3.3 Bileşen Testi Sürücü ve Koçan Gösterimi (ISTQB, 2015).....14

4.1 Gerçekleştirim Süreci.....20

4.2 Karaciğer Üzerinden Alınan Kesitler.....23

4.3 Yazılıma Ait Durum Geçiş Diyagramı.....25

4.4 Duman Testi için Seçilen Yol.....31

4.5 Resim Açılma Hızlarının Karşılaştırılması.....33

4.6 Yazılıma Ait İlk Ekran Görüntüsü.....35

4.7 Yazılıma Ait Güncellenmiş Ekran Görüntüsü.....36

4.8 Yazılıma Ait Güncellenmiş Ekran Görüntüsü – 2.....37

4.9 Yazılıma Ait Son Ekran Görüntüsü.....37

ÇİZELGELER DİZİNİ

Şekil **Sayfa**

2.1 Yıllara göre proje değerlendirmeleri (Atagören, 2012).....3

2.2 Yıllara göre proje değerlendirmeleri 2 (Standish , 2013).....4

3.1 Birim Testlere Genel Bakış.....13

4.1 Olası Geçişler ve Gerekli Eylemler.....26

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
Kısaltmalar	**Açıklama**
CMM	Capabiliy Maturity Model
CT	Computed Tomography
DICOM	Digital Imaging and communications in Medicine
GUI	Graphical User Interface
IEC	International Electrotechnical Commission
IEEE	The Institude of Electrical and Electronics Engineers
ISEB	Intermediate Enterprise and Solution Architecture
ISO	Information Systems Examinations Board
ISTQB	International Software Testing Qualifications Board
NEMA	National Electrical Manufacturers Association
ROI	Region of Interest
TS	Türk Standartları
TSE	Türk Standartları Enstitüsü
QoS	Quality of Service

1. GİRİŞ

Yazılım günümüzde tüm sektörlerde önemli bir ihtiyaç haline gelmiştir. Yazılım geliştirme yaşam döngüsü içerisinde yer alan "test", gelişen ihtiyaçlar, rekabet, maliyet gibi birçok sebepten dolayı gün geçtikçe artan bir öneme sahiptir.

Test, yazılım yaşam döngüsü içerisinde ürünün kalitesini artıran ve müşteri memnuniyetini sağlayan bir adımdır. Yazılım test sürecinin verimli geçmesi, analiz, tasarım ve yazılım geliştirme aşamalarında var olan hataların en az maliyetlerle düzeltilmesi adımını içerdiğinden, bu adımın başarısı, doğrudan proje başarısını etkilemektedir. Yapılan araştırmalara göre, 2000 yılından itibaren yazılım kalite güvence tekniklerinden en geniş oranda kullanılan yaklaşım testtir(Orso et al., 2014).

Akademik anlamda test üzerinde binlerce çalışma, araştırma yapılmakta ve yeni yöntemler geliştirilmektedir. Test üzerinde bu kadar fazla araştırmalar yapılmasının nedeni, yazılım dünyasının hayal gücümüzü sınırlayacak boyutlara ulaşmış olmasıdır. Günümüzde tek bir yazılım geliştirme dili kullanılan, homojen, dağıtık olmayan, küçük ya da orta ölçekli yazılımlara rastlamak neredeyse imkânsızdır. Artık yazılım geliştirirken geleneksel yöntemlere eskisi kadar yaklaşılmamaktadır. Günümüzdeki yazılımlar, çeşitlilik içeren birimlerden, dinamik, dağıtık ve birbirlerine entegre olarak yani heterojen olarak çalışmaktadırlar. Mobil uygulamalardan, bulut tabanlı uygulamalara, web uygulamalardan, yazılım ürünlerine ve servis temelli mimari yazılımlarına kadar geniş bir alanda yazılım geliştirilmektedir. Böylesine karmaşık ve rekabet ortamı içerisinde yazılımın kalitesi ve kalitenin testler aracılığı ile belirlenip geliştirilmesi oldukça önemli bir yer tutar hale gelmiş ve bu noktada akademik araştırmalar önem kazanmıştır (Orso et al., 2014).

Zaman içerisinde test yapmadan direkt olarak gerçek ortama aktarılan yazılımların çok büyük mali zararlara, mal ve hatta can kayıplarına dahi neden olduğu gözlemlenmiştir. Çalışılan proje sektörüne göre, bulunan hataların doğurduğu sonuçlar değişebilmektedir. Örneğin muhasebe ya da banka ile ilgli yazılımlarda yapılacak hatalar önemli miktarlarda para kaybına sebep olabilecekken, taşıma araçlarına ait yazılımlarda yapılacak hatalar(örn. Uçak, tren vb.) mal ve can kaybına bile neden olabilmektedir.

Dünyada, yazılım testlerinin gerekliliğine olan inanç yıllar önce geliştirilmiş ve bu doğrultuda 1998 yılında sertifikalı yazılım test mühendisi yetiştirmek üzere İngiliz Bilgisayar Topluluğu(British Computer Society, ISEB) tarafından harekete geçilmiştir. Bu hareket, ISTQB kurumunun kurulmasında temeli oluşturmuştur ve 2002 yılında Avusturya, Danimarka, Finlandiya, Almanya, İsveç, İsviçre, Hollanda ve İngiltere'nin katılımıyla kuruluş gerçekleşmiştir. 2004 yılında, yapılan sınav doğrultusunda, ilk kez sertifikalı ileri seviye test mühendisi sertifikaları verilmiştir. 2011 yılı sonunda 200.000'den fazla kişi, 2013 sonunda ise 300.000'den fazla kişi sertifikalı test mühendisi ünvanına kavuşmuştur(ISTQB, 2015).

Yazılım testlerine olan bakış açısı, Türkiye'de dünyadaki kadar hızlı bir biçimde gelişememiştir. Şirketler yukarıdaki örneklerde verilen durumları yaşadıkça, testin gerekliliğini idrak etmeye başlamıştır ve Türkiye'de de birçok şirkette test uzmanı, test mimarı, test analisti, test yöneticisi gibi pozisyonlar oluşmuş ve organizasyonel değişikliklere gidilmiştir.

Türkiye'de 2006 yılında ISTQB altında TTB(Turkish Testing Board) organizasyonu kurulmuştur ve bu tarihten itibaren de sektörün yazılım kalitesi konusunda bilgilendirilmesi, ISTQB sınavlarının yapılması ve adayların sertifikasyonu, standart haline gelmiş terimlerin Türkçeleştirilmesi gibi temel görev ve sorumlulukları yerine getirmektedir(Turkish Testing Board, 2015).

Dünyadaki ve Türkiye'deki bu gelişmelere ve test mühendisliği konusunda yapılan araştırmalara bakıldığında; test mühendisliğinin yükselen bir değer olduğu görülebilir. Bilişim sektöründe testin yeri hem akademik hem de endüstriyel anlamda her geçen gün sağlamlaşmaktadır. Yapılan çalışmalar oldukça ilgi çekicidir.

Bu tez çalışmasının amacı, tıp sektöründe geliştirilmiş bir yazılımın testinden yola çıkarak, yazılım testlerinin önemini vurgulamak ve bilgisayar mühendisliği eğitim müfredatına eklenmesi için öneride bulunmaktır.

Bu bağlamda, ikinci bölümde "Yazılımlarda Kalite ve Yazılımların Test Edilmesi" başlığı incelenecek, dünyadaki çalışmalardan önemli sonuçlar paylaşılacak ve kalite kavramı üzerinde durularak, yazılım testlerinin avantajlarından bahsedilerek tipik test süreci ve test türleri açıklanacaktır. Üçüncü bölümde, tez kapsamında geliştirilen yazılımda kullanılan yazılım geliştirim metotları ve test türleri irdelenecektir. Dördüncü bölümde ise, tez kapsamındaki vaka örneği olan "Karaciğer Volumetri Yazılımı"na ait süreçlerden bahsedilerek test süreci irdelenecektir. Beşinci bölümde genel sonuçlar ve tartışma başlığı altında yazılım testleri ile ilgili saptamalar ve önerilerde bulunulacaktır.

2. YAZILIMLARDA KALİTE VE YAZILIMLARIN TEST EDİLMESİ

2.1 Yazılımların Ölçülmesi ve Kalite

Yazılımların birçoğu istenilen verim elde edilemediği, teslim edilemediği, bitiminden sonra da değişikliklerin yapılması gerekliliği gibi nedenlerden dolayı rafa kaldırılmıştır.

Standish grubu, projelerin başarılı olup olmadığı konusunda araştırmalar yapan bir gruptur. Standish grubunun 1994 ve 2004 yılları arasındaki çalışmalarına göre proje değerlendirmeleri aşağıdaki Tablo 2.1'de bulunmaktadır(Atagören,2012):

Yıl	Başarı %	Başarısız %	İptal Edilen %
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18

Tablo 2.1- Yıllara göre proje değerlendirmeleri (Atagören, 2012)

2004 yılından sonraki dönemde yazılımların başarı ve başarısızlık oranları ise Tablo 2.2'de gösterilmiştir(Standish, 2013):

Yıl	Başarı %	Başarısız %	İptal Edilen %
2006	35	19	46
2008	32	24	44
2010	37	21	42
2012	39	18	43

Tablo 2.2 - Yıllara göre proje değerlendirmeleri 2 (Standish , 2013)

Tablo 2.1 ve Tablo 2.2 göz önüne alındığında, yıllar içerisinde projelerdeki başarı oranının arttığı görülmektedir. 1994 yılından 2012 yılına kadar geçen sürede %16 olan başarı oranı %39'lara kadar çıkarılmıştır. Başarıdaki bu artışın nedenleri yazılım mühendisliğindeki ilerlemeler ve yazılım kalite kontrolünün öneminin anlaşılması suretiyle yazılım testi pratiklerinin yaygınlaşmaya başlamasıdır(Standish, 2013).

Yazılımın kalitesi çeşitli bakış açılarıyla değerlendirilebilmektedir. Müşteri gözüyle bakıldığında, istenilenlerin yerine getirilmiş olması ve ara yüzlerin kullanışlı olması kaliteli anlamına gelebilirken; özel sektör hızlı çözüm getiren uygulamaları daha kaliteli bulmaktadır. Çünkü bu defa ön plana çıkan konu rekabettir.

Yazılımın kalitesinin değerlendirilmesi yukarıda bahsedilen öznel yaklaşımların yanı sıra bazı standartlar ile belirtilmiştir. Bu standartlar: ISO/IEC 15939 Yazılım Ölçüm Süreci, CMM, IEEE 1061, ISO 9000 Standartları(12207- Yazılım ömür devri süreçleri, 15288- Sistem Ömür Devri Süreçleri, 9126- Yazılım ürünü kalitesi, 14598- Yazılım ürünü değerlendirmesi, ISO 15504(SPICE)). Bu standartlardan; *ISO/IEC 15939 Yazılım Ölçüm Süreci* yinelemeli ölçüm adımlarından oluşuyorken, *CMM* yazılımların belirlenen çeşitli olgunluk aşamaları ile denetlenmesini sağlayan bir model olarak görülmektedir. IEEE 1061 yazılım kalitesi ile ilgili ölçütler tanımlamakta ve ISO 9000 tüm aşamaları kapsayan ve sürekli geişmeyi hedefleyen bir model olarak sayılmaktadır (ISO, 2007).

2.2 Yazılımların Testinin Avantajları

Genel itibariyle, yazılımın kalitesini belirleyen temel özellikler arasında: Doğruluk, bütünlük, kullanılabilirlik, çok amaçlı kullanılabilirlik, tekrar kullanılabilirlik, test edilebilirlik, kapasite, performans, performansın kontrolü, işletim sürekliliği, güvenilirlik, güvenlik, uyarlanabilirlik, hassaslık, belgeleme, bakım kolaylığı, transfer kolaylığı, geliştirme kolaylığı, raporlama yetenekleri sayılabilmektedir (Berztiss, 1996; Gorton, 2011).

Bu metrikler doğrultusunda yazılımın ölçülmesi ve beklenen gereksinimlerin sağlanıp sağlanmadığının kontrolü yapılmalı ve eğer gereksinimler sağlanmıyorsa niçin sağlanamadığının belirlenmesi gerekmektedir. IEEE Software Engineering Body of Knowledge'a göre (SWEBOK, 2004); "Yazılım testleri, bir programa ait davranışın, sonsuz durum kümelerinden seçilmiş, sonlu test senaryoları eşliğinde ve beklenen davranış karşısında dinamik olarak doğrulanmasıdır."

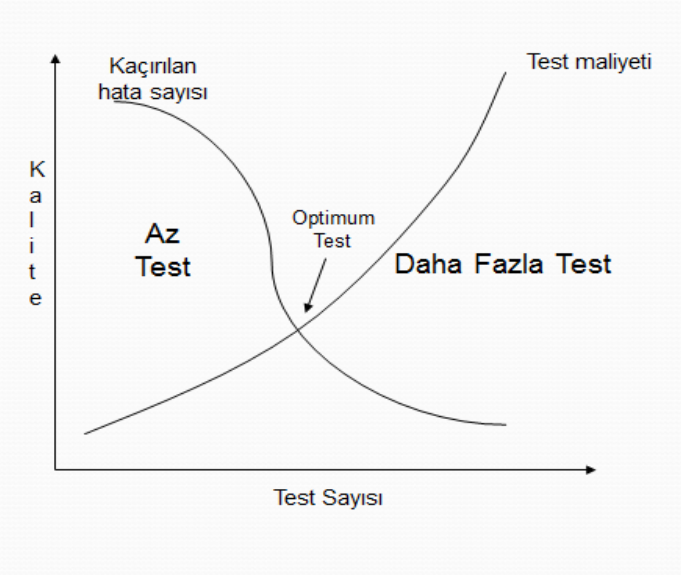
Yazılım testlerinin aşağıdaki konularda önemli getirileri bulunmaktadır:

1. **Eksiklerin ve Hataların Saptanması:** İnsanların yaradılışı gereği hata yapmaya meyillidirler. Yazılım geliştiricilerin çalışma şartları düşünüldüğünde, mesai saatleri boyunca sürekli düşünerek ve kod geliştirerek yazılımın gereksinimlerini yerine getirmeye çalışmaktadırlar. Mesai saatleri içerisinde her dakika boyunca aynı dikkat ve motivasyona sahip olmak mümkün değildir. Dolayısıyla insanlar hata yapabilirler. Bu hatalar çok önemsiz ve küçük hatalar olabilirken, önemli mantık hataları vs. de olabilmektedir. Test ise farklı bir bakış açısı sağlayarak, var olan hataları bulup çıkarmaya ve yazılımın kalitesini artırmaya yönelik bir süreç olarak ortaya çıkmaktadır.

2. **Müşterinin Güven ve Memnuniyeti:** Yazılımların test edilmesinin, yazılımın kalitesini artırdığı gerçeğinin yanı sıra, test edilmiş yazılımlar müşterilerde yazılımın daha güvenilir olduğu algısını yaratmaktadır. Test edilmiş yazılımlar, yazılımın kontrollü ve kaliteli bir aşamadan geçerek geliştirildiğinin bir işaretidir.

3. **Maliyet:** Testten geçen yazılımlar, daha sonraki aşamalarda fark edilen hataların çözümü düşünüldüğünde çok daha ucuza mal olmaktadır. Testten geçen yazılımların bakım maliyetleri de görece daha az olmaktadır. Maliyet, güven faktörü ile birlikte müşterilerin en çok önem verdikleri faktördür. Yazılım ne kadar kaliteli ve maliyeti az olursa, o oranda müşteriler tarafından tercih edilebilir olmaktadır.

Şekil 2.1'de test sayısı ve kalite arasındaki ilişki, bulunan hatalar ve maliyetler bakımından değerlendirilmektedir.



Şekil 2.1 - Test Sayısı ve Kalite Arasındaki İlişki(Patton, 2006)

Şekil 2.1 incelendiğinde, yazılım yaşam döngüsünde test ağırlığı arttıkça daha çok hatanın yakalandığı, buna karşılık maliyetlerin arttığı görülmektedit.

Dolayısıyla, testler optimum seviyede tutulabilirse, hem kalite hem de maliyet konusunda orta yol bulunmuş olur.

Yukarıda bahsedilen maddeler göz önünde bulundurulduğunda, yazılımı test eden kişinin vizyonu ve alabileceği inisiyatifler önem kazanmaktadır. Yazılım test eden kişi, test edeceği tekniği yazılımın tipine ve yazılım yaşam döngüsüne uygun seçmeli, senaryo seçiminde dikkatli olmalı ve tecrübeleri doğrultusunda yazılımda hata çıkabilecek yerleri bilerek ve buralardan şüphelenerek test aşamasını ilerletmelidir. Bütün bunlarla birlikte, yazımı geliştiricilerin de hata çıkabilme olasılığını kabullenerek test gerçekleştiren kişilerle birlikte çalışmalıdır. Testler sırasında, çıkan hatayı yazılım geliştiriciye anlatmak ve ikna etmek süreci beklenenden uzun olduğunda, her dakika maliyet olarak yansıyacaktır.

2.3 Tipik Bir Test Süreci ve Test Türleri

Yazılım dünyasında "Neden yazılımlar test ediliyor?" sorusu sorulduğunda iki yargıya varılmaktadır: Kalite ve Kabul Edilebilirlik (Jorgensen, 2013). Yazılımda kalite Bölüm 2.2'de belirlenen çeşitli özellikler ile ölçülebilmekte ve testler aracılığı ile kalite artırılabilmektedir. Yazılımlarda kabul edilebilirlik ise, müşterinin vermiş olduğu gereksinimlerin karşılanıp karşılanmadığının kontrolüdür. Yazılım geliştiriciye ya da proje yöneticisine göre mükemmel olan bir yazılım, eğer müşterinin istediklerini tam anlamıyla kapsamıyorsa, yine zaman ve para kayıplarına yol açmaktadır.

IEEE Software Engineering Body of Knowledge'a göre (SWEBOK, 2004); "Yazılım testleri, bir programa ait davranışın, sonsuz durum kümelerinden seçilmiş, sonlu test senaryoları eşliğinde ve beklenen davranış karşısında dinamik olarak doğrulamasıdır."

Yazılım geliştirme yaşam döngüsüne bağlı olarak test yapılış biçimi ya da test teknikleri değişkenlik gösterse de, tipik bir test süreci çerçevesi çizmek mümkündür. Test adımı, gereksinimlerin belirlenmesi aşamasından itibaren sürecin içindedir. Test mühendisleri her bir yazılım geliştirme yaşam döngüsü aşamalarına dâhil olarak kalitenin ölçülmesi, problemlerin analiz edilmesi ve uçtan uca bakış açısı ile projeye ait en önemli kararların verilmesinde bir mekanizma olarak görev yapmaktadır.

Test mimarı ya da test analisti görevindeki kişiler, proje kapsamı belirlenmesi sırasında düzenlenen gereksinimlerin belirlenmesi toplantılarına katılım sağlayarak, hem müşteri bakış açısı ile hem de proje grubu bakış açısı ile değerlendirmelerine başlamaktadır. Bu toplantılar sonucunda oluşturulan kapsam dokümanı test analisti ve test mühendisi görevindeki kişilere iletilir. Belirlenen kapsam doğrultusunda, problemlerin teknik olarak nasıl çözüleceğinin değerlendirilmesi için tasarım toplantıları düzenlenir. Test mimarının mümkünse bu toplantılarda da katılım göstermesi ve teknik açıdan konuya hâkim olması beklenmektedir. Tasarım toplantıları sonucunda oluşturulan çıktılar da test analisti görevindeki kişilere iletilir.

Kapsam ve tasarım toplantılarında elde edilen çıktılar test mühendisi tarafından uygunluk bakımından değerlendirilir. Bu aşamadan itibaren test mühendisi test senaryolarını yazmaya ve proje ekibinden gelen çıktılar doğrultusunda da senaryoları detaylandırmaya başlamaktadır. Test sürecinde hataların büyük çoğunluğu kodlama kaynaklı olsa da; tasarımda var olan hatalar, gereksinimlerin eksik ya da yanlış belirlenmesi, tasarım ile gereksinimler arasındaki uyumsuzluklar gibi sorunlar da test aşamasında tespit edilebilmektedir.

Test işletim aşamasına geçmeden önce tamamlanması gereken en önemli aşama test stratejisinin belirlenmesidir. Gereksinimlerin risk seviyesine ve entegre edilecek sistemlerin özelliklerine göre test analistleri ve test mimarları tarafından belirlenir, test işletimi için oluşturulması gereken ortamlar ve veriler ile birlikte, test koşumlarında hangi metodolojilerin izleneceği belirlenir. Test ortamlarının ve test verilerinin hazırlanması ile test senaryolarının tamamlanmasını takiben testlerin koşturulması adımına geçilmektedir. Test mühendisleri tarafından belirlenen riskler ve stratejiler doğrultusunda test senaryoları koşturulur, hata bulundukça yazılım geliştirici ile paylaşılır ve hataların düzelmesi süreci, yazılımdaki riskler en aza indirilinceye kadar devam etmektedir.

Test koşturma aşamasında çeşitli test türleri, metotları ve seviyeleri kullanılmaktadır. Test mühendisleri yazılım işlevlerini koda girmeyerek(kara kutu) ya da koda girerek(beyaz kutu) çeşitli şekillerde test ederler. Test türleri fonksiyonel ve fonksiyonel olmayan testler olmak üzere 2 ana sınıfta incelenebilir. İşlevleri belirleyen ve sistemin "ne" yapması gerektiğini belirleyen gereksinimler için fonksiyonel, işlevlere ek olarak sistem davranışı ve işlevleri "nasıl" yerine getirmesi gerektiği ile ilgili gereksinimlere için ise fonksiyonel olmayan test koşturmak uygun olacaktır.

Yazılıma ait fonksiyonel testler bittikten sonra, artık yazılımın tüm fonksiyonları gerçekleştirdiği ancak bu fonksiyonları gerçekleştirirken bazı seçilmiş koşullarda da bu işlevlerin doğru bir biçimde çalıştığının testi fonksiyonel olmayan testler olarak tanımlanır(Glinz, 2007). Bu kapsamda çeşitli stres, yükleme, performans, kullanılabilirlik ve güvenlik testlerinden yararlanılır.

Fonksiyonel ve fonksiyonel olmayan gereksinimler için günlük hayattan şu şekilde bir örnek verilebilir. Bir süt kutusu düşünüldüğünde, o kutunun sütü sızdırmadan taşıyabilmesi o kutunun fonksiyonel bir gereksinimidir; ancak o kutudan sütü sızdırıp sızdırmama işlevi dışında, süt markasının üzerinde yazması, alabileceği süt limiti, son kullanma tarihi vb. nitelikleri de taşıması beklenir. Bu nitelikler ise süt kutusuna ait fonksiyonel olmayan gereksinimlerdir.

Temel fonksiyonel testler sistemin istenen işlevi yerine getirip getirmediğini sınayan seviyelerden oluşmaktadır. Bunlar: Birim, Bileşen, Bütünleştirme, Sistem ve Kullanıcı Kabul seviye testleridir (Myers et al.,2011). Bu seviyelerin dışında, kullanılan yazılım geliştirme stratejisinin türüne göre ilerleme ve yineleme testleri gibi teknikler de kullanılmaktadır.

Projenin canlı ortama geçişinin yapılması için ise test ekibinin onayı gerekmektedir. Test koşturma süreci test mühendisi/mühendisleri yazılımda, entegrasyonda, sistemlerin iletişiminde problemli bir nokta tespit edemeyinceye kadar döngü halinde tekrar edilir. Kod en mükemmel hale gelince sıra "Gerçek Kullanıcı" testlerine gelmiştir. Ürün birkaç kullanıcının kullanacağı bir yazılım ise sadece bu kullanıcıların yazılımı kullanarak test etmeleri yeterlidir. Ürün geniş bir insan kitlesi tarafından kullanılacaksa önce belli bir grup kullanıcı tarafından test edildikten sonra geniş kitlenin kullanımına açılmalı ve bu geniş kitleden geri bildirim almak için bir mekanizma oluşturulmalıdır.

3. TEZ KAPSAMINDA GELİŞTİLEN YAZILIMDA KULLANILAN YAZILIM GELİŞTİRİM METOTLARI ve TEST TÜRLERİ

Tez kapsamında gerçekleştirilecek yazılım için ***"Artırımlı Yazılım Geliştirme***" metodu kullanıma uygun bulunmuş ve alt kırılım olarak da, yazılımın bir kişi tarafından geliştirildiği ve müşteri ile toplantılar yapılması gerektiği gerekçeleri göz önünde bulundurularak ***"Gereksinim Tabanlı Yaşam Döngüsü"*** modeli kullanılmıştır. Bu bölümde 1. Ve 2. alt bölümlerde sırasıyla bu konular incelenecektir. 3. alt bölümde ise tez kapsamında kullanılan test türleri detaylı olarak tanıtılacaktır.

3.1 Artırımlı Yazılım Geliştirme

Arttırımlı model Şelale modelden evrilmiş ve Şelale modelin kısıtlarından kurtulmayı hedefleyerek geliştirilmiş bir yazılım geliştirme tekniğidir (Jorgensen, 2013). Arttırımlı modelde ilk adım "Gereksinimlerin Belirlenmesi"dir. Ardından "Tasarım" aşamasına geçilir. Bundan sonraki aşama olan "Geliştirme" aşamasında, yazılım işlevleri önceliklere göre sıralanır her bir işlev için gerekli kodlar teker teker oluşturulur. Test aşamasına gelindiğinde, her artırımın sisteme eklenmesi ile bütünleştirme testleri, sisteme eklenen yeni artırımların önceki işlevi bozmadığına yönelik regresyon testleri ve artırımlara ait birim testler gerçekleştirilir.

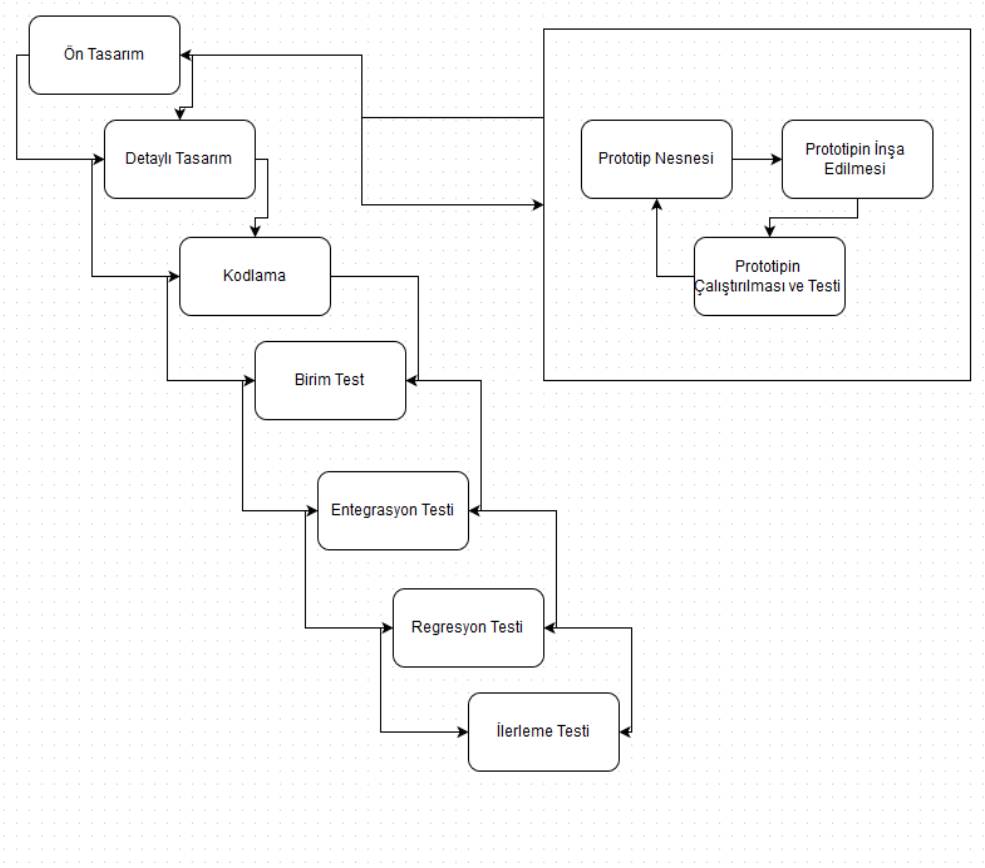
Bu şekilde tüm kodun geliştirimi gerçekleştirilir. Kodun belli olgunluğa eriştiği uygun zamanlarda müşteri ile bir araya gelinip gereksinimler ve yazılımın kullanım kolaylığı hakkında toplantılar düzenlenmesi toplam kod geliştirme süresini kısaltacaktır. Müşteri görüşmelerinden sonra gereksinimlerde değişiklik ya da ilk gereksinim belirleme toplantısında gereksinimlere dair yanlış anlaşılmalar olmuşsa kod geliştirici hemen düzeltime ve buna paralel olarak da test sürecine gider.

Eğer ürünün tek bir yazılım geliştirici yerine bir yazılımcı ekibi tarafından geliştirilmesi gerekiyorsa özellikle bütünleşme testleri daha büyük bir özen ve dikkatle yerine getirilmelidir. Test edilecek işlevlerin test ekibi tarafından risk analizi yapılıp belirli bir sıraya konulduktan sonra, test ekipleri arasında paylaşılması ve senaryoların uçtan uca koşulması bütünleştirme testi süresince riski en aza indirmek için yapılması gerekenler arasında yer almaktadır. Ekip çalışmalarında yazılım geliştirme süreci boyunca ekip üyelerinin bir araya gelerek yazılım hakkında sohbet etme temeline dayanan SCRUM tekniğini kullanmak büyük fayda sağlayacaktır(Beck et al, 2001).

Tez projesi kapsamında bir adet yazılım geliştirici bulunduğundan, SCRUM tekniğini kullanmak yerine "Gereksinim Tabanlı Yaşam Döngüsü" modeli kullanıma uygun bulunmuştur.

3.2 Gereksinim Tabanlı Yaşam Döngüsü

Sistemin, yazılım geliştirici ya da müşteri tarafından tam anlamıyla anlaşılmamış olduğu durumlarda, işlevlerin alt gruplara ayrılması en çok kullanılan tekniklerden birisidir. Barry Boehm(1988), bu konuda müşterileri şu şekilde tasvir etmektedir: "Ne istediğimi tam olarak bilmiyorum, ama görünce tanırım."(Jorgensen, 2013). Müşterilerin bu bakış açısına göre geliştirilmiş ve her aşamasının müşterilere gösterildiği bir model olarak gereksinim tabanlı yaşam döngüsü modeli kullanılabilir. Yazılım geliştirme aşamalarına dâhil olan müşteri, istediği özellikleri aşamalarda görecek ya da en azından yazılım geliştiricileri istekleri ve geri dönüşleri doğrultusunda yönlendirebilecektir. Şekil 3.1'de prototip(ilk örnek) yaşam döngüsü anlatılmaktadır:

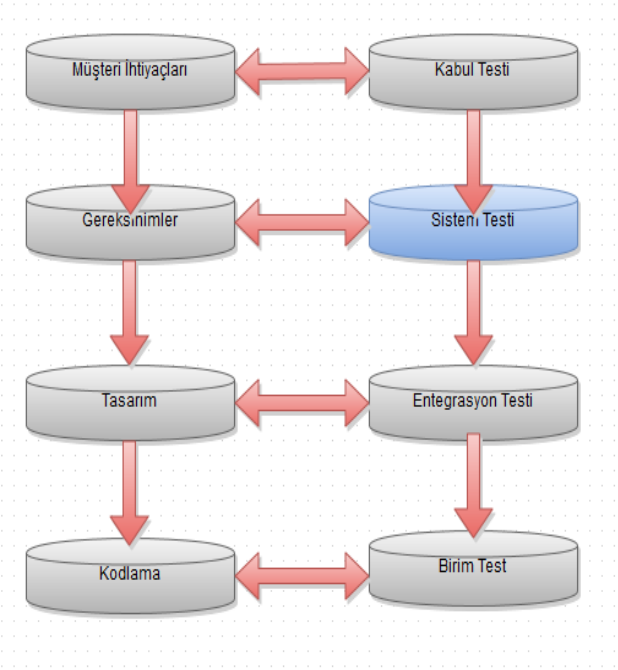


Şekil 3.1 – İlk Örnek(Prototip) Yaşam Döngüsü (Jorgensen, 2013)

Belirlenen gereksinimler üzerinden ilk örnekler oluşturulup, müşterinin beğenisine sunulur ve her ilk örnek sonrasında bir sonraki gereksinimin ne olacağına karar verilip, yeni sistem için ön tasarım yapılır. Bu döngü gereksinimler tamamlanıncaya kadar devam eder. Sistemi, uzun süre sonra, müşterinin beğenmeyeceği ve geri dönüşü de çok büyük zaman ve maliyet gerektiren bir sürece sokmak yerine; sürekli müşterilerin beğenisine sunulan, her aşamada ürüne ait bir artırım prototipi oluşan bir sürece sokmak hem zaman hem de maliyet açısından faydalar sağlayacaktır.

3.3. Tez Kapsamında Kullanılan Test Türleri

Tez kapsamında kullanılan test türlerini irdelemeden önce test seviyelerinin yazılım yaşam döngüsü ile ilişkisini Şekil 3.2 ‘den tekrar hatırlamak doğru olacaktır (Software Testing Class, 2015):



Şekil 3.2 - Test Seviyeleri ve Yazılım Yaşam Döngüsü İlişkisi

3.3.1 Birim testi

Yapısal bir programlama dilinde “birim” kelimesiyle, tek bir prosedür, bir fonksiyon, kodun bir fonksiyonu gerçekleştiren bir bölümü, bir sayfaya sığacak olan kaynak kod ya da kodun tek başına derlenip işletilebileceği en ufak bir bölümü olarak nitelendirilebilmektedir (Jorgensen, 2013). Nesne tabanlı bir programlama dilinde ise, genelde “birim” kelimesi ile nitelendirilen şey “nesne”dir (Jorgensen, 2013).

Birim testleri, yazılım geliştirme sırasında ve/veya yazılımın test takımına devri gerçekleşmeden hemen önce yazılım geliştiricisi tarafından yapılmaktadır ve fonksiyonel testler grubuna girmektedir. Birim testleri yazılım geliştiricisinin yapmasının sebebi, koda ve tasarıma yazılım geliştiricisinin daha fazla hakim olmasıdır, çünkü bu birimleri yaratan yazılım geliştiricinin ta kendisidir (Carlson, 2015).

Birim test, mümkün olan en alt seviyeleri de test edecek şekilde yapılmalıdır. Örneğin kodun bir bölümü mantıksal olarak ikiye ayrılmışsa; birim test bu mantıksal ayrımı gözetecek ve kodun her satırına girecek şekilde yapılmalıdır. Bu metot, beyaz kutu testi olarak da bilinir.

Birim testin amacı, sistemin en küçük bağımsız parçalarının, yine bağımsız olarak çalışıyor olduğunun; örneğin koddaki herhangi bir fonksiyon ya da prosedüre verilen girdiler sonucunda, doğru çıktıların döndüğünün kontrolüdür(ISTQB Exam Certification, 2015).

Yazılım birim testlerden geçmeden, bütünleştirme testi ya da entegrasyon testi aşamasına geçmesi doğru değildir. Entegrasyon testi aşamasında, yazılımlardaki birimlerde bulunan hatalar zaman kayıplarına yol açmaktadır. Birim testlerin yapılmasının, yazılım maliyetine bazı olumlu etkileri bulunmaktadır. Bunlar:

- **Hataların Erken Fark Edilmesi :** Birim test süreci, yazılımın test takımına geçmeden önce, yazılım geliştiriciye kendi kodunu test etmesi imkanını sunmaktadır. Böylelikle en erken düzeyde hatalar ayıklanmış olmaktadır. Yazılım birim testlerden geçtikten sonra, sonraki test adımlarında, birimlerden kaynaklanacak bir hatanın olmayacağı böylelikle teyit edilmiş olmaktadır (ISTQB Exam Certification, 2015).

- **Hata Düzeltilmesi Maliyeti :** Sistemdeki hatalar ne kadar erken bulunursa, bu hataların düzeltilmesinin maliyeti de o kadar azalacaktır. Sistem seviyesi testlerinde bulunacak birimlerdeki bir hata, test sürecinin tekrarlanmasını gerektirecek ölçüde bile olabilmektedir. Burada örneğin bir değişkenin alacağı bir değerden bahsediyor olalım, bu değere göre kod içerisinde bazı dallanmalara girdiğini ve buna göre çıktı ürettiğini düşünelim. Sistem testi sırasında, yapılan testlerin bazıları testlerden başarıyla geçmiş olsa bile, bu değişkenin hatalı değerde olması; aslında kod özelindeki bazı hataların doğru sonuç vermesine yol açmış olabileceğini gösterir. Bu durumda etkilenecek kısımlar için ayrıca regresyon testi planlanması gerekir ki; maliyet ve kaynak açısından getirdiği negatif etki kolaylıkla görülebilmektedir. Birim testleri, bu etkileri en aza indirmektedir.

Birim testler, bazı alt başlıklar halinde incelenebilir. Birim testler sırasında yazılımın tipine göre aşağıdaki tipteki testlerin de gerçekleştirilmesi uygun olur(Jorgensen, 2013).

- Uç Değer Testi
  - Normal Uç Değer Testi
  - Güçlü Uç Değer Testi
  - En Kötü Durum Uç Değer & Güçlü En Kötü Durum Uç Değer Testi
- Eş Değerlilik Sınıfları Testi
- Karar Tablosu Tabanlı Test
  - Karar Tabloları
  - Neden-Sonuç Grafikleri
- Yol Testi
- Veri Akış Test

Birim testler yazılım geliştirici tarafından gerçekleştirilen testlerdir. Bu test aşamasına, test mühendisleri, analistler ya da üçüncü parti kişiler dahil olmamaktadır. Yukarıdaki tekniklerin bir kısmı beyaz kutu yöntemini, bir kısmı

ise kara kutu yöntemini kullanmaktadır. Birim testlere ait özet tablo olan Tablo 3.1 referans alınabilir.

Birim Test Tekniği	Kullanılan Test Metotu	Testi Gerçekleştiren Kişi
Uç Değer Testi	Kara Kutu	Yazılım Geliştiricisi
Eş Değerlilik Sınıfları Testi	Kara Kutu	Yazılım Geliştiricisi
Karar Tabloları Testi	Kara Kutu	Yazılım Geliştiricisi
Yol Testi	Beyaz Kutu	Yazılım Geliştiricisi
Veri Akışı Testi	Beyaz Kutu	Yazılım Geliştiricisi

Tablo 3.1 - Birim Testlere Genel Bakış

3.3.2 Bileşen testi

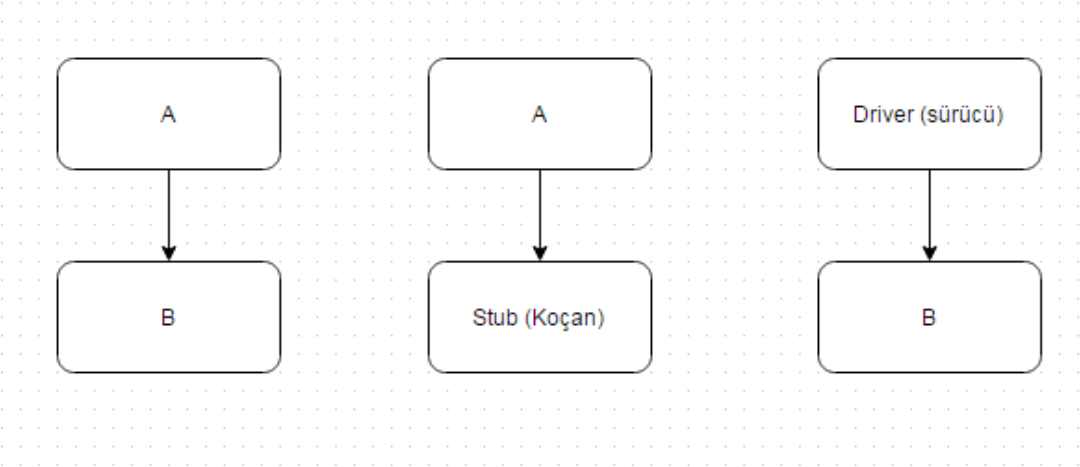
Bileşen testi, ISTQB'ye göre, sistemi oluşturan her bileşenin birbirinden bağımsız ve doğru çalışıp çalışmadığının kontrolüne ait testtir. Bir yazılım sisteminde beş adet bileşen olduğu düşünüldüğünde, her bileşenin birbirinden bağımsız ve ayrı bir şekilde test edilmesi "bileşen testi" olarak adandırılmaktadır. Bu test tipi "modül testi" olarak da geçmektedir.

Bileşenlerin birbirinden bağımsız olmadığı durumlarda, örneğin A ve B modülleri var ve A modüllünün test edilmesi için B modülünden gelece bilgilere ihtiyaç olduğu durumlarda, "stub(koçan)" ve "driver(sürücü)" adı verilen kod parçacıklarından faydalanılır. Bu kod parçacıklarının bibirinden farkı, stub'ın yukarıdan aşağıya entegrasyon, driver'ın ise aşağıdan yukarı entegrasyon adımlarında kullanılmasıdır.

Bir örnek ile açıklanacak olursa, bir uygulamaya giriş yapmak için yazılmış olan giriş sayfası ve henüz yazımı tamamlanmamaış Anasayfa ve Kullanıcı modülleri tasarlanmış olsun. Giriş sayfasının testi için, Anasayfa'dan girilmiş ve Kullanıcı bilgileri alınmış olmalı ise, bu modüller henüz tamamlanmadığı için, "***stub***" adı verilen kod parçacığı ile, giriş sayfasına ait bilgilerin girilmiş olduğu simülasyon ile yaratılmış olur. Bu yukarıdan-aşağıya test modeli ile uyumlu bir durumdur. Bu örnek değiştirildiğinde, yani Anasayfa ve Kullanıcı modülleri var olduğunda ve Giriş sayfası henüz hazır olmadığında, "***driver***" adı verilen kod

parçacığı yazılarak simülasyon sağlanmış olur. Özetle stub çağrılan, driver ise çağıran modüller için çalışacaktır denebilir.

Aşağıda Şekil 3.3'te bu durum açıklanmaktadır:



Şekil 3.3 – Bileşen Testi – Sürücü ve Koçan Gösterimi(ISTQB Exam Certification, 2015)

Bileşen testlerindeki amaç, entegrasyon testlerinden önce her bileşenin doğru fonksiyonalite ile çalıştığının kesinleştirilmesi ve entegrasyon testleri sırasında alınan hatalar için, birimlerden şüphelenmek yerine birleşim noktalarındaki alınan ve gönderilen veriler, türleri, iletişimin nasıl yapıldığı vb. konularına odaklanılmasını sağlamaktır.

Yazılım geliştiriciler, kullandıkları yazılım dili veya tasarımın engellediği faktörler dışında, yazılımları bileşenler halinde geliştirirler. Böylelikle üçüncü parti bir yazılım şirketinin bir ürünü sistem ile entegre edilmek istendiğinde, yalnızca ilgili bileşene ait verilerin, girdilerin veya çıktıların paylaşılması yeterli olacaktır(Gao, 2000). Bu durumda eğer entegrasyon sırasında bir sorun ile karşılaşılırsa, ilgili bileşende sorun olmadığına emin olunur ve sorunun kök sebebi başka noktalarda aranarak vakit kaybının önüne geçilmiş olur.

Yazılımı bileşenler halinde geliştirmenin bir diğer avantajı, bileşenlerin yeniden kullanılabilirliğini artırmaktır. Yeniden kullanılabilirlik sadece yazılım geliştirme adımında değil, test adımında da gündeme gelmektedir. Her bileşenin akışı ve metotları belli ise, bu bileşeni test etmek için kullanılan test senaryoları, test verileri, scriptler vs. de yeniden kullanılabilir olmaktadır(Gao,2000).

Bir önceki test seviyesi olan birim testlerde, testler yazılım geliştirici tarafından yapılmaktaydı, bileşen testi aşamasından itibaren kod test mühendisinin kontrolü altına girer ve bu testler test mühendisleri tarafından tamamlanır. Yazılım geliştirici yalnızca birim testler aşamasına dahil olmaktadır.

3.3.3 Duman testi

Çok eski dönemlerde bir donanımın çalışıp çalışmadığını anlamak için içinin açılıp, duman çıkıp çıkmadığına bakmak olarak kullanılırmış. Yazılımlarda da aynı mantık bir nevi devam etmektedir. Bu yüzden bu testin isimlendirilmesinde bu teknikten ilham alınılmıştır. Geliştirilen yazılımların derinlemesine testlerine başlanmadan önce, temel birkaç fonksiyonalitenin çalışıp çalışmadığına bakarak, projenin teste devri değerlendirilir. Temel amaç, test ortamlarına kurulumun yapıldığının ve sistemin teste hazır olup olmadığını belirleyebilmektir.

Duman testleri, hataların önceden fark edilmesini sağladığı ve zaman kazandırdığı için sistem testlerine başlanmadan önce mutlaka yapılmalıdır.

3.3.4 Sistem testi (Uçtan uca test)

Sistem testi aşamasında, yazılıma ait tüm bileşenlerin birbirlerine entegre oldukları ve entegrasyon testlerinden başarılı bir şekilde geçtikleri bilinmektedir. Bu aşamada, yazılımın tek başına test edilmesinin yanı sıra, çeşitli çevre koşullarına olan etkisi de test edilmektedir. Örneğin yazılım çalıştığı sürede bilgisayar kaynaklarından ne kadarını kullanmaktadır, işletim sistemi veya çalışan küçük iş birimleri ile ilgili bir sorun oluşturur mu gibi sorunlar incelenebilmektedir. Sonuç olarak, fonksiyonel testlerin yanı sıra, fonksiyonel olmayan testlerin de bu aşamada gerçekleştirilmesi bu söz konusudur. Testler test mühendisi tarafından gerçekleştirilmektedir.

Testler, gereksinimlerin ve beklentilerin karşılanıp karşılanmadığını son ürün üzerinden gözlemlemeye çalıştığı ve artık kodun içi ile ilgisi kalmadığı için kara kutu yaklaşımı ile ele alınmaktadır.

Bu sorulara cevap bulabilmek için, mümkün olan en fazla ölçüde yazılımın canlı ortama yakın olması, günlük kullanıma en yakın koşullarda olması gerekmektedir. Jorgensen, sistem testleri sırasında yeni bir kavram üzerinde durmuştur(2013): ASF (Atomic System Function, En küçük sistem fonksiyon birimi). Bu kapsamda sistemdeki thread'leri(küçük iş parçacıkları) incelemiştir. En küçük sistem birimi fonksiyonları yazılımda veriler, aksiyonlar, aygıtlar, olaylar, küçük iş parçaları(thread) bazında irdelenmiştir. Bazı yazılımlar veri merkezli olarak tasarlanmaktadır, bu durumlarda en küçük fonksiyon birimlerini verilerden seçmek mantıklı olacaktır. Bazı durumlarda aksiyon merkezli yazılımlar tasarlanmaktadır, bu durumda ise en küçük fonksiyon birimi aksiyonlar olarak belirlenebilmektedir.

Test, daha önce de ifade edildiği gibi, entegre olmuş tüm uygulamalar üzerinden gerçekleştirilmektedir, bu sayede entegre olan bileşenlerin de birbirine olan etkileri uçtan uca test yapılarak görülebilmektedir. Sistem testleri bu sebeple "uçtan uca test" olarak da anılmaktadır. Uçtan uca testler hem fonksiyonel hem de fonksiyonel olmayan gereksinimler için kullanılabilmektedir. Kullanılabilen

sistem testi çeşitleri arasında: kullanılabilirlik testi, yük testi, regresyon testi, kurtarma testi, göç testi, donanım testi sayılabilir.

3.3.5 İlerleme testi

İlerleme testleri, regresyon testleri olarak da bilinmektedir. Yazılıma güncelleme geldikten sonra, tüm işlevlerin doğru yerine getirilip getirilmediğini kontrol eden testlerdir. Gerçekleştirilen değişiklikler istenmeyen yan etkiler doğurabilir. İlerleme testleri ile, önceki testlerde elde edilen sorunların giderildiğinden ve bu sorunların düzeltilme sürecinde var olan işlevlerin etkilenmediğinden emin olunmaktadır.

Özellikle artırımlı test süreçlerinde, sürecin doğasına uygun olarak belirli sürelerde yeni artırımlar gelmekte ve yazılım sürekli büyümektedir. Bu projelerde riski en aza indirmek için her artırım sonrasında bir ilerleme testi koşulmalıdır.

Burada dikkat edilmesi gereken önemli bir nokta, yazılıma ait tüm test senaryoların her defasında baştan itibaren koşulamayacağıdır. Bu hem zaman kaybına hem de gereksiz maliyet kaybına neden olabilir. Yıllar içinde, regresyon test setinin seçim teknikleri ile ilgili çalışmalar yapılmıştır(Rothermel et al 1997; Li et al, 2007). Çözümlerden en uygunu Test Güdümlü Yaklaşım kullanarak eskiden oluşturulmuş testleri kaydetmektir. Bu şekilde tüm senaryolar kısa sürede test edilebilir.

İlerleme testi, fonksiyonel testler arasında incelenmiştir. Ancak sorunların düzeltilmesi sonrası oluşturulacak regresyon test senaryolarının koşturumu sırasında, yavaşlık vb. gibi sistemin fonksiyonel olmayan gereksinimlerine de etki ettiği görülebilir. Bu testlerinin bir amacı da, yazılımdaki değişimlerin var olan yapıyı etkileyip etkilemediğini gözlemlemektir, bu etkilere fonksiyonel gereksinimler gibi fonksiyonel olmayan gereksinimler de dâhildir.

3.3.6 Kullanılabilirlik testi

Kullanıcı kabul testi, yazılım test süreçlerinin son adımıdır. Bu aşamaya kadar, son kullanıcı gözü ile yazılım incelenmemiş, teknik konular geliştirilmiş ve teknik sorunlardan arındırılmıştır. Kullanıcı kabul testlerine, önceki tüm aşamaların yani geliştirme ve test aşamalarının test edilmesi aşamaları bittikten sonra başlanabilir.

Kullanılabilirlik, bir ürünün potansiyel kullanıcıları tarafından, amaçlandığı gibi etkin ve verimli bir şekilde kullanabilmesinin testidir. Bilgisayar ile iletişim kurma aracı yazılımlara ait ara yüzlerdir. Ana amaç da yazılımı son kullanıcının kullanmasını sağlayarak verimli sonuçlar üretmek olduğuna göre, yazılımın kullanılabilirliği en önemli metriklerden biri haline gelmektedir.

Örnek vermek gerekirse, ODTÜ İnsan-Bilgisayar Etkileşimi Araştırma ve Uygulama Laboratuvarı‟nda yapılan çalışmalarda, ara yüz belirlendikten sonra, bu çalışmalar sırasında kullanılarak araçlar (örn. Anketler), hedef kullanıcı grubu ve hedef kullanıcıların yapacakları görevler belirlenmektedir(2015). Her görev için görevi başarıp başaramadığı, yaptığı hata sayısı ve süre bilgileri tutulmaktadır. Test aşamasında kullanıcıdan sesli dününmesi istenerek, ara yüzde nerelerde sorun yaşadığı ve ne düşünüldüğü öğrenilmektedir.

4. VAKA ÖRNEĞİ: KARACİĞER VOLUMETRİ YAZILIMI TESTİ

Karaciğer transplantasyonu son evre karaciğer hastalığının en etkin tedavi yöntemidir. Gelişen cerrahi teknikler ve cerrahi sonrası uygun hasta takibi hem alıcı hem de verici mortalitesinde düşüşe neden olmuştur. Canlı vericiden olan nakillerde verici olası risklerden uzak olmayıp cerrahi öncesi ve sonrası dönemlerde oldukça dikkatli değerlendirilmelidir. Cerrahi öncesinde vericinin yaşı, immünolojik durumu, medikal öyküsü, karaciğer ve böbrek fonksiyonları, var olan hastalıkları değerlendirilmelidir. Bu değerlendirmede radyolojik incelemeler önemli rol oynamaktadır. Cerrahi öncesi incelemede radyolojik olarak şu unsurlar değerlendirilir:

- Karaciğerdeki yağlanma oranı
- Arteriyel, venöz ve portal vaskülarizasyon ve anatomi
- Total ve segmental volüm ölçümü
- Hepatektomi yapılacak alanın cerrahi öncesi çizimi

Karaciğer volümü ve kilo parametreleri verici uyumluluğu açısından en önemli değerlendirme kriterleridir. Verici cerrahi sonrası hayatını devam ettirmek için gerekli karaciğer dokusuna sahip olmalıdır.

Volümetri için kontrastlı üst abdomen BT çekimleri kullanılır. Günümüzde volumetri manuel yöntemle, *İnteractive Volumetry Assist Software* veya otomatik yöntemlerin kullanıldığı programlar ile yapılmaktadır. Manuel incelemenin uzun süre alması radyologları diğer 2 yöntemi kullanmaya yönlendirmiştir.

Karaciğer boyutunun ve konturunun değişken ve kişiye özgü olması, karaciğer dokusunun yoğunluğunun komşu barsak ve kas dokuları ile yakın olması nedeniyle sınırların keskin olmayışı ve parsiyel volüm etkisi (bir BT artefaktı) nedeniyle karaciğer dokusunun çevre dokudan ayrımının zorlaşması volumetrinin otomatizasyonunu zorlaştırır.

Karaciğer hacim ölçümü için en uygun radyoloji modalitesi bilgisayarlı tomografidir. Ölçüm düşük kesit aralıkları ile y ekseni boyunca tarama yapılarak gerçekleştirilmelidir. Taranan resimler bir DICOM dosyasına kaydedilir. Volumetri yazılımlarında doktor karaciğer kesidi içeren resimler arasında uygun olanlarını seçerek karaciğer dokusuna ait konturları belirler. Yazılım bir resimde bu kontur içindeki veri noktasının sayısını belirleyerek alanı hesaplar. Kesitler arası mesafe bilgisi kullanılarak da hacim değeri hesaplanır.

4.1 Gereksinimlerin Belirlenmesi ve Tasarım

Mevcut volumetri programının sahip olduğu bazı problemlerden dolayı yeni bir volumetri programı geliştirilmesi için gereksinim ortaya çıkmıştır. Mevcut volumetri yazılımının şu problemleri mevcuttur:

1. Karaciğer dokusu kontörü fareyle nokta nokta seçim yaparak belirlenmektedir. Bu seçim yöntemi, karaciğerin boyutuna göre belirlenmiş 10-20 resim üzerinde ve karaciğer sınırları her bir resimde noktalar halinde doktorun dahli ile yapılmaktadır. Bu durum kullanım zorluğuna neden olmaktadır.
2. Kullanım zorluğu nedeniyle bir süre sonra doktor hassasiyetini kaybedebilmekte ve kontür seçiminde hatalar yapabilmektedir. Karaciğer dışındaki noktalar da karaciğere dahil edilebilmekte ya da karaciğer sınırı içinde bir bölgeden kontür seçimi yapılabilmektedir.
3. Yanlış seçim yapılmış bir resme ait iptal ya da düzenleme olanağı yoktur. Yazılım hata kabul etmemektedir.
4. Var olan yazılım ile, hasta çekim yapılan BT cihazının bulunduğu merkeze bağlı kalmak durumundadır. Var olan program yalnızca bağlı bulunduğu BT cihazı iş istasyonunda çekim yapıldığında iş görmektedir.

Bu problemleri gideren bir yazılım geliştirimi için 1 proje yöneticisi, 1 uzman radyolog, 1 yazılım geliştirici ve 1 test uzmanından oluşan 4 kişilik bir ekip oluşturulmuştur. Çok modüllü ve büyük kapsamlı yazılım geliştirim projelerinde roller daha fazla çeşitlenebilmekte ve gereksinim belirleme toplantılarına mutlaka test mimarları da katılmaktadır. Ancak tez projesi kapsamında üretilecek olan çözüm bir tek modülü belirlediğinden ve problem net bir şekilde anlaşıldığından dolayı gereksinim belirleme toplantılarında test mimarı görev almamıştır. Belirlenen gereksimler aşağıdaki gibidir:

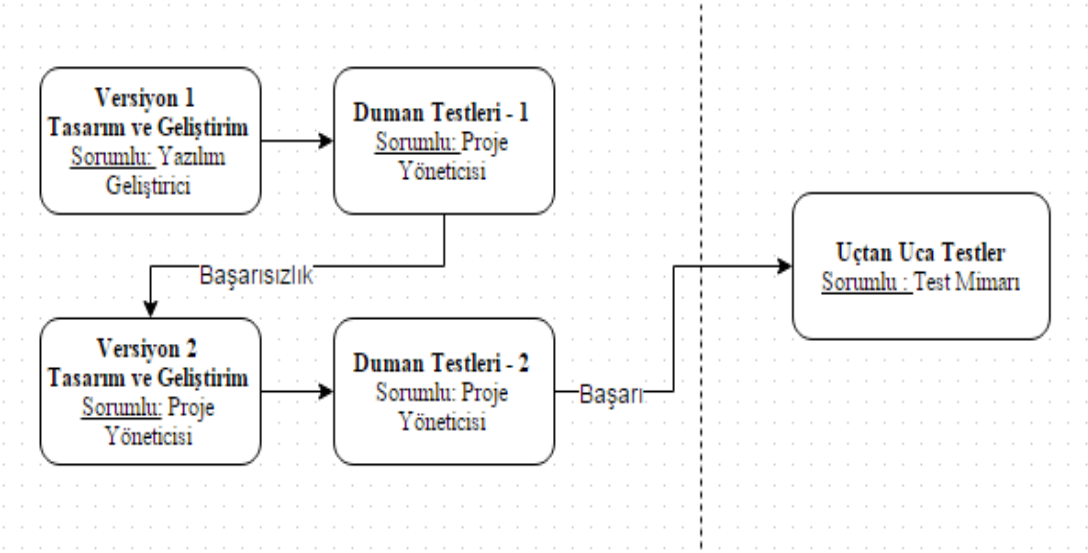
1. Yazılım geliştiriminde görüntü işleme olanaklarına sahip olması nedeniyle MATLAB kütüphanesi kullanılacaktır.
2. Görüntüler kontrastlı BT görüntü olacaktır ve DICOM dosyaları içinde yazılıma kabul edilecektir.
3. Nokta nokta seçim yapmak yerine fare sürükleme yoluyla kullanım kolaylığı ve hız sağlanacaktır.
4. Bu şekilde yanlış kontör seçimlerini minimize edilmiş olacak ve gerçek hacim değerine en iyi yaklaşım elde edilecektir.
5. Seçim yapılan her resimde elde edilen alan bilgisi ekranda listelenmeli ve bir tuş tıklanınca hacim hesabı doğru olarak yapılmalıdır.
6. Yazılım bir BT cihazına bağlı çalışmayacaktır. Kişisel bilgisayarlara yüklenmesi yeterli olacaktır. Böylece hasta tek bir merkeze bağlı kalmayacaktır. Gidip istediği yerde çektirdiği bir kontrastlı BT dosyayı operasyon sonrası takipte kullanabilecektir.

Gereksinimlerin belirlenmesinden sonra, yazılım geliştiricinin üstlendiği tasarım aşamasına geçilmiştir. Uzman radyalog tarafından da 10 uygun üst abdomen CT DICOM CDsi sağlanmıştır.

4.2 Gerçekleştirim Süreci

Tasarım, bir önceki başlıkta bahsedildiği gibi yazılım geliştiricinin inisiyatifine bırakılmıştır ve bu süreçte proje yöneticisi ve test mimarının katkısı olmamıştır. Sürecin böyle olması, yazılıma ait iki versiyonun oluşturulmasına neden olmuştur. Yazılım geliştiricinin ürettiği ilk versiyonda, yazılımın uçtan uca teste devri gerçekleşmeden önce proje yöneticisi tarafından duman testi gerçekleştirilmiştir ve bu test sırasında tasarımda oldukça fazla hatalar yapıldığı fark edilmiştir.

Hataları kabul etmek istemeyen ve gerekli düzenlemeleri yapmayan yazılım geliştirme sorumlusunun artık projede devam etmemesine, tasarım ve kodlama aşamalarına ait ikinci bir versiyon oluşturulmasına karar verilmiştir. Yaşanan bu duruma ait özet şekil Şekil 4.1'de verilmiştir:



Şekil 4.1 - Gerçekleştirim Süreci

4.2.1 Tasarım geliştirim ve duman testleri – Versiyon 1

Yukarıda belirtildiği gibi; tasarım, yazılım geliştiricinin inisiyatifine bırakılmıştır ve bu süreçte proje yöneticisi ve test mimarının katkısı olmamıştır. Gerçekleştirim sırasında belli aralıklarla proje yöneticisi ve yazılım geliştirici bir araya gelerek fonksiyonel ve fonksiyonel olmayan işlevlerden örnekler testleri gerçekleştirmişlerdir. Bu şekilde birkaç oturum gerçekleştirilmiş ve tasarımın hatalı olduğu kanısına varılmıştır. Bu versiyonda oturumlar sırasında tespit edilen hatalar ve muhtemel sebepleri aşağıda maddeler halinde belirtilmiştir:

- **Yazılımdaki Yavaşlık:** İlk DICOM resminin görüntülenmesi için 20 sn. kadar beklenmektedir. Resimler arası geçiş sırasında da yavaşlık, bazen de kilitlenmeler tespit edilmiştir. Bu durumun sebebi, gerek olmadığı

halde herbir dosyaya ait aynı bilgileri içeren meta dosyasının okunması ve verilerin isme göre sıralanmaya çalışmasıdır.

- **Örnek DICOM Verilerinin Kullanımındaki Hata:** İletilen test DICOM verisi 1000 adet resim içermektedir. Geliştirilen uygulamada ise yalnızca ilk 15 adet karaciğer dokusu içermeyen görüntü ile işlem yapılmaktadır.
- **Görüntüler Arasındaki Seçim Yapma Olanağı:** İlk oturumda görüntüler arasında seçim yapma imkanı bulunmamaktadır. Daha sonraki düzenlemeler ile istenildiği gibi "<" ve ">" tuşları yerine 1., 11. Gibi sabit bir resim atlama aralığı konulduğu ve atlanan resimlerin ekranda görüntülenmediği görülmüştür. Oysa doktor tüm resimleri görebilmeli ve uygun gördüğü resimde durarak seçim yapabilmelidir.
- **Atlama ve Seçim İşlemleri:** Atlama ve seçim işlemleri uygulamada kilitlenmeye neden olmaktadır. Yazılımcı bu probleme çözüm olarak, uygun resme gelindiğinde, kendi çözümü olan "START" butonuna basılması sureti ile kontör seçimi işlemine geçilebilecektir. Bu şekilde bir kullanım uygun bir kullanım değildir, projenin tümüyle iptaline yol açacak bir kullanımdır.
- **Arayüz Tasarımı:** Yazılım arayüz tasarımı özensiz ve dağınıktır. Menüler gereksiz işlevleri içermektedir. Yazılım geliştiricisi volumetri yazılı için gerekli olmayan bir fonksiyonu da yazılıma katarak arayüze onunla ilgili eklemeler yapmıştır. Arayüzde kullanılan metinlerin rengi, fontu ve büyüklüğü de farklıdır.

Gerçekleştirilen birkaç oturumda da yazılımda istenilen seviyeye gelinememiş ve yazılım geliştiricinin hataları kabullenmemiş olmasından dolayı yazılım geliştirici ile proje bağı koparılmıştır. Bu aşamadan sonra, yazılımın tasarımının incelenmesi ve gerekirse ikinci versiyonuna ait tasarım ve geliştirimin yapılabilmesi için proje yöneticisi inisiyatifi ele almıştır. Tüm bunlara ek ve "gereksinim tabanlı yazılım geliştirme modeli"ne uygun olarak, oturumlar sırasında doktorun görüş ve önerileri doğrultusunda yeni gereksinimler eklenmiştir. Bu gereksinimler:

1. Yazılımın penceresi küçüktür. BT resimler küçük çıkmaktadır. Pencere ekranı kaplamalıdır. Ayrıca arayüz daha göze hoş hale getirilmelidir.
2. Görüntü kalitesi düşüktür. Görüntü kalitesinin arttırım yolları araştırılacak ve uygulanacaktır.
3. Yazılım kullanıcının (doktorun) istediği resimlerde seçim yapmasına izin vermelidir.
4. Kullanıcı daha önceden seçim yapmış olduğu bir kesidi/kesitleri iptal edebilmelidir.
5. Ana menüye yazılım kullanımını açıklayan bir *help* menüsü eklenmelidir.
6. Kullanıcı >,< tuşları yanı sıra fare scrolunu da kullanarak resimler arasında dolaşabilmelidir.

Bu bilgiler ışığında versiyon 2 için incelemeler başlatılmıştır.

4.2.2 Tasarım, geliştirim ve duman testleri – Versiyon 2

Tasarımın incelenmesi adımına kaynak kod irdelenerek başlanmıştır. Gecikmeli olarak iletilen kaynak koda ait ilk incelemelerde tespit edilen unsurlar:

- Kod oldukça dağınıktır.
- Hiçbir işleve karşılık gelmeyen kod parçacıkları bulunmaktadır.
- Bazı işlemler gereksiz karmaşıklıkta yapılmıştır. Bu durum çalışma zamanını ve bilgisayar kaynak kullanımını etkileyecek önemli bir faktördür.
- Birçok gereksiz değişken bulunmaktadır.
- Değişken isimleri amacına uygun seçilmemiştir.

Bu unsurlara ek olarak, MATLAB aracılığı ile de kod kalitesi hakkında raporlar edinebilmek mümkündür. Ek 2 ve Ek 3'te MATLAB'ın oluşturduğu kod analiz ve bağımlılık raporlarından örnek ekran görüntüleri mevcuttur.

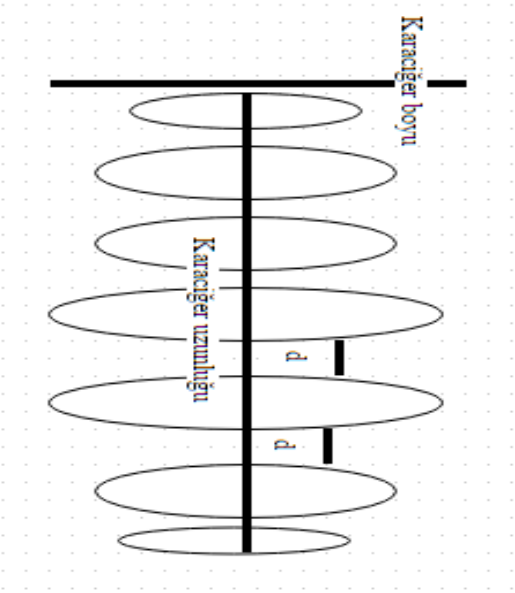
Proje yöneticisi 5 gün boyunca kod üzerinde temizlik ve düzenlemeler yaparak iyileştirmeler gerçekleştirmiştir. Ancak arayüz öğeleri arasında akışla ilgili problemi çözmek mümkün olmamıştır. Daha doğru bir ifade ile proje yöneticisi bu problemin çözümünün mümkün olmadığını fark etmiştir. Dolayısı ile yazılım duman testlerinden sınıfta kalmıştır.

Proje yöneticisi tarafından MATLAB ile ilgili yazılımlar incelenip, araştırmalar yapıldıkça; yazılım geliştiricinin yanlış yazılım geliştirme tekniği kullandığı farkedilmiştir. *MATLAB ortamının guide isimli bir arayüz oluşturma olanağı vardır. Görsel yazılım geliştirme tekniğine uygun olarak önce bu ortamda arayüz oluşturulması gerekmektedir. Bu arayüz saklandıktan sonra otomatik olarak bir .m dosyası oluşmaktadır. Bu dosyada her bir GUI öğesine ait callback fonksiyonları otomatik olarak yerleşmiş gelmektedir. Yazılım geliştiriciler bu fonksiyonlarının gövdelerini oluşturarak yazılımını gerçekleştirmektedirler.* Ancak yazılım geliştirici bu yolu tercih etmemiş, .m uzantılı dosyayı sıfırdan yazarak oluşturmuş ve tüm statik GUI öğelerinin dinamik GUI öğeleri gibi *uicontrol* fonksiyonu ile yaratılmasını sağlamıştır.

Kod irdelenmesi tamamlandıktan ve doğru strateji bulunduktan sonra versiyon 2 tasarım ve geliştirimi yapılmıştır. Birinci versiyonda kod incelemesi sonrasında fark edilmiş ve çözüm getirilmiş diğer konular aşağıda maddeler halinde belirtilmiştir:

- **Hacim Değerinin Yanlış Hesaplanması:** Formülün implementasyonunda kesitler arası aralık değeri ve atlanan kesit sayısı kullanılmamıştır. Ayrıca hacim hesabı kodu çok uzundur. Gereksiz veriler de kullanılmış ve işler dolandırılarak 500 satıra yakın bir kodlama ile hacim hesabı yapılmıştır. 2. versiyonda 8 satırlık bir kod hacim hesabı için yeterli olmuştur: *Bu durum*

*eski yazılım geliştiricinin birim testleri gerçekleştirmediğinin ya da yetersiz gerçekleştirdiğinin bir ispatıdır.*Yeni tasarım modeline göre;



Şekil 4.2 - Karaciğer Üzerinden Alınan Kesitler

Şekil 4.2'den görülebileceği gibi, karaciğer üzerinden dikey kesitler alınmış ve kesitler arası uzaklık olan "d" uzunluğu sabittir. Bu uzunluğun değeri görüntü çekimi sırasında ayarlanabilmektedir. Seçilen bölgeler ise, karaciğerin şekline göre birbirinden farklı boyutlardadır. Bu nedenle, hacim hesabının **seçilen kesitin alanı * d** olarak her birim için hesaplanması uygun olacaktır. Daha sonra belirlenen alanlar için kümülatif hacim toplamı karaciğerin hacmini verecektir. Ancak burada dikkat edilmesi gereken önemli bir nokta tespit edilmiştir. Daha önce bir karaciğerin yaklaşık 200 DICOM dosyasından oluştuğu belirtilmiştir. Doktorun herbir dosyayı inceleyip üzerinden seçim yapması mümkün değildir. Doktor uygun bulduğu alanlar üzerinden istediği miktarda seçimi yapabilecektir. Bu durumda 200 DICOM dosyasından Şekil 4.3'te görüldüğü gibi 7 adet kesit alanının belirlendiği durumlarda yazılım tarafından bir yakınsama yapılması gerekliliği tespit edilmiştir. Açıklamak gerekirse; ilk alan seçiminin 10.resim üzerinden yapıldığını varsayalım. Yazılım, bu alanın, kendinden önceki 9 tane alan ile aynı olduğu varsayımını kullanmaktadır. İkinci alan seçiminin ise 17. Resim üzerinden yapıldığını varsayalım, bu durumda ilk 10 tanesi için hesaplamanın tamamlandığı ve (17 – 10) tane kesit için hesaplama yapıldığı görülmektedir. Elde edilen 7 sonucu, 17. Alanın değeri ile çarpıldığında, hacim toplamına eklenecek yeni değer elde edilebilecektir. Bu hesaplama, seçilen tüm alanlar ile ilgili hesaplamalar tamamlandıktan sonra son bulacak ve ekrana sonuç değeri yazılacaktır.

- **Gürültülerin de Hesaba Dahil Edilmesi:** Yazılımın bilimsel çekirdeğini görüntü işleme oluşturmaktadır. Fare ile sürüklenerek oluşturulmuş kapalı

alandan karaciğer dokusuna ait olmayan parçalar atılmalı ve alan hesabı kalan veriler kullanılarak gerçekleştirilmelidir. Bu durum tüm hacmin hesaplanmasında doğruluk ölçütünü arttırarak gerçek hacme en yakın değere ulaşılmasını sağlayacaktır. Yazılım irdelendiğinde, gürültülerin de ilk versiyon hesabına dahil edildiği görülmüştür. İkinci versiyonda bu problem *opening* işlemi ile çözülmüştür. Bu işlemde belirlenen bir array görüntünün üzerinde dolaşır ve fark ettiği karaciğere ait olmayan kısımları siler. Her pixelin bir renk değeri olduğundan, bu array ilgili pixelin karaciğere ait olup olmadığı kolaylıkla anlaşılabilmektedir.

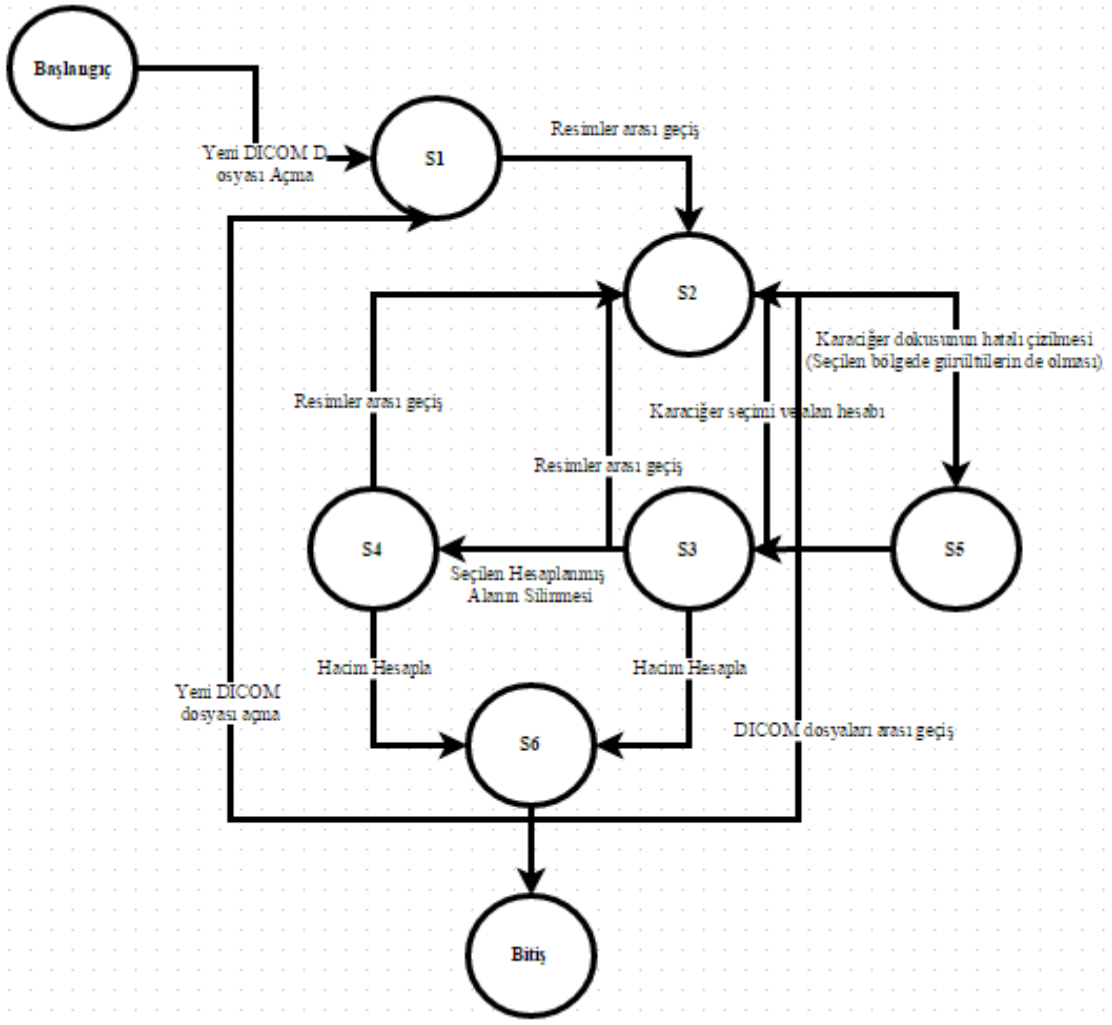
- **Görüntü Kalitesi:** Karaciğerin diğer dokulardan belirgin bir biçimde ayrılması gereksinimler arasında mevcuttu, ancak ilk versiyonda bu konuyla ilgili hiçbir çözüm tasarımı yapılmamıştı. İkinci versiyonda bu konuya dinamik aralık (dynamic range) ayarı yoluyla çözüm getirilmiş ve kalite yeterli derecede iyileştirmiştir. CT dosyalarında nokta değerleri 0 ve 4095 arasında değişir. Ekranda kullanılabilecek en yüksek piksel değeri ise 255'tir. Normalizasyon yapılarak [0-4095] aralığındaki değerlerin [0-255] aralığına çekilmesi ile bu sorun çözüme kavuşmuştur.

Bu çözümler üretildikten ve kodlama tamamlandıktan sonra, uçtan uca testlerin gerçekleştirilebilmesi için kodun test mimarına teslim edilmesi gerekmektedir. Teslim öncesi, proje yöneticisi tüm birim testlerini ve hatta duman testini de gerçekleştirerek test mimarına kodu teslim etmiştir. Sıradaki başlıkta, test mimarının aktiviteleri detaylı bir biçimde anlatılacaktır.

4.3 Test Süreci

Test aşaması, dokümanların incelenmesi, senaryolarının seçimi ve test koşturumu adımlarından geçmektedir. Yazılıma ait gereksinimler ve proje yöneticisi tarafından yazılıma ait sağlanan bilgi ve belgeler detaylı bir biçimde irdelendikten sonra test senaryo seçimi ve yazımı aşamasına geçilmiştir.

Test senaryoları hem kara kutu tekniği ile (son kullanıcı bakış açısını değerlendirmek üzere), hem de beyaz kutu tekniği ile (teknik olarak kodun irdelenmesi ve yazılımcı bakış açısı ile tüm bloklara girildiğinin ve akışın doğru olduğunun tespitini yapmak üzere) belirlenmiştir. Kara kutu tekniği ile oluşturulmuş test senaryoları Ek – 1'de verilmiştir. Beyaz kutu tekniği için; tasarımın bazı durumlardan oluştuğu ve bu durumlara çeşitli eylemler sonucunda ulaşılabildiği yargısından yola çıkılarak durum geçiş diyagramı oluşturulmasına karar verilmiştir. Yazılıma ait test mimarı tarafından oluşturulmuş durum- geçiş diyagramı aşağıda Şekil 4.4'te bulunmaktadır.



Şekil 4.3 - Yazılıma Ait Durum Geçiş Diyagramı

Şekil 4.3'teki durum geçiş diyagramı incelendiğinde, yazılımın başarılı bir şekilde sonlanabilmesi için son durumun mutlaka S6 olması gerektiği görülmektedir. Olası geçişler bir tabloda özetlendiğinde Tablo 4.1'deki veriler ortaya çıkmaktadır:

Güncel Durum	**Geçiş Eylemi**	**Sonraki Durum**
Başlangıç	Yeni DICOM dosyası açma	S1
S1	Resimler arası geçiş	S2
S2	Karaciğer seçimi ve alan hesaplanması	S3
S2	Seçilen karaciğer dokusunda gürültülerin	S5

	bulunması	
S3	Resimler arası geçiş	S2
S3	Seçilen hesaplanmış alanın silinmesi	S4
S3	Hacim hesabı	S6
S4	Resimler arası geçiş	S2
S4	Hacim hesabı	S6
S5	Alan hesabı	S3
S6	Yeni DICOM dosyası açma	S1
S6	Resimler arası geçiş	S2
S6	Çıkış tuşuna basılması	Bitiş

Tablo 4.1 Olası Geçişler ve Gerekli Eylemler

Yazılıma ait durum geçiş diyagramı ve geçiş yöntemleri netleştirildikten sonra, sıra olası yolların belirlenmesi aşamasına gelmiştir. Dikkat edilmesi gereken önemli bir nokta, yazılımın başarı ile sonuçlanması için mutlaka son durum S6 olmalıdır. Bu bilgiler ışığında izlenebilecek yollar belirlenmeye başlamıştır. Yollar şu şekillerde belirlenebilir:

S1'den gidilebilecek yol yalnızca S2'dir. S2'den gidilebilecek iki yol mevcuttur: S3 ya da S5 'tir. Buradan itibaren iki ayrı yol üzerinden ilerlenebilir:

S1 → S2 → S3 ve **S1 → S2 → S5**

Sırada bulunan düğümler için gidilebilecek düğümler incelendiğinde, S3'ten gidilebilecek yollar S2, S4 ve S6; S5'ten gidilebilecek yollar ise yalnızca S3'tür. Bu durumda elde edilen yolların sayısında artış olmuştur. Yollar şu şekilde güncellenmiştir:

S1 → S2 → S3 → S2 **S1 → S2 → S5 → S3**

S1 → S2 → S3 → S4 ve

S1 → S2 → S3 → S6 (+)

S6 ile elde edilen yol, uçtan uca bulunan ilk test senaryosunu temsil etmektedir. Bu yol: **S1 → S2 → S3 → S6** yoludur. Adımlar ise yeni DICOM dosyasının açılması, DICOM dosyaları arası geçiş, karaciğer seçimi ve alan hesabı ve hacmin hesaplanması olarak görülmektedir.

Yollar güncellenmeye devam edilmektedir. S2 ile biten adım için gidilebilecek yol S3 veya S5'tir. S3 ile biten yol için gidilebilecek yol S2, S4 ve S6; S4 ile biten yol için ise S2 ve S6'dır. Bu bilgiler ışığında güncellenen yollar aşağıda belirtilmiştir:

S1 → S2 → S3 → S2 → S3

S1 → S2 → S3 → S2 → S5 ve **S1 → S2 → S5 → S3 → S2**

S1 → S2 → S3 → S4 → S2 **S1 → S2 → S5 → S3 → S4**

S1 → S2 → S3 → S4 → S6 (+) **S1 → S2 → S5 → S3 → S6 (+)**

Bu aşamada da S6 ile biten 2 adet senaryo tespit edilmiştir. Bunlar da uçtan uca test senaryoları arasına dahil edilmek üzere ayırılmıştır. Bu yollar **S1 → S2 → S3 → S4 → S6** ; yani DICOM dosyaları arası geçiş, karaciğer seçimi ve alan hesabı, seçilen alanın silinmesi ve hacmin hesaplanması; **S1 → S2 → S5 → S3 → S6**; yani DICOM dosyaları arası geçiş, karaciğer seçimi ve alan hesabı, alanın gürültülere sahip olması ve hacim hesabına bu gürültülerin dahil edilmediğinin kontrolü olarak belirlenebilir.

Aynı yöntem ile adımlar artırılarak S6'ya ulaşma hedefi devam etmektedir. Güncellenen yollar;

S1 → S2 → S3 → S2 → S3 → S2

S1 → S2 → S3 → S2 → S3 → S4 **S1 → S2 → S5 → S3 → S2 → S3**

S1 → S2 → S3 → S2 → S3 → S6 (+) **ve S1 → S2 → S5 → S3 → S2 → S5**

S1 → S2 → S3 → S2 → S5 → S3 **S1 → S2 → S5 → S3 → S4 → S2**

S1 → S2 → S3 → S4 → S2 → S3 **S1 → S2 → S5 → S3 → S4 → S6 (+)**

S1 → S2 → S3 → S4 → S2→ S5

Bu aşamada yine S6 ile biten yollar tespit edilmiş ve uçtan uca test senaryoları arasına eklenmiştir. Bu yollar; **S1 → S2 → S3 → S2 → S3 → S6**, yani DICOM dosyaları arası geçiş, karaciğer seçimi ve alan hesabı, yeni resme geçiş ve hacim hesabı ve **S1 → S2 → S5 → S3 → S4 → S6,** yani DICOM dosyaları arası geçiş, karaciğer seçimi ve alan hesabı, seçilen karaciğer dokusunda gürültülerin tespit edilmesi, alan hesabı, seçilen hesaplanmış alanın silinmesi ve hacmin hesaplanması olarak belirlenmiştir. Bu aşamadan itibaren bir başka noktaya da dikkat etmek gerekmektedir. Bazı adımlarda S2 ve S3 arasında bir döngüye girildiği görülmektedir. Bu adımlar zaten DICOM dosyaları arasındaki geçişleri temsil etmektedir ve dosya sayısı kadar geçiş sayısına sahip olabilir. Bu nedenle, örneğin 200 dosyalık bir test için, bu döngüyü 200 geçiş kadar test etmek mantıksızdır, önemli olan birden fazla dosya arasında geçiş yapılabildiğini görmektir. Bu nedenle S3'ten gidilebilecek S2 ve S2'den gidilebilecek S3 yolları için daha fazla kombinasyon yapılmamasına karar verilmiştir. Bu durumda S2 adımından gidilebilecek tek yol S5 ve S3'ten gidilebilecek yollar ise S4 ve S6 olarak indirgenmiş olmaktadır. Bu bilgiler ışığında güncellenen yollar aşağıdaki gibidir:

S1 → S2 → S3 → S2 → S3 → S2 → S5

S1 → S2 → S3 → S2 → S3 → S4 → S2

S1 → S2 → S3 → S2 → S3 → S4 → S6 (+)

S1 → S2 → S3 → S2 → S5 → S3 → S4

S1 → S2 → S3 → S2 → S5 → S3 → S6 (+)

S1 → S2 → S3 → S4 → S2 → S3 → S4

S1 → S2 → S3 → S4 → S2 → S3 → S6 (+)

S1 → S2 → S3 → S4 → S2→ S5 → S3

Bu çalışma doğrultusunda yanına "(+)" konularak işaretlenmiş 3 adet uçtan uca senaryo daha test kümesine eklenmektedir. Bunlar **S1 → S2 → S3 → S2 → S3 → S4 → S6 ,** yani yeni DICOM dosyasının seçilmesi, birkaç DICOM dosyası arasında geçiş yapılması ve alan hesaplanması, seçilen alanlardan birinin silinmesi ve hacmin hesaplanması; **S1 → S2 → S3 → S2 → S5 → S3 → S6 ,** yani yeni DICOM dosyasının seçilmesi, birkaç DICOM dosyası arasında geçiş yapılması ve alan hesaplanması, seçilen alanlarda gürültü tespit edilmesi ve hacim hesabında bu gürültülerin yer almadığının testi; **S1 → S2 → S3 → S4 → S2 → S3 → S6 ,** yani yeni DICOM dosyasının seçilmesi, seçilen doku için alan hesabının yapılması, seçilen alanın silinmesi, tekrar DICOM dosyaları arasında geçiş yapılarak doku seçilmesi, alan hesabı ve hacmin hesaplanması olarak belirlenmiştir.

Yukarıda bahsedilen aynı mantıkla S4 ile S2 arasındaki yol da birden fazla resim arasındaki geçişi temsil ettiğinden indirgenebilir. Bu durumda S4'ten gidilebilecek tek yol S6 olmaktadır.

S1 → S2 → S3 → S2 → S3 → S2 → S5 → S3

S1 → S2 → S3 → S2 → S3 → S4 → S2 → S5

S1 → S2 → S3 → S2 → S5 → S3 → S4 → S6 (+)

S1 → S2 → S3 → S4 → S2→ S5 → S3 → S4

S1 → S2 → S3 → S4 → S2→ S5 → S3 → S6 (+)

Bu güncellemeden sonra 2 adet uçtan uca test senaryosu da test kümesine dahil olmuştur. Bunlar; **S1 → S2 → S3 → S2 → S5 → S3 → S4 → S6,** yani yeni DICOM dosyasının seçimi, birkaç resim arasında geçiş, çizilen alanda gürültü tespit edilmesi ve hacim hesabı; **S1 → S2 → S3 → S4 → S2→ S5 → S3 → S6,** yani yani yeni DICOM dosyasının seçimi, birkaç resim arasında geçiş, seçilen alanın silinmesi, resimler arasında geçiş, seçilen alanda gürültü bulunması ve hacim hesabı olarak belirlenmiştir. S6 ile bitmeyen diğer yollar da S6'ya kadar ilerletilecektir. Bu şekilde üç(3) adet yol mevcuttur:

S1 → S2 → S3 → S2 → S3 → S2 → S5 → S3 → S4

S1 → S2 → S3 → S2 → S3 → S2 → S5 → S3 → S6 (+)

S1 → S2 → S3 → S2 → S3 → S4 → S2 → S5 → S3

S1 → S2 → S3 → S4 → S2→ S5 → S3 → S4 → S6 (+)

Bu güncellemeler sonrasında S6 ile biten "+" ile işaretlenmiş 2 adet daha yol belirlenmiş ve uçtan uca testler arasına eklenmiştir. Bu yollar; **S1 → S2 → S3 → S2 → S3 → S2 → S5 → S3 → S6,** yani yeni DICOM dosyasının seçimi, birkaç resim arasından seçim, seçilen resimlerde gürültü bulunması ve hacim hesaplanması; **S1 → S2 → S3 → S4 → S2→ S5 → S3 → S4 → S6,** yani yeni DICOM dosyasının seçimi, birkaç resim arasından seçim, seçilen alanın silinmesi, yeni resim seçimi, seçilen resimde gürültü bulunması, yeni resim seçimi, seçilen alanın silinmesi ve hacim hesaplanmasıdır. Bu aşamadan sonra yalnızca kalan 2 adet "+" ile işaretli olmayan yol mevcuttur ve bu yollar da ilerletilmeye devam edilmektedir. İlk yol S4 ile bittiğinden dolayı, bir sonraki adım zaten zorunlu olarak S6'dır.

S1 → S2 → S3 → S2 → S3 → S2 → S5 → S3 → S4 → S6 (+)

S3 ile biten yol ise, önceki tekrarlarda yapıldığı gibi S4 ve S6 şeklinde iki yola ayrılacaktır. S4 ile biten yolun da sonraki adımı zorunlu olarak S6 olduğundan dolayı, son durumda eklenen 2 adet uçtan uca senaryo aşağıdaki gibi olmaktadır.

S1 → S2 → S3 → S2 → S3 → S4 → S2 → S5 → S3 → S6 (+)

S1 → S2 → S3 → S2 → S3 → S4 → S2 → S5 → S3 → S4 → S6 (+)

Bu yöntem ile 15 adet uçtan uca test senaryosu çıkarılmış bulunmaktadır. Bu noktada unutulmaması gereken şey, aslında yolların S6'da son bulmadığıdır. Bu aşamadan sonra ya "çıkış" tuşuna basılarak uygulamadan çıkılır, ya yeni bir hastaya ait yeni bir DICOM dosyası görüntülenir ve yeni hacim hesaplama işlemine gidilir, ya da resimler arası geçişe devam edilerek, hacim hesabına eklemeler yapılabilir. Belirlenen 15 adet senaryoyu bu üç kombinasyon ile çoğaltmak hem zaman kaybına yol açacak hem de mantıksız olacaktır. Dolayısı ile S6 adımına gelmiş olan bu senaryoların sözgele 5 tanesini "Çıkış" tuşuna basarak sonlandırma, diğer 5 tanesini yeni bir hastaya ait görüntüler ile devam etme, kalan son 5 tanesini de yeniden hacim hesabına döndürme işlemleri şeklinde bitirilmesinde bir sakınca bulunmamaktadır. Sonuç olarak elde edilen uçtan uca test senaryo sayısında bir değişme olmayacak ama sonuç adımları güncellenecektir.

Dikkat edilmesi gerekli diğer bir nokta, uçtan uca test senaryolarını belirlemenin önemi ve zorluğudur. Tez kapsamında tek modülden oluşan nispeten küçük bir yazılım üzerinde bile zor bir süreç olan test senaryosu belirleme işlemi, modüller artırıldığında, sistemler entegre edildiğinde iyice karmaşık bir hal almakta ve test mimarına, özellikle uçtan uca akışların test edilmesi noktasında büyük bir sorumluluk yüklemesine neden olmaktadır. Bu noktada, tüm sistemler hakkında detaylı bilgiye sahip olunması, dikkatli bir şekilde sistemin, gereksinimlerin ve kodların incelenmesi, gerekli olduğu takdirde test amaçlı kod parçacıklarının yazılması vb. işlemlerin büyük bir özen ile gerçekleştirilmesi gerekmektedir. Test konusunda çalışan kimselere, diğer paydaşların önemli ölçüde destek olmaları gerekmektedir. Tez projesi kapsamında karşılaşıldığı gibi, hatanın yazılım geliştirici tarafından kabullenilmemesi çoğu zaman sürecin gereksiz yere uzamasına, zaman ve para kayıplarına yol açmasına neden olmaktadır.

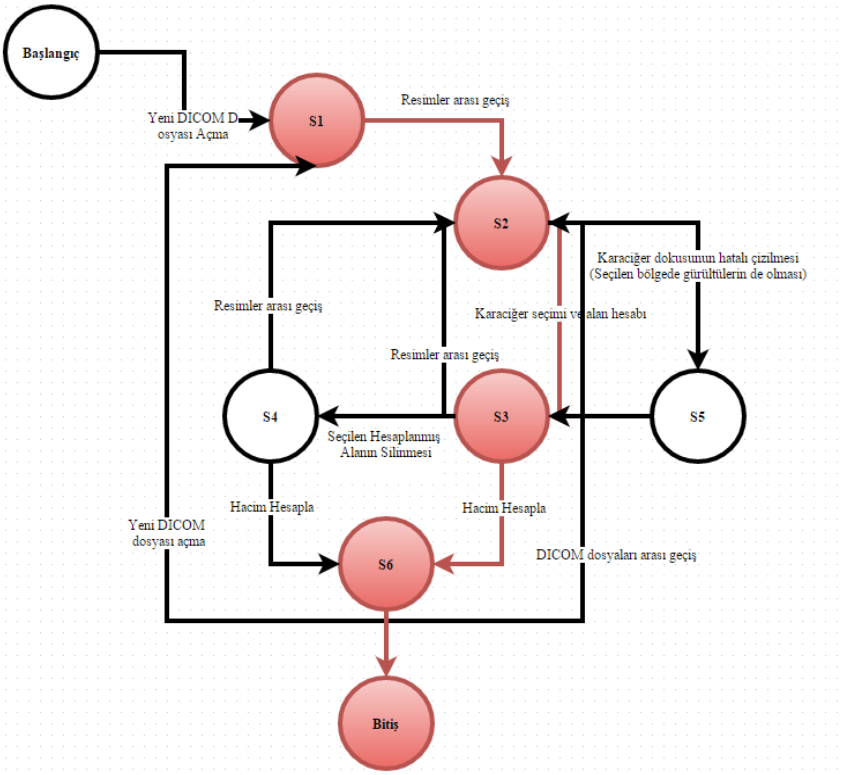
Bu bölüm altında belirlenen senaryolar eşliğinde test koşumları gerçekleştirilmiştir. Kullanılan testler: Birim Test, Bileşen Testi, Duman Testi, Sistem Testi, İlerleme Testi, Kullanılabilirlik Testi.

Bu testlerden birim test yazılım geliştirici sorumluluğundadır ve birim testleri yapılmış yazılım teste devredilmiştir. İkinci versiyonda bu aşamadan proje yöneticisi sorumlu olmuştur. Birim testler hariç diğer kullanılan test yöntemleri aşağıda açıklanmaktadır.

4.3.1 Duman testleri

Yazılım iki versiyona ait olduğundan, duman testleri iki sefer tekrarlanmak durumunda kalmıştır. Duman testlerinde amaç yazılımın temel bazı işlevleri yerine getirdiğini gördükten sonra detaylı bir şekilde incelemeye alınmasıdır. Eğer seçilen senaryo başarılı bir şekilde çalışmıyorsa, birim test aşamalarında ya da tasarım aşamalarında hata var anlamına gelmektedir ve yazılımın teste devri yapılmadan, yazılım geliştirici tarafından düzeltilmesi gerekmektedir.

Duman testi için seçilmiş olan senaryo **S1 → S2 → S3 → S6** senaryosudur. Yeni DICOM dosyası açılacak, dosya üzerinden karaciğer sınırlarını belirleyen bir alan çizilecek, hacim hesabı yapılacak ve "Çıkış" tuşuna basılacaktır. Yol aşağıda Şekil 4.4'te gösterilmiştir:



Şekil 4.4 – Duman Testi için Seçilen Yol

Nitekim, ilk versiyona ait duman testlerinin başarısızlıkla sonuçlandığı ve bu nedenle ikinci versiyona geçilmeye karar verdiğinden önceki konularda bahsedilmişti. İkinci versiyon kodu proje yöneticisi tarafından teslim edilmeden önce temel işlevler kontrol edilmiştir ancak test mimarı kodu aldıktan sonra kendi inisiyatifinde bu testleri tekrarlamıştır. İkinci versiyonda yukarıda Şekil 4.4 ile belirlenmiş temel fonksiyonalitelerin yerine geldiği görülmüştür ki bu duman testlerinin başarılı olduğu anlamına gelmektedir. Burada dikkat edilmesi gereken en önemli nokta, duman testi sırasında hacim hesabının doğru olup olmadığı kontrol edilmemektedir. Önemli olan temel bir akışın doğru bir şekilde

gerçekleştiğinin tespitidir ve bu aşamadan başarı ile geçildiğine göre, bileşen testleri ile yazılım irdelenmeye devam edilmiştir.

4.3.2 Bileşen testleri

Yazılım her ne kadar tek modülden oluşmuş da olsa, irdelendiğinde durum geçişleri, işlev fonksiyonları vb. gibi bileşenlerin olduğu görülmüştür. Bu noktada tüm bileşenlerin ayrı ayrı bağımsız bir şekilde çalışabildiğinin test edilmesi gerekmektedir. Test kapsamında değerlendirilen bileşenler: Görüntülerin ekrana yansıtılması, alan seçimi, hacim hesabının yapılışı, gürültülerin hacim hesabından çıkarılışı, görüntü kalitesinin düzenlenmesi sayılabilmektedir.

Görüntülerin ekrana yansıtılması, dosyaları okuyan ve ekranda gösteren bir bileşendir. Elde edilen örnek DICOM verileri üzerinden görüntülerin okunması ve ekrana yansıtılması görevleri yerine getirilmektedir. Bu bileşen, aynı zamanda hızı etkileyen bir bileşendir. İlk versiyona ait testlerde, herbir dosya için meta bilgileri gerek olmadığı halde defalarca okunduğundan dolayı yavaşlığa sebep olmuştur.

Alan seçimi, ekrandaki görüntüler üzerinden karaciğer dokusunun seçilmesi işlevini kontrol etmektedir. Örnek DICOM verileri üzerinden bu bileşenin testi gerçekleştirilmiştir. Ekrana görüntü geldikten sonra, fare sürüklemesi ile doku sınırlarının çizilebilir olduğu görülmüştür. Başlanılan noktada bitmeyen çizimler için ise, uygulamanın otomatik olarak çizimi başlangıç noktasına bağladığı görülmüştür.

Hacim hesabı bileşeni, seçilen kesit alanı ve uzunluğun çarpımlarından oluşan kümülatif toplamı veren bir bileşendir. Çeşitli sayıdaki kesitler ile yapılan testler sonrasında, matematiksel sonucun doğruya yakın olduğu, ve yakınsama tekniği kullanıldığından dolayı, seçim yapılan resmin ve sayısının doğruluğu kuvvetlendirici bir etken olduğu saptanmıştır.

Gürültülerin hacim hesabından çıkarılışına ait geliştirilen çözüm sonrasında karaciğere ait olmayan dokuların matematiksel sonuçtan çıkarıldığı görülmüştür. Yapılan çeşitli sayıdaki denemeler ve çeşitli kesitler üzerinde yapılan denemeler sonrasında, karaciğere ait olmayan dokunun kolaylıkla ayırt edilebildiği ve çıkan matematiksel hesabı doğruluğa yaklaştırdığı sonucuna varılmıştır.

Son olarak, görüntü kalitesi de önemli bir gereksinim olduğundan ve ayrı bir bileşen olarak düşünülerek testi gerçekleştirilmiştir. Tasarım sırasında uygulanan normalizasyon yönteminin gözle görülür bir iyileşme sağladığı sonucuna varılmıştır.

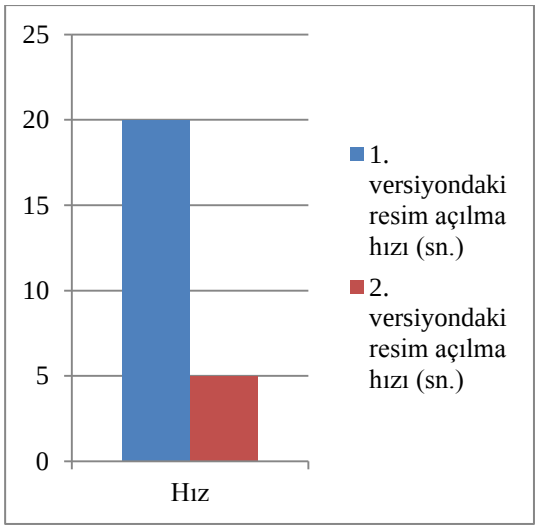
Bu şekilde, belirlenen tüm bileşenlerin ayrı ayrı testinin başarılı olduğu görülmüş ve tüm bileşenlerin birlikte doğru çalışabildiğinin testi olarak sistem testi aşamasına geçilmiştir.

4.3.3 Sistem testleri

Bileşen testleri başlığında, tüm bileşenlerin birbirlerinden bağımsız olarak çalışabildiği görüldükten sonra, tüm sisteme ait değerlendirme yapılabilmesi adına sistem testlerine başlanılmıştır. Sistem testlerinde amaç, uçtan uca tüm akışların doğru bir biçimde gerçekleştiği, yazılımda tıkanma olmadığı ve yazılımın sistem kaynaklarını doğru bir şekilde kullandığı, kapladığı yer vs. bakımlarından değerlendirilmesi anlamına gelmektedir. Uçtan uca testler için 4. Bölüm başında belirlenen tüm test senaryolarının koşturulması hedeflenmiştir, böylelikle durum diyagramındaki geçişleri temsil eden okların gerçekten belirlenen yönlere ve belirlenen şekillerde işlev gördüğünün onaylanması gerekmektedir.

Tez kapsamında incelenen yazılım, hedef bilgisayarlara konulduktan sonra herhangi bir hastahane sistemi ile ortak çalışabilir olması istenseydi bu aşamada mutlaka ve mutlaka entegrasyon testlerinin yapılması gerekli idi. Ancak tez kapsamında böyle bir gereksinim bulunmadığından, yazılıma ait sistem testleri aşamasına geçmek sakıncalı değildir. Ek 1 deki tüm kara kutu test senaryoları bu aşamada koşturulmuştur. Bunların yanı sıra belirlenmiş tüm akış senaryoları da koşturulmuş, akışın tıkandığı yerler için tekrar güncelleme talep edilmiş ve sorun kalmayıncaya dek sistem testleri sürdürülmüştür.

Daha önce yalnızca gözle görülür değerlendirmeler yapılmış olan fonksiyonel olmayan gereksinimler için –ki tez kapsamında bu madde hız olarak kabul edilmektedir- bu aşamada matematiksel bazı yargılara varılmıştır. İlk versiyonda, ilk resmin açılması yaklaşık 20 sn.lik bir süreyi kapsarken, yapılan iyileştirmeler sonrasında ilk resmin 5 sn. diğerlerinin ise saliseler seviyesinde hızlı açıldığı görülmüştür. Şekil 4.5'te bu durum ile ilgili özet bir grafik bulunmaktadır:



Şekil 4.5 Resim Açılma Hızlarının Karşılaştırılması

Yazılımın kapladığı yer bakımından ise dörtte üç oranında azalma görülmüştür.

4.3.4 İlerleme (Regresyon) testleri

Kod sürekli revize edilmiş ve son haline bu güncellemeler sonrasında kavuşmuştur. Test aşamasında mutlaka hatalar tespit edilmektedir ve her hatanın çözümü ile kod güncellenmektedir. Böylelikle test aşamasına gelen yazılımlar, tasarım – yazlım geliştirme ve test döngüsünden defalarca geçmektedirler. Bazı durumlarda analiz adımına bile geri dönülebilmektedir ve bu döngüye analiz adımı da dahil olmaktadır.

Tez kapsamında, kod üzerinde bulunan hataların güncellenmesi sürecinin dışında, daha sonradan eklenen gereksinimler olmuştur. Bu gereksinimlerin de tüm sistem akışına sorunsuz bir şekilde dahil olabilmesi gerekmektedir. Bu nedenle ilerleme ya da diğer adıyla regresyon testlerinin koşulması zorunludur.

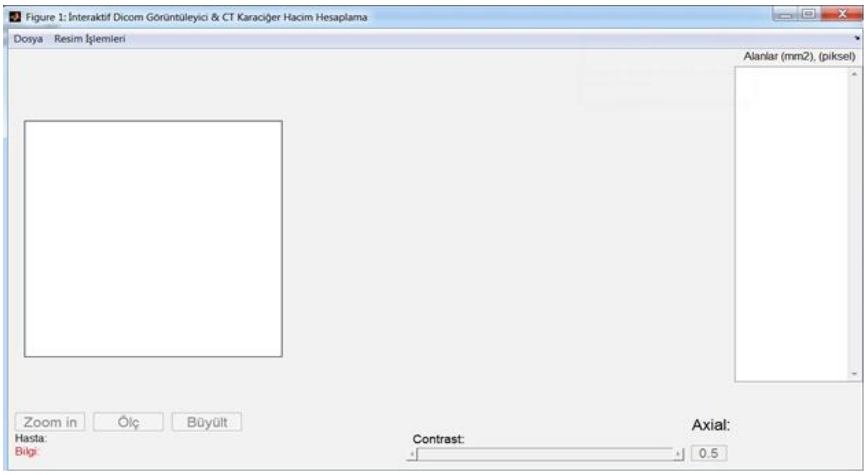
Bu noktada öncelikli ihtiyaç, etki analizinin belirlenmesidir. Bu doğrultuda, kodda güncellenen yerlerin hangi işlevleri ve gereksinimleri etkilediğini belirlemek gerekmektedir. Tez kapsamında etki analizi şu şekilde belirlenmiştir:

- Resimler arası geçiş
- Hacim hesabının doğruluğu

Yukarıda belirlenen iki madde, yazılımın ana işlevini kapsamaktadır. Bu nedenle Ek1 – 7 nolu test senaryosunun koşumu uygun bulunmuştur. Bu test koşumu sonucunda resimler arası geçişlerin başarılı bir şekilde yapılabildiği, istenilen resimler arasından alan seçimi yapılarak, ikinci versiyonda hacim hesabının da sorunsuz bir şekilde yapılabildiği görülmüştür. Sonuç olarak, yazılım ilerleme testlerinden de başarı ile geçmiştir. Bu aşamadan itibaren yazılımın belirlenen işlevleri başarı ile yerine getirdiği ve teknik sorunların tamamen giderildiği, ancak kullanıcı açısından değerlendirilmek üzere kullanılabilirlik testlerinin yapılması gerekliliğine karar verilmiştir.

4.3.5 Kullanılabilirlik testleri

Kullanılabilirlik testleri, tez kapsamında ekran tasarımı ve müşterinin kolay ve basit kullanım ile istediği tüm özelliklerin yerine getirilebilmesi hedeflenerek koşturulmuştur. Yazılım işlevsellik bakımından iki versiyonda oluşturulmuştu, ancak ekran tasarımı var olan ve yaratılan ekranlara ek güncellemeler ile son haline kavuşmuştur. Bu yolculukta ekran gelişimi ve kullanılabilirlik yorumları aşağıda yer almaktadır. İlk ekran görüntüsü Şekil 4.6'da gösterilmiştir:



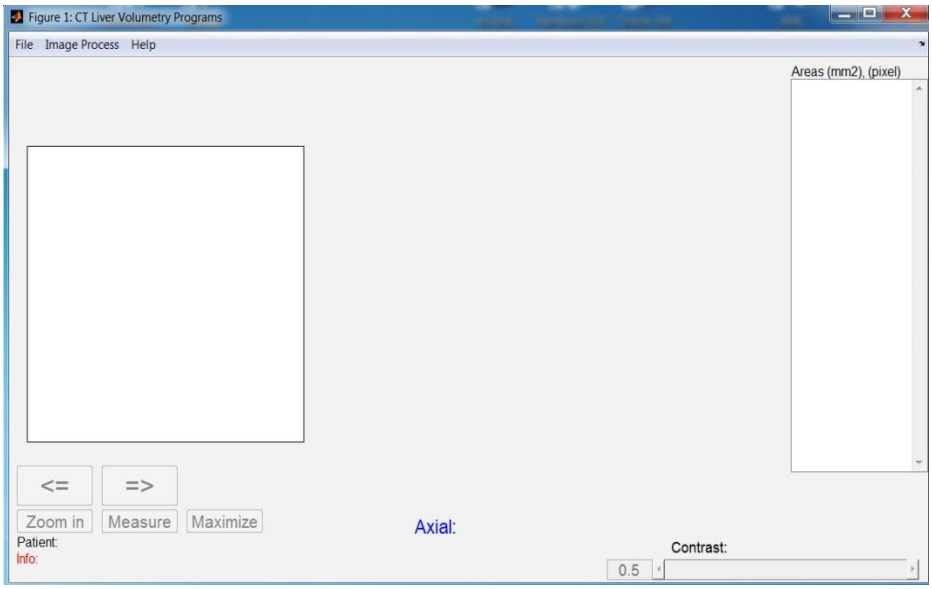
Şekil 4.6 Yazılıma ait İlk Ekran Görüntüsü

Bu ekran incelendiğinde, görüntü dosyasının seçilebildiği ve ekranda belirlenmiş alanda görüntüleneceği, yakınlaştırıp uzaklaştırma özelliğinin mevcut olduğu, ölçümün tek tuş ile yapılabildiği, istenildiği takdirde kontrasın değiştirilebileceği görülmektedir. Görselliği sağlayan bu ekran, proje yöneticisi ve test mühendisi tarafından incelenmiş ve bazı geri bildirimlerde bulunulmuştur. Bu geri bildirimler:

- Ekranda İngilizce ve Türkçe olarak iki dil kullanıldığı görülmüştür. Ekrana tek bir dilin hâkim olması gerektiği ve yazılımın uluslararası boyutta da kullanılabileceği gözönüne alınarak, hakim olan dilin İngilizce olmasına karar verilmiştir.

- Kesitlerde ilerlemeler “Scroll Bar” ile yapılıyor. Kullanım kolaylığı olması açısından kesitler arası ileri-geri işlemlerinin “<<” ve “>>” şeklindeki iki adet butonla yapılmasının daha kolay olacağı belirtilmiştir.

Bu geri bildirimler doğrultusunda, ekranda güncellemeler yapılmış ve yeni bir toplantı gerçekleştirilmiştir. Yazılıma ait güncelenmiş ekran görüntüsü aşağıda Şekil 4.7’de gösterilmiştir:



Şekil 4.7 Yazılıma ait Güncellenmiş Ekran Görüntüsü

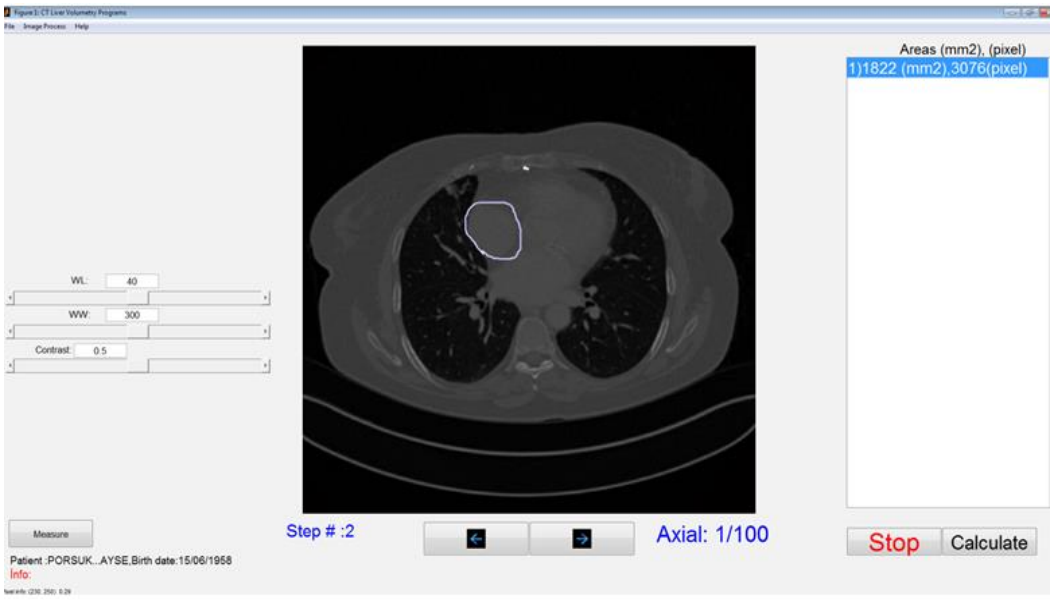
Güncellenmiş ekran incelendiğinde, daha önceki geri bildirimlerin yerine getirildiği görülmüştür. Ekrana hâkim olan dil İngilizce olarak güncellenmiş ve istenen butonlar konulmuştur. Bu aşamada artık görüntü ekranda gösterilebilecek aşamadadır ve bu bağlamda denemeler yapılmıştır. Bu aşamada, yazılım kullanılabilirliğine bağlı olarak bazı ek gereksinimler belirmiştir:

- Yazılım içerisinde, görüntü için ekranda ayrılan yer çok küçüktür. Bu ekranın büyütülmesi gerekmektedir.

- Görüntü kalitesinin artırılması için filtre araştırılması yapılması gerekmektedir.

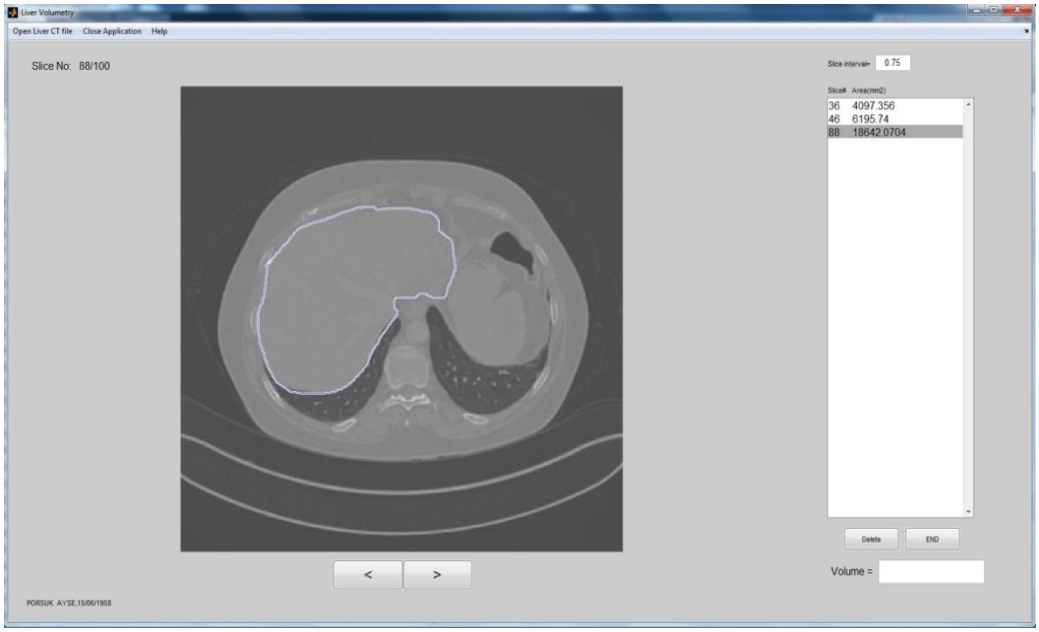
- Ekrana "Kesit Atlama" işlevinin getirilmesi gerektiği önerilmiştir.

Gereksinim sahibine güncellenerek gösterilen ekran aşağıda Şekil 4.8'de bulunmaktadır:



Şekil 4.8 - Yazılıma ait Güncellenmiş Ekran Görüntüsü - 2

Gereksinim sahibi, ekranı incelediğinde, karaciğerin, hala diğer dokulardan net bir şekilde ayrılamadığını; görüntü kalitesinin iyileştirilmesi gerektiğini ve kesit atlama değerinin değiştirilmemesi gerektiğini belirtmiştir. Gereksinim sahibi, yazılım kullanıcısının hacim hesabı yapacağı kesiti özgür olarak belirlemesi gerektiğini ifade etmiştir. Elde edilen GUI'nin son görüntüsü Şekil 4.9'da verilmiştir.



Şekil 4.9 Yazılıma Ait Son Ekran Görüntüsü

5. GENEL SONUÇLAR VE TARTIŞMA

Bu çalışmada örnek bir vaka üzerinden yazılımların test edilmesinin önemi ortaya konmuştur. Yazılımın birinci versiyonuna ait tasarım ve gerçekleştirim 1.5 yılda tamamlanmıştır. İkinci versiyonda ise hataların tespit edilerek doğru yazılımın gerçekleştirilmesi 15 günlük süreyi kapsamıştır. Yazılımın testler için, test mühendisi tarafından 20 günlük efor harcanmıştır. Bu süre içerisinde tüm senaryolar çıkarılarak uçtan uca testler gerçekleştirilmiştir. Testler sırasında 4 adet hata tespit edilmiş ve bu süreç içerisinde tüm hatalar düzeltilmiştir. Tez projesi kapsamında, yazılım testleri ile elde edilen kazanımlar önceki bölümlerde detaylı şekillerde açıklanmıştır. Uygulamanın işleyişi, akışların doğruluğu gibi uygulama özelindeki kazanımların yanı sıra, uygulamanın kapladığı alan ve performans sorunları da tespit edilerek çözüme kavuşturulmuştur.

Tez projesinde hem günlük hayatta karşılaşılan bir probleme çözüm üretmek, hem de çözüm üretilen uygulama üzerinden testin önemini vurgulamak hedeflenmiştir. Testin önemi hayat merkezli yazılımlarda daha çok ortaya çıkmaktadır. Özellikle savunma, askeri alanlarda ya da bazı taşıtlar(hızlı trenler, uçaklar) üzerindeki uygulamalardaki en ufak hatalar kritik hatalara, hayatların kaybolmasına neden olabilmektedir. Tez projesi kapsamında ise sağlık sektöründe üretilen çözümde, yazılım geliştirinin kaderine bırakılmış bir yazılım olsaydı, ve yazılım geliştirici yazılımı ve testleri kendi inisiyatifinde bitirmiş olduğu beyanı ile hareket edilmiş olsaydı, yanlış hesaplanan karaciğer hacimleri ile yanlış organ nakilleri yapılabilecek ya da yanlış tedavi yöntemleri kullanılabilecekti.

Çok yakın geçmişe kadar, yazılım testleri ve teknikleri, zaman ve para kısıtlarından dolayı pratikte sıklıkla kullanılmayan süreçler olarak kalmıştır. Ancak Dünya yazılımdaki testlerin öneminin farkına varmış ve çeşitli kalite dernekleri aracılığı ile yöntemlerin geliştirilmesi, kullanımı ve yaygınlaştırılması hedefi ile çalışmalara başlanmıştır. Test süreci ile ilgili, profesyonel hayatta kullanılmak üzre çeşitli roller belirlenmiştir: Test proje yöneticisi, test mühendisi, test mimarı, test analisti vb. bu rollerin başlıcalarıdır.

Dünya'da birçok üniversitede yazılım kalitesi ve yazılım test teknikleri, yazılım mühendisliği başlığından ayrı ve teste detaylar incelenecek şekilde ders olarak konulmuştur. Türkiye'de, yazılım test merkezleri kurulmaya başlanmış ve şirketlere, çözümlere test desteği sağlanmaya başlamıştır. Ancak bazı üniversitelerde seçmeli ders, bazı üniversitelerde ise yazılım mühendisliği dersi altında bir bölüm içerisinde bahsedilen yazılım testleri konusunda gerekli eğitimi üniversiteden alamayarak profesyonel hayata atılan çalışanlar çoğunluktadır. Bu konuda çalışmaların başlatılması, yazılım testi konusunun üniversitelerde zorunlu ders olarak konulması ya da tartışmalara öncü olması niteliğinde "test mühendisliği" bölümünün açılması hem akademik hem de profesyonel olarak birçok yeni çalışmaya yön verecek ve profesyonel hayata atılan kişiler için temel oluşturacaktır.

KAYNAKLAR DİZİNİ

Atagören, Ç., 2012, Yazılım Mühendisliği Projelerinde Hata Ölçümü, Kök Neden Analizleri ve Örnek Bir Vaka İncelemesi (Doctoral dissertation), Ankara, 91s.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... and Thomas, D., 2001, Manifesto for agile software development

Berztiss, A., 1996, Software Quality Attributes, In Software Methods for Business Reengineering, Springer New York, 73-82 pp.

Boehm, Barry W., 1988, "A spiral model of software development and enhancement."Computer 21.5 (1988): 61-72 pp.

Carlson, S. D., 2015, Testing in the healthcare Informatics environment.Mastering Informatics: A Heatlhcare Handbook for Success, 61.

Gao, J., 2000, Component testability and component testing challenges. In Proceedings of International Workshop on Component-based Software Engineering (CBSE2000, held in conjunction with the 22nd International Conference on Software Engineering (ICSE2000).

Glinz, M., 2007, On non-functional requirements. In Requirements Engineering Conference, 2007. RE'07. 15th IEEE International (pp. 21-26). IEEE.

Gorton, I., 2011, "Software quality attributes.", Essential Software Architecture, Springer Berlin Heidelberg, 23-38 pp.

ISO (Information Systems Examinations Board), "ISO/IEC 15939:2007, 2007, Systems and Software engineering – Measurement process." Geneva, ISO/IEC

ISTQB (International Software Testing Qualifications Board) , 2015, "About ISTQB", http://www.istqb.org/about-istqb/history.html", (Erişim Tarihi: 06.10.2015)

ISTQB Exam Certification, *"What is component testing?",* "http://istqbexamcertification.com/what-is-component-testing/, (Erişim tarihi: 07.06.2015)

Jorgensen, P.C., 2013, Software testing: a craftsman's approach, CRC press, US, 464 p.

Li, Z., Harman, M., & Hierons, R. M., 2007, Search algorithms for regression test case prioritization. Software Engineering, IEEE Transactions on, 33(4), 225-237 pp.

Myers, Glenford J., Corey Sandler, and Tom Badgett, 2011, The art of software testing. John Wiley & Sons

ODTÜ İnsan – Bilgisayar Etkileşimi Araştırma ve Uygulama Laboratuvarı, "Kullanılabilirlik", http://hci.cc.metu.edu.tr/kullanilabilirlik, , (Erişim Tarihi: 07.07.2015)

KAYNAKLAR DİZİNİ (Devam)

Orso, A., and Rothermel, G. , 2014, Software testing: a research travelogue (2000–2014), In Proceedings of the on Future of Software Engineering (117-132 pp.)

Patton, Ron., 2006, Software testing. Sams Pub.

Rothermel, G., & Harrold, M. J., 1997, A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology (TOSEM), 6(2), 173-210 pp.

SWEBOK, Guide to the Software Engineering Body of Knowledge 2004. IEEE, 2004.

Software Testing Class, "System Testing: What? Why? & How?", http://www.softwaretestingclass.com/system-testing-what-why-how/ , (Erişim Tarihi: 06.10.2015)

The Standish Group International Inc., 2013, Manifesto, C. H. A. O. S. Think Big, Act Small, 52 p.

Turkish Testing Board, "TTB Hakkında", http://turkishtestingboard.org/turkish/ttbhakkinda.htm, (Erişim Tarihi: 06.10.2015)

ÖZGEÇMİŞ

Ad Soyad **:** Gül Deliorman
Doğum Tarihi **:** 06.01.1988
Doğum Yeri **:** İzmir
Uyruğu **:** T.C.
Eğitim **:** **Yüksek Lisans 2012-2015:**Ege Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı
Lisans 2006-2011:Ege Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü

Çalışma Hayatı : ERICSSON Telecommunication Test Engineer(2011-2013)
ERICSSON Telecommunication Test Architect(2013-2014)
ERICSSON Telecommunication Test Team Lead (2015)

EKLER

Ek 1 Kara Kutu Test Senaryoları

Ek 2 MATLAB Tarafından Oluşturulan Kod Analiz Raporu

Ek 3 MATLAB Tarafından Oluşturulan Bağımlılık Raporu

Ek 1 – KARA KUTU TEST SENARYOLARI

Senaryo NO	Amaç	Ön koşullar	Girdiler	Beklenen Çıktılar
1	Ekrandan DICOM verisi seçilmeli ve seçilen dosyaya ait görüntünün ekranda belirdiği görülmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir.	DICOM dosyası	Ekranda görüntü belirmelidir.
2	Doktor fare yardımı ile resim üzerinde seçim yapmalıdır.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir.	DICOM dosyası	Görüntü üzerinde seçim yapılabilmelidir.
3	Resim üzerinde yapılan seçime ait alan ekrandaki listeye yazılabilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir. > Resim üzerinde doktor tarafından seçim yapılabilmelidir.	DICOM dosyası, seçim yapılan alan	Seçilen alan listede okunabilmelidir.
4	Resim üzerinde istenildiği kadar alan seçimi yapılabilmeli ve bu bilgiler listeye yazılabilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir. > Resim üzerinde doktor tarafından seçim yapılabilmelidir.	DICOM dosyası, seçim yapılan alan	Birden fazla seçilen alanın listeden okunabildiği görülmelidir.
5	Ekrandaki listeden doktor tarafından gereksiz görülen alanların silme işlemi yapılabilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir. > Resim üzerinde doktor tarafından seçim yapılabilmelidir.	DICOM dosyası, seçim yapılan alan	Silinen alanların artık listede yer almadığı görülmelidir.

6	Doktor istediği görüntü gelene kadar resimler arasında ileri ve geri tuşları ile hareket edebilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir.	DICOM dosyası, seçim yapılan alan	Resimler arası geçiş yapılabildiği görülmelidir.
7	Doktorun seçim yaptığı alanlara ait hacim hesabı başarılı bir şekilde yapılabilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir.	DICOM dosyası, seçim yapılan alan	Hacim hesabının başarılı bir şekilde yapıldığı görülmelidir.
8	Görüntüye sahip hasta bilgileri ekranda görüntülenebilmelidir.	> Örnek DICOM dosyaları test senaryo koşumuna başlanmadan önce hazır edilmelidir. > DICOM görüntüsü ekranda başarılı bir şekilde görüntülenebilmelidir.	DICOM dosyası, seçim yapılan alan	Hastaya ait bilgilerin ekrana geldiği görülmelidir.

Ek 1 Test Senaryoları

Ek 2 – MATLAB TARAFINDAN OLUŞTURULAN KOD ANALİZ RAPORU

Code Analyzer Report

Code Analyzer Report

This report displays potential errors and problems, as well as opportunities to improve your MATLAB programs (Learn More).

Rerun This Report | Run Report on Current Folder

Report for file C:\Users\Family-Two\Desktop\yazılım-2015-07-04\yazılım\ct.m

101 messages

```
49 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
67 Input argument 'hObject' might be unused, although a later one is used. Consider replacing it by ~.
67 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
78 The function 'pushbutton1_Callback' might be unused.
78 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
99 The function 'pushbutton2_Callback' might be unused.
99 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
121 The function 'listbox1_Callback' might be unused.
121 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
134 The function 'listbox1_CreateFcn' might be unused.
134 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
134 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
147 The function 'edit1_Callback' might be unused.
147 Input argument 'hObject' might be unused. If this is OK, consider replacing it by ~.
147 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
147 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
157 The function 'edit1_CreateFcn' might be unused.
157 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
157 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
171 The function 'figure1_WindowButtonDownFcn' might be unused.
171 Input argument 'hObject' might be unused. If this is OK, consider replacing it by ~.
171 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
171 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
179 The function 'figure1_ButtonDownFcn' might be unused.
179 Input argument 'hObject' might be unused. If this is OK, consider replacing it by ~.
179 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
179 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
186 The function 'figure1_KeyPressFcn' might be unused.
228 The function 'figure1_WindowButtonMotionFcn' might be unused.
228 Input argument 'hObject' might be unused. If this is OK, consider replacing it by ~.
228 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
228 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
238 The function 'Open_Callback' might be unused.
238 Input argument 'eventdata' might be unused, although a later one is used. Consider replacing it by ~.
295 The function 'Help_Callback' might be unused.
295 Input argument 'hObject' might be unused. If this is OK, consider replacing it by ~.
295 Input argument 'eventdata' might be unused. If this is OK, consider replacing it by ~.
295 Input argument 'handles' might be unused. If this is OK, consider replacing it by ~.
```

Ek 3 – MATLAB TARAFINDAN OLUŞTURULAN BAĞIMLILIK RAPORU

Report for File C:\Users\Family-Two\Desktop\yazılım-2015-07-04\yazılım\ct.m
Parent functions cannot be displayed in dependency reports for a single file.

MATLAB File List	Children (called functions)
ct	subfunction : ct_OpeningFcn subfunction : ct_OutputFcn
ct>ct_OpeningFcn	
ct>ct_OutputFcn	
ct>pushbutton1_Callback	toolbox : \images\iptformats\dicomread.m subfunction : startDragFcn
ct>pushbutton2_Callback	toolbox : \images\iptformats\dicomread.m subfunction : startDragFcn
ct>listbox1_Callback	
ct>listbox1_CreateFcn	
ct>edit1_Callback	
ct>edit1_CreateFcn	
ct>figure1_WindowButtonDownFcn	
ct>figure1_ButtonDownFcn	
ct>figure1_KeyPressFcn	toolbox : \images\iptformats\dicomread.m subfunction : startDragFcn
ct>figure1_WindowButtonMotionFcn	
ct>Open_Callback	toolbox : \images\iptformats\dicominfo.m toolbox : \images\iptformats\dicomread.m subfunction : startDragFcn
ct>Help_Callback	
ct>Open_ButtonDownFcn	
ct>pushbutton2_KeyPressFcn	
ct>figure1_WindowScrollWheelFcn	toolbox : \images\iptformats\dicomread.m subfunction : startDragFcn
ct>startDragFcn	toolbox : \images\images\bwarea.m toolbox : \images\images\im2bw.m toolbox :? Multiple class methods match imopen.m toolbox : \images\images\strel.m toolbox : \images\imuitools\imfreehand.m toolbox :? Multiple class methods match sortrows.m unknown : createMask
ct>pushbutton3_Callback	