FIRAT ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
Yazılım Mühendisliği Bölümü

YMH319 – Programlama Dilleri
Proje Uygulaması ve Dokümantasyonu

Dökümantasyon Kontrol Sistemi

Geliştirenler

İsmail Çağan 16541542

Muhammet Yasin Yıldız 16541554

Tugay Ayar 15542514

Akın Burak Yazlık 16541550

İçindekiler

1. Tanıtım

Projenin Adı: Dokümantasyon Kontrol Sistemi

1.1 Projenin Amacı

Akademik araştırmaların artması ve bilginin paylaşımındaki artan istek makale ve tez yazımlarını arttırmıştır. Bunula birlikte her alanda olduğu gibi bu alanda da sahtecilik ve intihal olaylarının yanında ister istemez gözden kaçan hatalar olmaktadır. Projemizdeki genel amaçta bu sahtecilik, intihal ve hataların önüne geçmesidir.

1.2 Projenin Kapsamı

Projemiz birçok alanda çalışan tez yazarları, makale yazarları ve denetlemek isteyen kurum ve kuruluşları kapsamaktadır.

1.3 Tanımlamalar ve Kısaltmalar

İntihal: Çalıntı, sahtecilik

Tez: Doküman

2. Planlama

2.1 Giriş

Dokümantasyon Kontrol Sistemi yapmaya başlanmadan önce planlama yapılması şarttır. Yazılım gelişim sürecinin ilk adımı planlamadır. Bu bölümde yazılımın tüm gereksinimleri planlanır. Proje planlama aşamasında yapılan işlemler;

Proje kaynaklarının belirlenmesi,

Proje maliyetinin kesinleştirilmesi,

Proje ekip yapısının oluşturulması,

Ayrıntılı proje planının yapılması,

Projenin izlenmesidir.

Proje planı tüm proje süresince kullanılıp, müşterinin isteği ve projenin gelişimi göz önünde bulundurularak güncellenecek bir belgedir. Bu belge projenin temel taşlarını oluşturur. İyi bir planlama projenin en az hata ile başarılı sonuçların elde edilmesi için en gerekli bölümdür.

Projede hataların az olması maliyeti azaltır ve aynı zamanda maksimum kazanç sağlanır. Planlama sürecinde müşterinin istekleri de oldukça önemlidir.

2.2 Projenin Plan Kapsamı

Maliyet Kestirim Dokümanı

Adı: Dokümantasyon Kontrol Sistemi

Ölçüm Parametresi	Sayı	Ağırlık	Toplam
Kullanıcı Girdi Sayısı	1	4	4
Kullanıcı Çıktı Sayısı	1	5	5
Kullanıcı Sorgu Sayısı	10	4	40
Kütük	0	10	0
Ana İşlev Nokta Sayısı(AİN)			49

Teknik Karmaşıklık Faktörü (TKF)	Puan
1. Sistem, güvenilir yedekleme ve kurtarma gerektiriyor mu?	1
2. Veri iletişimi gerektiriyor mu?	1
3. Performans kritik mi?	5
4. Sistem, mevcut ve ağır yükü olan bir işletim ortamında mı çalışacak?	3
5. Sistem, çevirim içi veri girişi gerektiriyor mu?	1
6. Çevirim içi veri girişi, bir ara işlem için birden çok ekran gerektiriyor mu?	3
7. Girdiler, çıktılar ve sorgular karmaşık mı?	1
8. Sistem içi işlemler karmaşık mı?	4
9. Tasarlanacak kod yeniden kullanılabilecek mi?	2
10. Dönüştürme ve kurulum, tasarımda dikkate alınacak mı?	3
11. Sistem birden çok yerde yerleşik farklı kurumlar için mi gerçekleştiriliyor?	5
12. Tasarlanan yazılım (projesi dahil) geliştirilecek mi?	5
Teknik Karmaşıklık Faktörü (TFK) Toplam	34

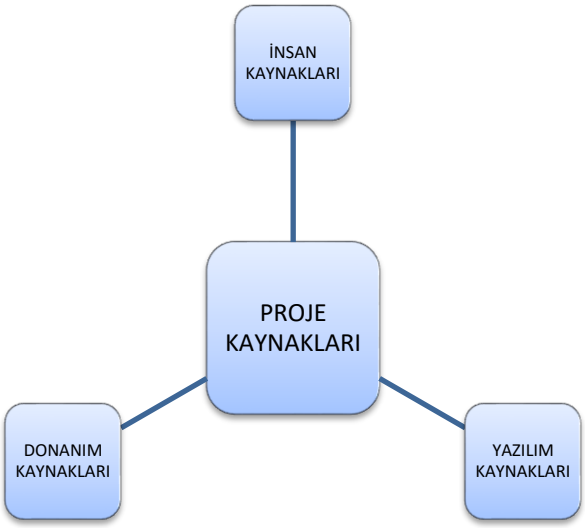
Kullanıcı Girdileri: Kullanıcının dokümanı seçmesi

Kullanıcı Çıktısı: Dokümanda bulunan hatalar **ve** intihaller

Kullanıcı Sorguları: Kullanıcı tarafından seçilen dokümanın uygulamaya yüklemesinden sonra çıktı olarak dokümanda bulunan hatalar ve intihaller

Kütükler: Projemizde veri taban bulunmamaktadır

2.2.1 Proje Kaynaklar



Şekil 2-2.1 Proje kaynakları

2.2.2 İnsan Kaynakları

*****Proje Yöneticisi

*Proje Ekibi

*Programcı

*Test Ekibi

2.2.3 Donanım Kaynakları

*****Kullanıcı bilgisayarı(Pc)

*****Monitör

*Yazıcı

2.2.4 Yazılım Kaynakları

***** Microsoft Windows İşletim Sistemi

***** Windows 10

* Python ya da C# (Nesne tabanlı programlama dilleri)

Hesaplamalar:

Ana İşlev Nokta Sayısı(AİN) = 49

Teknik Karmaşıklık Faktörü(TFK) = 34

İN = AİN * (0.65 * 0.01 * TKF) => İN= 49 *(0.65*0.01*34)

İN= 10,829

Satır Sayısı(S) = İN*30 => Satır Sayısı(S) = 10,829*30

Satır Sayısı(S) = 324,87

İş Gücü(K) = a * S^b (a, b: Her bir model için farklı katsayılar alır; S bin türünden satır sayısı.)
Ayrık Proje İçin: İş Gücü(K) = 2,4*$S^{1,05}$ => 2,4 * 0.32487 ^{1,05}

İş Gücü(K) = 0,74 => 1 Kişi Bu Projeyi Yapabilir.

Zaman(T) = c * K^d (c, d: Her bir model için farklı katsayılar alır.)

Ayrık Proje İçin: Zaman(T) = 2.5*$K^{0.38}$ =>Zaman(T) = 2,5 * $0,74^{0,38}$

Zaman(T) = 2,23 => Bu proje 2,5 ay sürecektir.

Üretkenlik = İN/Kişi - Ay

Üretkenlik = 10,829/ 4 – 2,5

Üretkenlik = 7,22

Maliyet = S / İN

Maliyet = 0.32487 / 10,829

Maliyet = 0,03

Kalite = Hatalar / İN

Kalite = 1/ 10,829

Kalite=0,092333

Satır Sayısı = İN * Dil Kodu

Satır Sayısı = 10,829* 30

Satır Sayısı = 324,87

2.3 Proje Zaman ve İş Planı

Projemizin isterilerinin tam olarak belirlenmesi aşamasında farklı kaynaklardan yararlanılmıştır. Bu kaynaklara ulaşılmasında problemler yaşanmıştır ve istenmeyen zaman kayıpları doğurmuştur. İsterilerin belirlenmesinden sonra tasarım aşamasında da zaman kayıpları yaşanabilir.

Analiz süreci keşfetme ve geliştirme sürecidir. Gereksinim analizi yapılmalı ve kullanıcının istekleri göz önüne alınmalıdır. Analiz süresi çok uzun tutulmamalıdır. İyi sonuçlar elde etmek için analiz sürecinde dikkatli olunmalıdır.

Mimari tasarım aşaması diğer aşamalara göre daha az zaman alabilir. Çünkü görsel tasarım fazla karmaşık değildir.

Yazılımın kodlanması aşaması ise geliştirici için kritiktir. Çünkü yazılımın kodlanmasında okunula bilirlik ve yazıla bilirlik çok önemlidir. Kodlama sırasında sistemin iyi bilinmesi gerekir. Oluşabilecek hatalara karşı çözüm üretmek amacı için sistemi iyi tanıyan kişiler görev almalıdır.

Kodlama aşaması sonrasında test değerlendirme aşaması gerçekleşmektedir. Oluşan sistemin belirli bir kullanıcıya sunularak test yapılır ve olumsuzluklar üzerinde değerlendirilmede bulunulur. Eksiklik varsa giderilmeye çalışılır. Test aşamasına ayrılan süre de çok olmamalıdır.

Belgelendirme aşamasında projenin detayları raporlanarak yazılım geliştirme süreci tamamlanır.

Son aşama balım aşamasıdır ve bu aşamanın süresi kesin olarak bilinmez. Sistemde zaman içinde oluşabilecek sorunlara çözümler üretilir.

#	Aşamasını Adı	Haftalar													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Giriş														
2	Plan														
3	Sistem Çözümleme														
4	Tasarım														
5	Risk Analizi														
6	Test														
7	Bakım														

2.4 Proje Ekip Yapısı

Proje ekibimiz 4 kişiden oluşmaktadır. 4 kişilik bu kadromuzda yazılım yaşam döngüsünün her bir adımını paylaşımlı olarak düzen içinde yürütülmüştür. Ekip içerisinde bulunan her birey uygulamanın test kısmında da yer almıştır

2.5 Önerilen Sistemin Teknik Tasarımı

Sistemim offline olduğu için herhangi bir internet bağlantısına gerek duyulmaz. Windows işletim sistemine sahip; laptop ve masaüstü bilgisayardan programa erişim sağlanabilir.

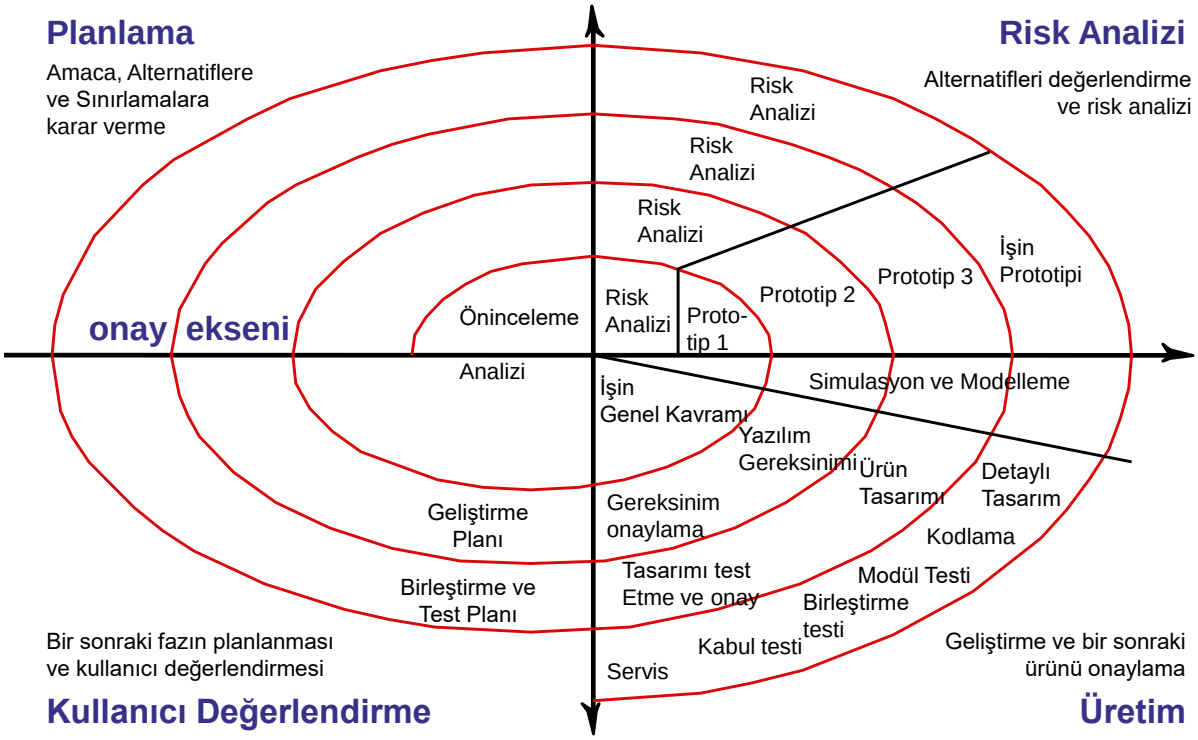
Program nesne tabanlı bir programlama dilinde gerçekleştirilecek ve kullanıcıya kolay kullanım sağlanacaktır.

2.6 Kullanılan Özel Geliştirme Araçları ve Ortamları

- C# kullanıldığı için Visual Studio 2017 ve üst versiyonları seçilebilir.
- Python 3.8
- Proje süresince Helezonik (Spiral) Model kullanılacaktır.

2.7 Proje Standartları, Yöntem ve Metodojiler

Projede Helozonik (Spiral) Model tercih edilmektedir. Bunu tercih etmemin nedeni projeyi müşterinin değerlendirme yaparak daha olumlu sonuçların elde edilmesidir. Böylelikle hem müşteri memnuniyeti sağlanır hem de program içi hata varsa daha kolay fark edinilebilir.



Şekil 2.2 Helezonik model

Helezonik model genel olarak art arda tekrarlanan dört aşamadan meydana gelir. Bunlar;

- Amaçların belirlendiği, olası seçeneklerin ve kısıtlamaların değerlendirildiği planlama aşaması
- Diğer yöntemlerde bulunmayan, risklerin tanımlandığı ve olası çözüm yöntemlerinin irdelendiği risk çözümleme aşaması

- Ürünün geliştirildiği mühendislik aşaması
- Geliştirilen ürünün müşteriyle beraber incelendiği değerlendirme aşaması

2.7.1 Helezonik Modelin Üstün Yönleri

Yazılım geliştirme süreç modellerinden; Gelişi Güzel Model, Barok Modeli, Çağlayan Modeli, V Modeli, Evrimsel Model ve Araştırma Tabanlı Model seçenekleri arasından Helezonik Modelin ayrıcalıklarını aşağıdaki gibi açıklayabiliriz.

- **Kullanıcı Katkısı:** Üretim süreci boyunca ara ürün üretme ve üretilen ara ürünün kullanıcı tarafından sınanması temeline dayanır. Yazılımı kullanacak kişinin sürece erken katılması ileride oluşabilecek istenmeyen durumları engeller.
- **Yönetici Bakışı:** Gerek proje sahibi, gerekse yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje boyunca karşılaştıkları için daha kolay izleme ve hak ediş planlaması yapılır.
- **Yazılım Geliştirici (Mühendis) Bakışı:** Yazılımın kodlanması ve sınanması daha erken başlar.

2.8 Kalite Sağlama Planı

Yazılımın kalite sağlama planında, SPICE (ISO 15504) Modeli referans alınacaktır. Bu modelin kapsamında;

- Yazılımın satın alınması,
- Yazılımın geliştirilmesi,
- Yazılımın sınanması ve iyileştirilmesi,
- Yazılımın işletimi,
- Bakım ve destek süreçleri

Model kapsamı içindedir.

2.9 Konfigürasyon Yönetim Planı

Konfigürasyon yönetimi tasarım ve üretim süreçlerindeki işlemlerin kontrol altında bulundurulması, müşterinin talep ettiği bilgilere anında ve kolay ulaşılması, hataların azaltılması ve verimliliğin artırılması önemli katkılar sağlayacaktır.

Konfigürasyon yönetimi, ürünün ömür döngüsünde ürünü oluşturan parçaların üretim hatalarına karşı kalite ve performans iyileştirmesine yönelik olarak izlenmesi; düzeltilmesi için gereken mali, zaman ve iş gücü kaynaklarına önemli ölçüde azaltılmasına katkı sağlamaktadır.

2.10 Kaynak Yönetim Planı

Projede çalışan ekibin Helezonik Modelin her aşamasında olması, her bölümün teker teker gözlemlenmesi, ölçülendirilmesi ve test edilmeye gerek duyulduğundan dolayı Kaynak Yönetim Planı çok önemlidir. Kaynak Yönetim Planında yapılması gerekenler şunlardır;

- Planlamada temel hedeflerin açık anlaşılır bir şekilde belirlenmesi
- Risk yönetimi için temel hedeflerin belirlenmesi
- Planlamadan oyuncu ve müşteri değerlendirmelerine kadar olan bölümde(Planlama, Risk Yönetimi, Üretim, Oyuncu ve Müşteri Değerlendirmesi) uygun bütçenin çıkarılması
- Kabul edilmiş risklerin belirlenmesi
- Risk yönetim işlemleri, yönetim ve tekniklerinin tanımlanması
- Proje ile ilgili gerekli önceliklerin belirlenmesi

2.11 Eğitim Planı

Uygulamanın kurulum sonrasında kullanıcıyı eğitici yönergeler bulunacaktır. Kullanıcı ara yüzü sade ve anlaşılır olduğundan kısa bir eğitim ile başlayabilirler.

2.12 Test Planı

Yapılan program tüm Windows işletim sisteminde çalışabilmektedir. Yazılımı müşteri ile deneyerek programın düzgün çalışıp çalışmadığını test etmiş olacağız.

2.13 Bakım Planı

Sistem offline bir sistem olduğu için fazla bakım ihtiyacı duymaz. Nitekim gereksinimlerin artması yeni güncellemelere neden olabilir. Bunun olması durumunda uygulamanın yazılımını geliştiren şirket tarafından güncel sürümü yayınlanacaktır.

3. Sistem Çözümleme

3.1 Mevcut Sistem İncelemesi

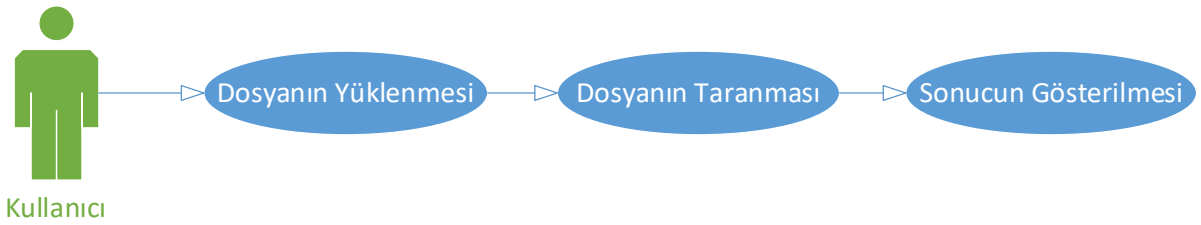
Projemizin planlama aşamasında, var olan mevcut sistemi inceledik, bu sayede mevcut sistemdeki açıkları, hataları ve eksiklikleri tespit ettik. Geliştireceğimiz projede bunları dikkate alarak daha verimli ve güvenilir ürün ortaya koyacağız

3.2 Var olan ve Gereksenen Sistem

3.2.1 Örgüt Yapısı

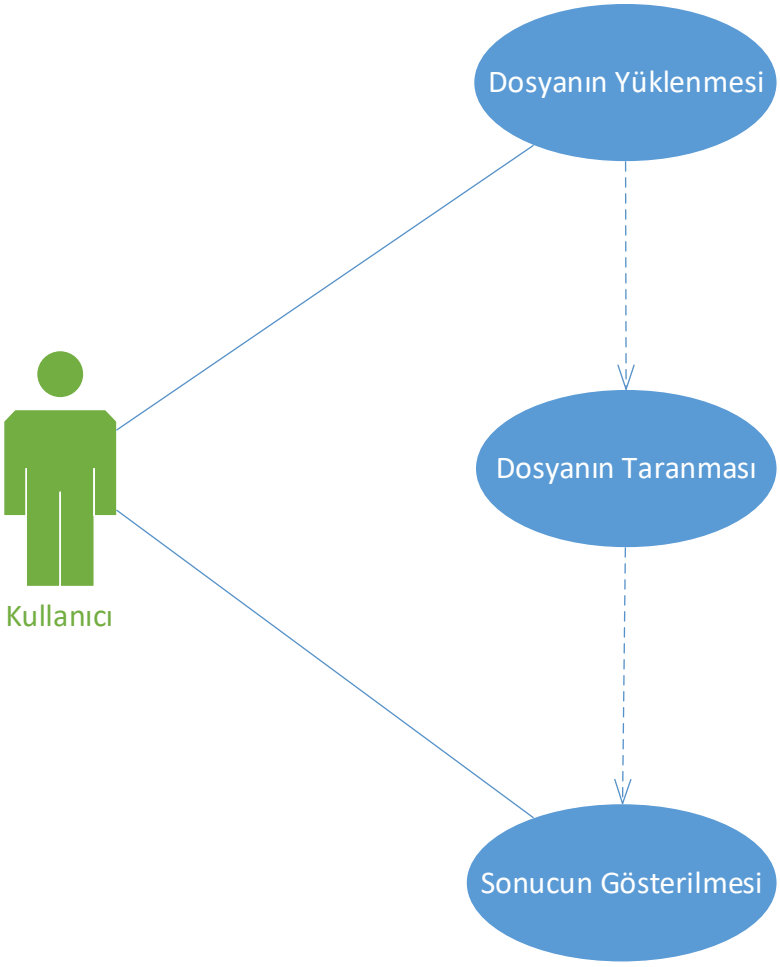
- Dokümanı programa yüklemesi
- Dokümanı taratması
    - Sonuç olarak bulunan hataların ve intihallerin ayrı bir ara yüzde gösterilmesi

3.2.2 İşlevsel Model



Şekil 3.1 İşlevsel model

3.2.2.1 Use Case Diyagramı



Şekil 3.2 Use Case Diyagramı

3.2.3 Bilgi Sistemleri/Nesneler

3.2.3.1 Sınıf Diyagramları

Şekil 3.3 Sınıf diyagraımı

3.3 Arayüz (Modül) Gerekleri

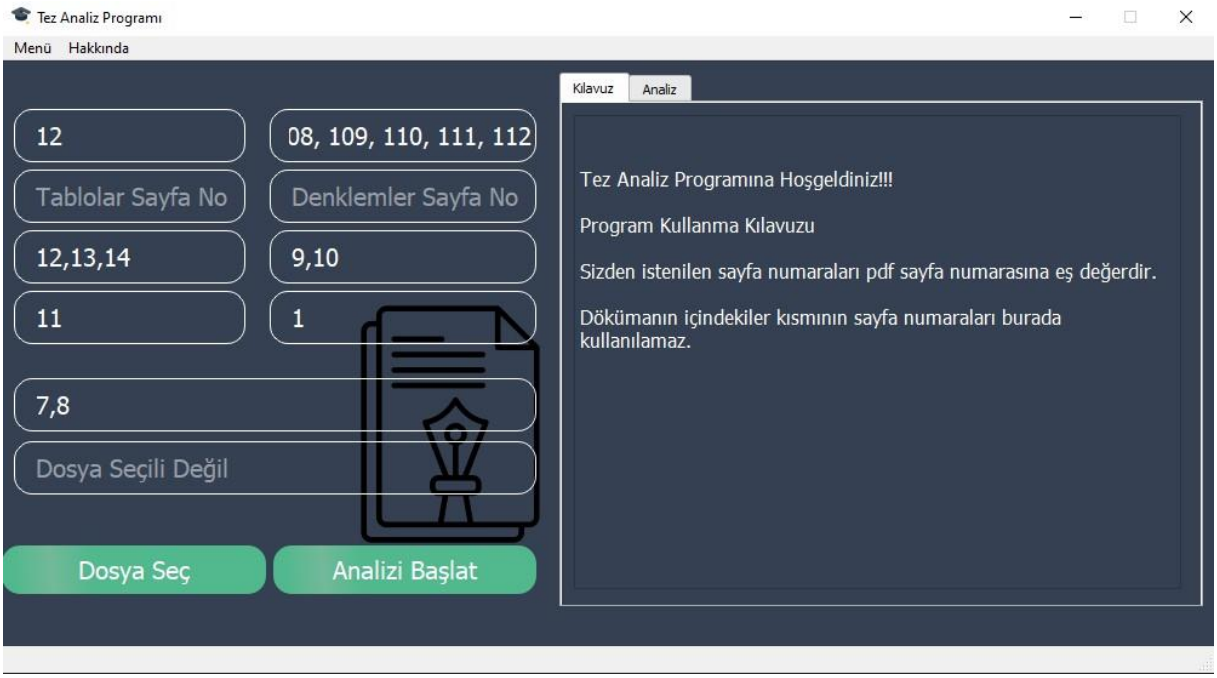
3.3.1 Yazılım Arayüzü

Projemiz genel kitleye hitap ettiği için özel bir kullanıcı adı ve şifre gerektirmemektedir, arayüz olarak 2 ara yüzümüz bulunmaktadır. İlk ara yüzümüzde PDF'in görüntüleneceği alan, ekle butonu, PDF'in adını görmemiz için label ve taratma butonu bulunmaktadır.

İkinci ara yüzümüzde taranmış halde gözüken PDF, hata sayısını görmemiz için gereken alan, tekrardan PDF eklemek için bir önceki sayfaya geçiş butonu ve tamamen kapatmak için çıkış butonu yer almaktadır.

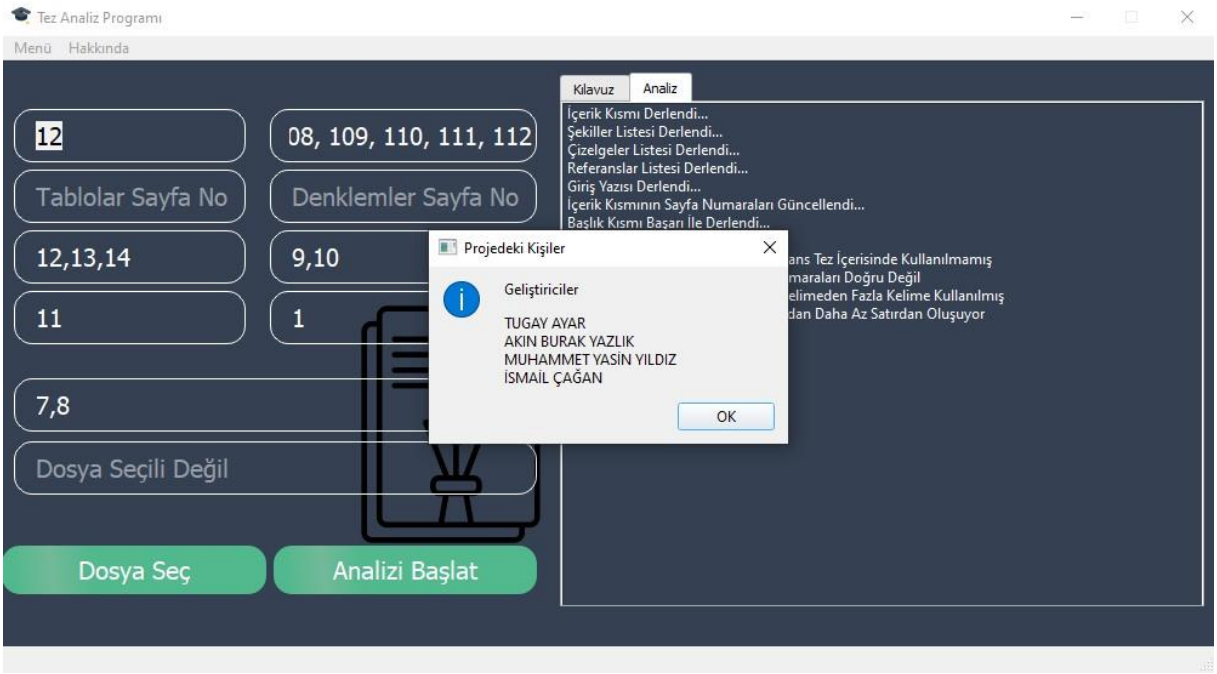
3.3.2 Kullanıcı Arayüzü

Şekil 3.4 Giriş ekranı



Şekil 3.5 Kılavuz sekmesi

Şekil 3.6 Analiz sekmesi



Şekil 3.7 Geliştirici ekip

3.4 Belgeleme Gerekleri

3.4.1 Geliştirme Sürecinin Belgelendirmesi

Geliştirme süreci belirtilmiş olan kriterler doğrultusunda gerçekleştirilmiştir. Bu geliştirme süreçleri (V Süreç Modeli, Helezonik Model, Evrensel Geliştirme Süreç Modeli, Araştırma Tabanlı Süreç Modeli) arasından Helezonik Model kullanılmıştır.

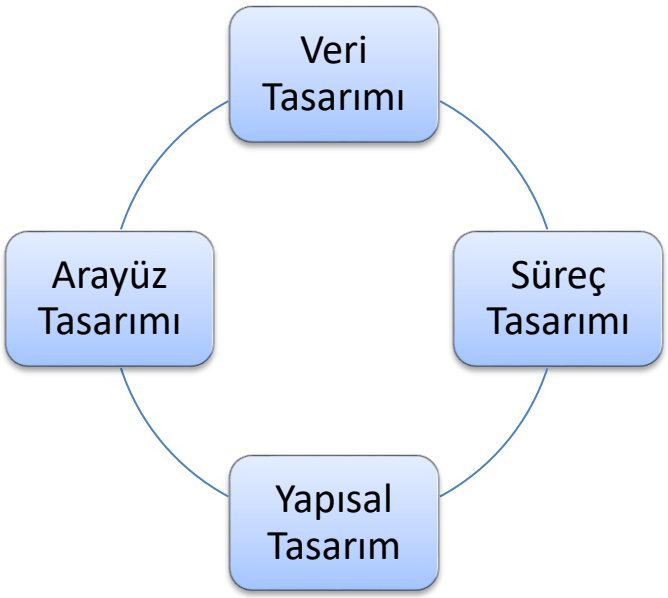
4. Sistem Tasarımı

4.1 Genel Tasarım Bilgileri

Şekil 4.1

4.1.1 Genel Sistem Tasarımı

Temel hedef, yazılım yapısına ait diyagramları oluşturmaktır. Veri akış diyagramlarından yola çıkarak sistemdeki işlerin işleyişinin tanımlanmasıdır.



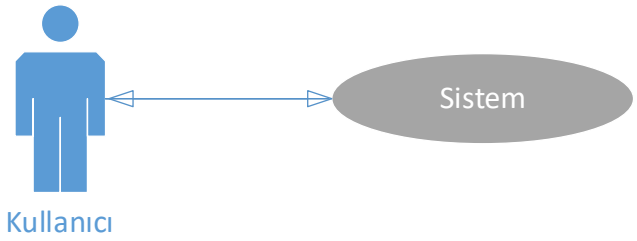
Şekil 4.2

4.1.2 Varsayımlar ve Kısaltmalar

- Proje bitmesi gereken zamanda bitecektir
- Proje sonuna kadar bir ekip çalışması olacaktır
- Projenin her aşamasında proje yöneticiyle toplantılar yapılacaktır
- Proje sonuna kadar eklenen her yeni bilgiden tüm ekibin haberdar olması gerekmektedir
- Projenin kısa zamanda ve düşük maliyette bitirilmesi amaçlanmaktadır

- **Sistem;**
  - Sistemi sahip olan herkes kullanabilmektedir.
  - Sistemi kullanan kişi dokümanı ekleyip, istediği gibi tarama yapabilmektedir.

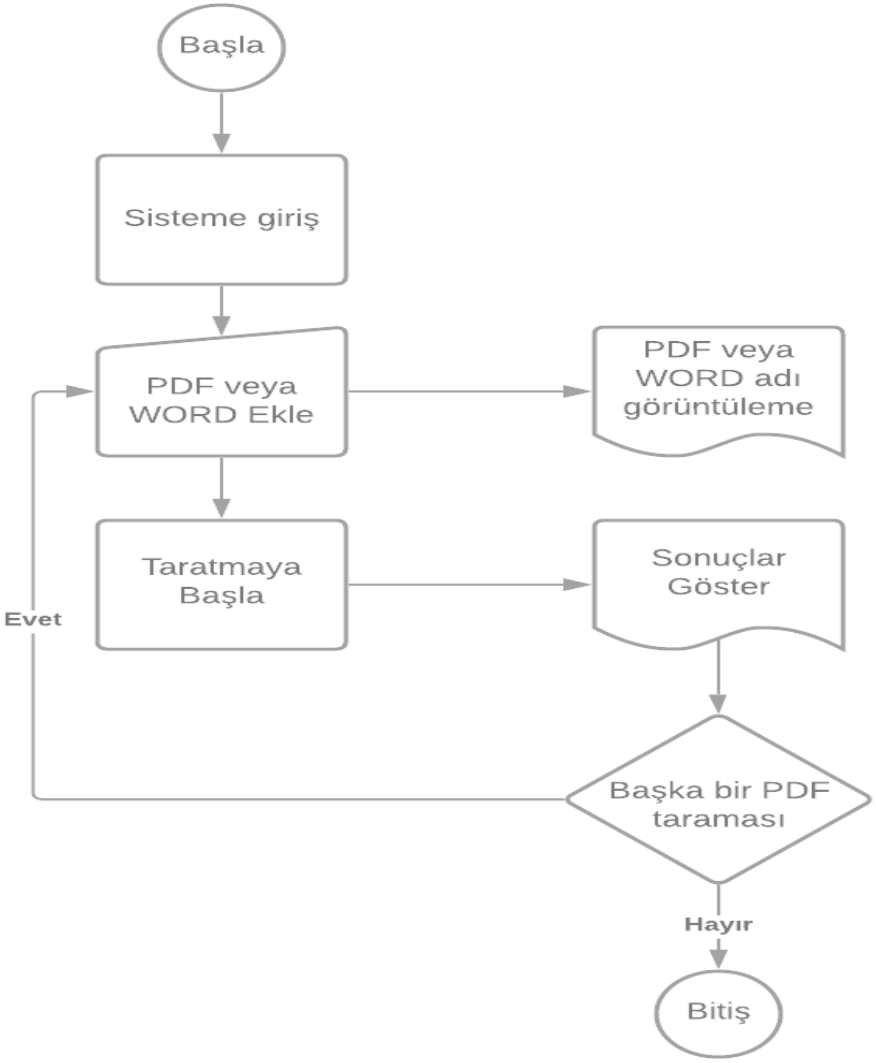
4.1.3 Sistem Mimarisi



Şekil 4.3 Sistem

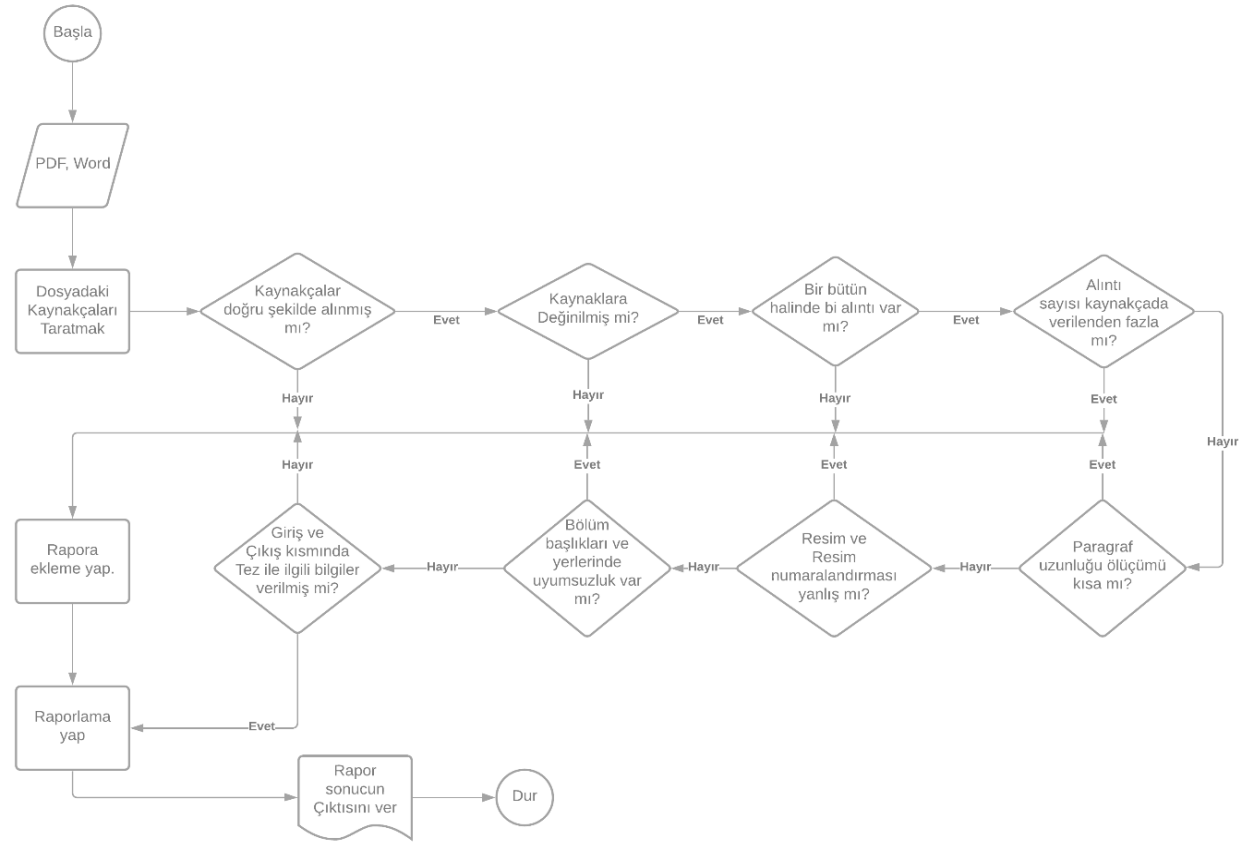
4.1.3.1 Akış Diyagramları

Arayüz Akış Diyagramı:



Şekil 4.4 Arayüz Akış Diyagramı

Algoritma Akış Diyagramı:

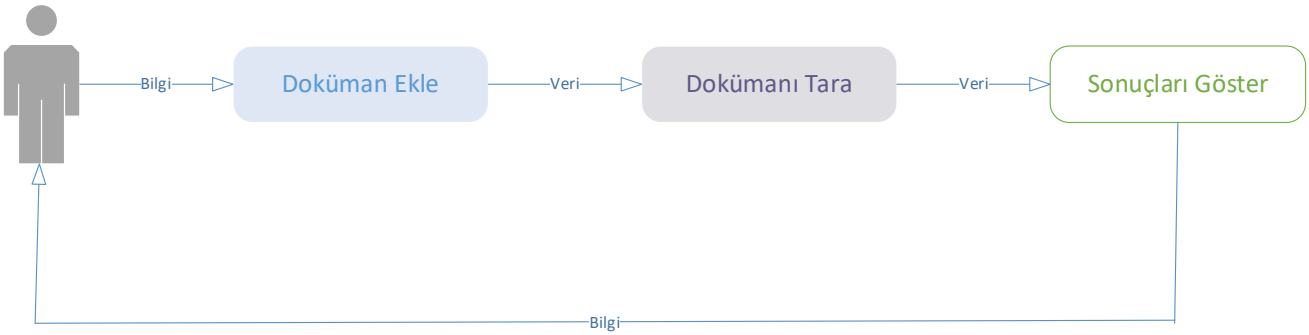


Şekil 4.5 Algoritma Akış diyagramı

4.1.4 Dış Arabirimler

4.1.4.1 Kullanıcı Arabirimleri

- Programı Çalıştır
- Doküman Ekle
- Dokümanı Tara
- Sonucu Göster



Şekil 4.6 Kullanıcı Arabirimi

4.1.5 Testler

Proje yapımı boyunca oluşabilecek her türlü sorun test edilecek ve aynı zamanda yazılımın kendi sistem ve donanım hatalarına dair raporları da test aşamasına etki edecektir.

4.1.6 Performans

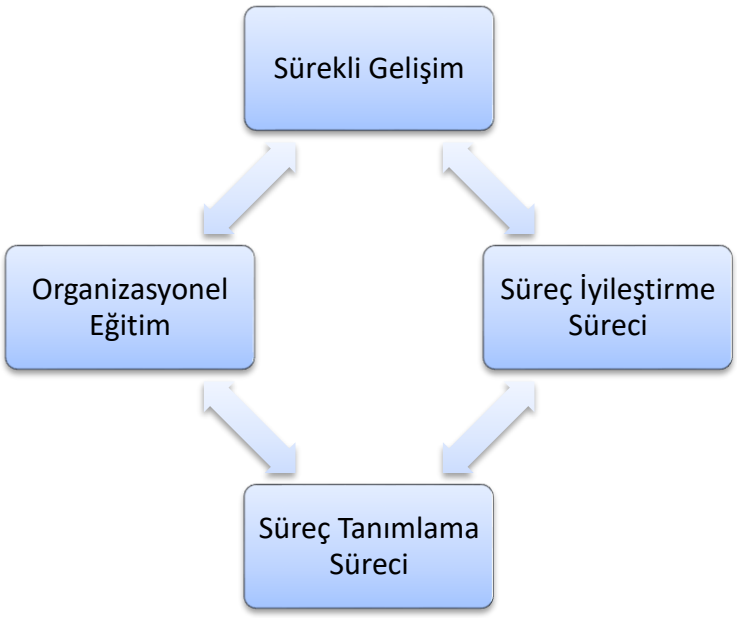
Yazılımın belirlenen süreç modeline(Helezonik Model) göre yönetilmesi, gerekli dokümantasyonun hazırlanmasında ve plana bağlı yazılımın gerçekleştirilmesi performans değerlerine öncülük etmektedir. Yapılanan uygulamanın performansının daha iyi olması için gerekli bilgi test aşamasında detaylı bir şekilde aktarılmıştı.

4.2 Süreç Tasarımı

4.2.1 Genel Tasarım

Yazılım geliştirme süreçleri sürekli yaşayan ve iyileştirilen süreçler olmak zorundadırlar. Dolayısı ile, bir süreç yönetim mekanizması kurulması zorunludur. Süreç yönetimi, süreç tanımlama, organizasyonel eğitim, süreç iyileştirme süreçleri şeklinde organize edilebilir.

- **Süreç Tanımlama Süreci:** Süreç Tanımlama Sürecinin amacı, iş süreç için tutarlı, tekrarlanabilir, performansını destekleyecek süreç kütüphanesi oluşturacaktır. Bu süreç için, süreç tanımlama formu, süreç iş çıktıları hazırlama formu gibi şablonlar tanımlanmalıdır.
- **Organizasyonel Eğitim:** Çalışanların kendilerine verilen rolleri verimli ve etkin bir şekilde gerçekleştirmelerini sağlamaktır.
- **Süreç İyileştirme Süreci:** Organizasyon süreçleri ve süreç varlıkları ile ilgili kuvvetli ve zayıf alanlar baz alınarak; organizasyon süreç iyileştirme aktivitelerinin planlanması ve uygulanmasıdır.



Şekil 4.7 Süreç Tasarım

4.2.2 Modüller

4.2.2.1 Kullanıcı Modülü

4.2.2.1.1 İşlev
Kullanıcı bilgisayar dosyalarından doküman seçer ve taratır.

5. Sistem Gerçekleştirimi

5.1 Giriş
Ürünümüz akademisyenlerin tez gibi dokümanlar üstünde araştırma yapmasını sağlamak amacıyla üretildiği için kişiye özel kullanıcı adı ve şifre gerektirmemektedir ve bu amaç doğrultusunda kullanımı pratik olup, olabildiğince basitleştirilmiştir.

İki arayüz bulunmaktadır, ilk arayümüzde dökümanı ekleme ve taratma seçeneğimizin yanında yüklenen dokümanın orijinalini ve ismini görebilmekteyiz.

İkinci arayüzümüzde hatalarıyla beraber dokümanın çıktısı, hata sayısı, yeni doküman ekleme seçeneği ve çıkış seçeneği bulunmaktadır.

5.2 Yazılım Geliştirme Ortamları
Yazılım geliştirme ortamı, tasarım sonunda üretilen fiziksel modelin, bilgisayar ortamında çalıştırılabilmesi için gerekli olan;

- Projemizi geliştirmek için gerekli olan çalışma ortamı(discord, trello, github, …) ve gerekli olan araç ve gerekçeler(PC, Visio, Microsoft Word, Visual Studio, Adobe Reader eklentisi, …)
- Projeyi gerçekleştirecek elemanların belirlenmesi ve aralarındaki iletişimlerinin sağlanması
- Gerekli dokümanların hazırlanması(planlama, iş-akış diyagramı, zaman diyagramı, use-case diyagramı, algoritma akış diyagramı, …)
- Projenin gerçekleştirim aşaması(kodların yazılması)
- Gerçekleştirilen projenin test aşaması

Bileşenlerinden oluşur. Günümüzde söz konusu bileşenler oldukça farklılık ve çeşitlilik göstermekte ve teknolojinin değişimine uygun olarak gelişmektedir.

5.2.1 Veri Tabanı Yönetim Sistemleri

Projemizde veri tabanı bulunmamaktadır.

5.2.1.1 Hazır Program Kütüphane Dosyaları

Hemen hemen tüm programlama platformlarının kendilerine özgü hazır program kütüphaneleri bulunmaktadır. Söz konusu kütüphaneler;

- PDF ve Word kullanabilmek için geliştirmiş kütüphane dosyaları ve eklentiler(Visual Studio için)
- IO(dosyaların eklenmesi için) kütüphanesi

Söz konusu yazılımlar program geliştirme hızını oldukça arttırmaktadır. Günümüzde bu tür kütüphanelerin edinilmesi, internet sayesinde oldukça kolaylaşmıştır.

5.2.1.2 Case Araç ve Ortamları

Günümüzde bilgisayar destekli yazılım geliştirme ortamları(CASE) oldukça gelişmiş durumdadır. CASE araçları, yazılım üretiminin hemen her aşamasında (planlama – çözümleme – tasarım – gerçekleştirim – sınama) üretilen bilgi ya da belgelerin bilgisayar ortamında saklanmasını, bu yolla kolay erişilebilir ve yönetilebilir olmasını olanaklı kılar. Hemen hemen tüm CASE araçları, belirli bir standarda uygun olarak geliştirme yapar. Bu yolla yapılan üretimin yüksek kalitede olması sağlanır.

5.3 Kodlama Stili

Hangi platformda geliştirilirse geliştirilsin, yazılımın belirli bir düzende kodlaması yazılım yaşam döngüsünün uygulama boyutu açısından oldukça önem taşımaktadır. Yazılım ya da bilgi sistemleri, doğaları gereği durağan değildir. Yazılımın gerektirdiği güncel değişikliklerin ilgili yazılıma da aktarılması gerekir. Bu nedenle yazılımın kodlarına zaman zaman başvurmak, yeni kod parçaları eklemek ya da var olan kodlarda değişiklikler yapmak yazılım yaşam döngüsünün işletimsel boyutunun en önemli işlevlerinden biridir. “Bakım Programcısı” kavramı bu tür gereksinimlerden doğmuştur. Bakım programcısının temel görevleri; var olan yazılıma ilişkin kodlar dâhil üretilmiş tüm bilgi ve belgeleri incelemek ve yazılım üzerinde değişiklikler yapmak biçiminde özetlenebilir. Kolay okunabilir ve anlaşılabilir kodları olmayan yazılımın bakımı oldukça zorlaşır ve büyük maliyetlere ulaşır.

Yazılım Kod stili konusunda herhangi bir kabul edilmiş standart bulunmamaktadır. Bu konuda; yazılım geliştiren ekiplerin, kodlama aşamasına başlamadan kodların düzeni konusundaki standartlarını ya da kurallarını geliştirmeleri ve bu kuralların uygulamaya geçmesini sağlamaları önerilmektedir.

5.3.1 Açıklama Satırları

Bir program parçasını anlaşılabilir kılan en önemli unsurlardan biri, bu program kesiminde içerilen açıklama satırlarıdır. Her bir program modülü içerisine;

- Modülün başlangıç açıklamaları
- Modül kod açıklamaları
- Boşluk satırları

Eklenmelidir.

Bir programın karmaşıklığını arttıran en önemli bileşenler, program içerisinde kullanılan denetim yapılarıdır (koşullu deyimler, döngü deyimleri). Bu tür deyimlerin hemen öncesinde bu denetim işleminin açıklamasını ve bu deyimde olabilecek olağan dışı durumları içeren kod açıklama satırları koyulmalıdır.

Program kodu içerisinde; kodların görüne bilirliğini ve anlaşıla bilirliğini arttırmak amacıyla yeterli oranda boşluk satırı koyulmalıdır. Boşluk satırları programın etkinliğine (bellek kullanımı, performans, vb.) hiçbir olumsuz etki yapmaz. Bu nedenle gerek kod satırları arasına uygun olarak, gerekse bir satırın kendi içerisinde yeterli boşluk bırakılması önerilmektedir.

5.3.2 Kod Biçimlemesi

Kod biçimlenmesi, açıklama satırlarına olan ihtiyacı azaltır. Kod biçimlemesinde önemli olan az satır değil kodun okunabilirliğidir.

Sistemimizde programın okunabilirliğini artırmak ve anlaşıla bilirliğini kolaylaştırmak amacıyla açıklama satırlarının kullanımının yanı sıra belirli bir kod yazım düzeni de kullanılmıştır. Örneğin;

- Değişken tanımlamalarda, değişkenin kullanılacağı alan isminin değişken isim olarak atanması.
- Nesne tanımlamalarda nesneye sınıfın isminin verilmesi

5.3.3 Anlamlı İsimlendirme

Anlamlı isimlendirme de kullanılan önemli teknik, hangi değişkenlere hangi modüllerle ilgili olduklarının belirtilerek adlandırılmasıdır. Kodların okunabilirliğini ve anlaşıla bilirliğini sağlayan önemli unsurlardan biri de kullanılan ve kullanıcı tarafından belirlenen belirteçlerin (değişken adları, kütük adları, işlev adları, yordam adları, vb.) anlamlı olarak isimlendirilmesidir.

İsimlendirme yöntemi uygulamayı geliştirenler tarafından çözümleme-tasarım aşamalarında belirlenmeli ve gerçekleştirim aşamasında uygulanmalıdır. Kod incelenirken en azından ilk bakışta bilgilerin yalnızca değişken adına bakılarak anlaşılabilmelidir. Anlamlı isimlendirmede belirtilmesi gerekenler;

- Hangi değişkenin hangi verileri kapsadığı hangi tablolardan kullanıldığı
- Hangi değişkenin dışarıdan girildiği
- Hangi değişkenin çıktı olarak gösterileceği
- Hangi değişkenler sadece ara değişkenler olarak kullanılacağı
- Hangi değişkenlerin yazıcı çıktılarında görülmek üzere kullanıldıkları
- Hangi değişkenlerin yalnızca ilgili kod içerisinde ara değişkenler olarak kullanıldıkları
- Hangi değişkenlerin klavye ve ekran aracılığı ile değer aldıkları

5.3.4 Yapısal Programlama Yapıları (YPY)

Yapısal Programlama Yapıları, okunabilirlik ve anlaşıla bilirlik bakımından önemlidir. Üç dala ayrılan YPY, yazılımın kodlanması sırasında sık sık kullanılacaktır. Program kodlarının; okunabilirlik, anlaşıla bilirlik, bakım kolaylığı gibi kalite etmenlerinin sağlanması ve program karmaşıklığının azaltılması amacıyla YPY kullanılarak yazılması önemlidir. Yapısal Programlama Yapıları temelde; içinde "84 oto" deyimi bulunmayan, "tek giriş ve tek çıkış" öbeklerden oluşan yapılardır. Teorik olarak herhangi bir bilgisayar programının, yalnızca Yapısal Programlama Yapıları kullanılarak yazılabileceği kanıtlanmıştır.

Bu yapıların her biri "tek giriş ve tek çıkış" yapılardır. Programlar, yalnızca bu yapılar kullanılarak kodlandığında ve uygun açıklama satırları ile desteklendiğinde; program bakımı kolaylaşmakta ve geliştirme hızlanmaktadır.

Üç temel Yapısal Programlama Yapısı şunlardır;

- **Ardışık İşlem Yapıları:** Herhangi bir koşula bağlı olmaksızın birbiri ardına uygulanması gereken işlemler olarak tanımlanır. Hemen her türlü programlama dilinde bulunan; aritmetik işlem deyimleri, okuma/yazma deyimleri bu tür yapılara örnek olarak verilebilir.
- **Koşullu İşlem Yapıları:** 70'li yılların ortalarından sonra gelen programlama dillerinin hemen hepsinde, bu yapılar doğrudan desteklenmektedir. Üç tür Koşullu İşlem Yapısı bulunmaktadır;
  - Tek koşullu işlem yapısı → if-then
  - İki koşullu işlem yapısı → if-then-else
  - Çok koşullu işlem yapısı → case-when
- **Döngü Yapıları:** Belirli bir koşula bağlı olarak ya da belirli sayıda, bir veya daha çok yinelenecek işlemler için kullanılan yapılardır. Temelde üç tür Döngü Yapısı bulunmaktadır;
  - Belirli sayıda yinelenecek işlemler için kullanılan yapılar → for
  - Bir koşula bağlı olarak, sıfır ya da birden çok kez yinelenecek işlemler için kullanılan yapılar → while-end
  - Bir koşula bağlı olarak, bir ya da birden çok kez yinelenecek işlemler için kullanılan yapılar → repeat-until

5.4 Program Karmaşıklığı

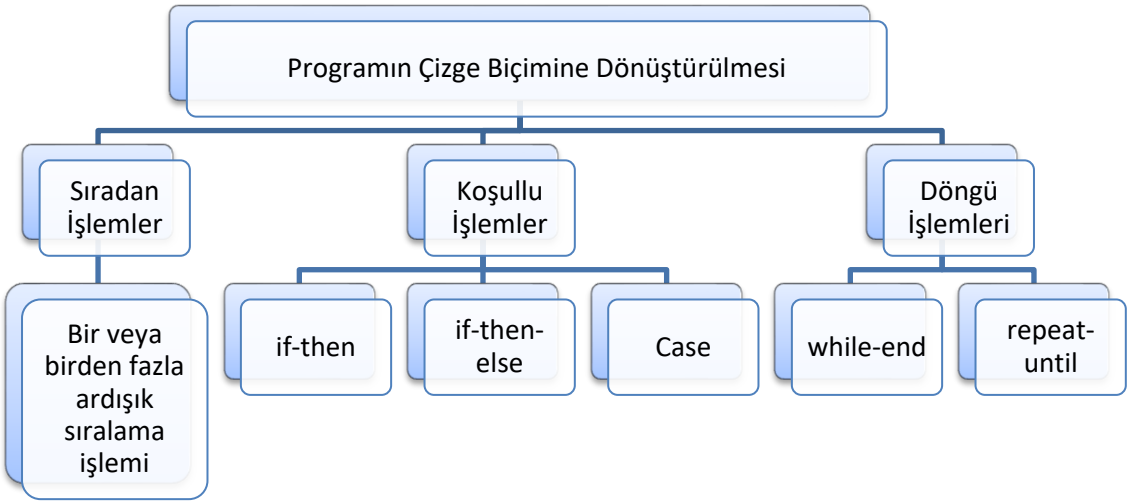
Program karmaşıklığını ölçmek için bir çok teorik model geliştirilmiştir. Bu modellerin en eskisi ve yol göstericisi McCabe karmaşıklık ölçütüdür. Söz konusu ölçüt 1976 yılında McCabe tarafından geliştirilmiştir. Bu konuda geliştirilen diğer ölçülerin çoğu, bu ölçütten esinlenmiştir.

McCabe ölçütü, bir programda kullanılan "koşul" deyimlerinin program karmaşıklığını etkileyen en önemli unsur olduğu esasına dayanır ve iki aşamada uygulanır;

1. Programın çizge biçimine dönüştürülmesi
2. McCabe karmaşıklık ölçütü hesaplanması

Şekil 5.1

5.4.1 Programın Çizge Biçimine Dönüştürülmesi



Şekil 5.2 Program Çizge

5.4.2 McCabe Karmaşıklık Ölçütü Hesaplama

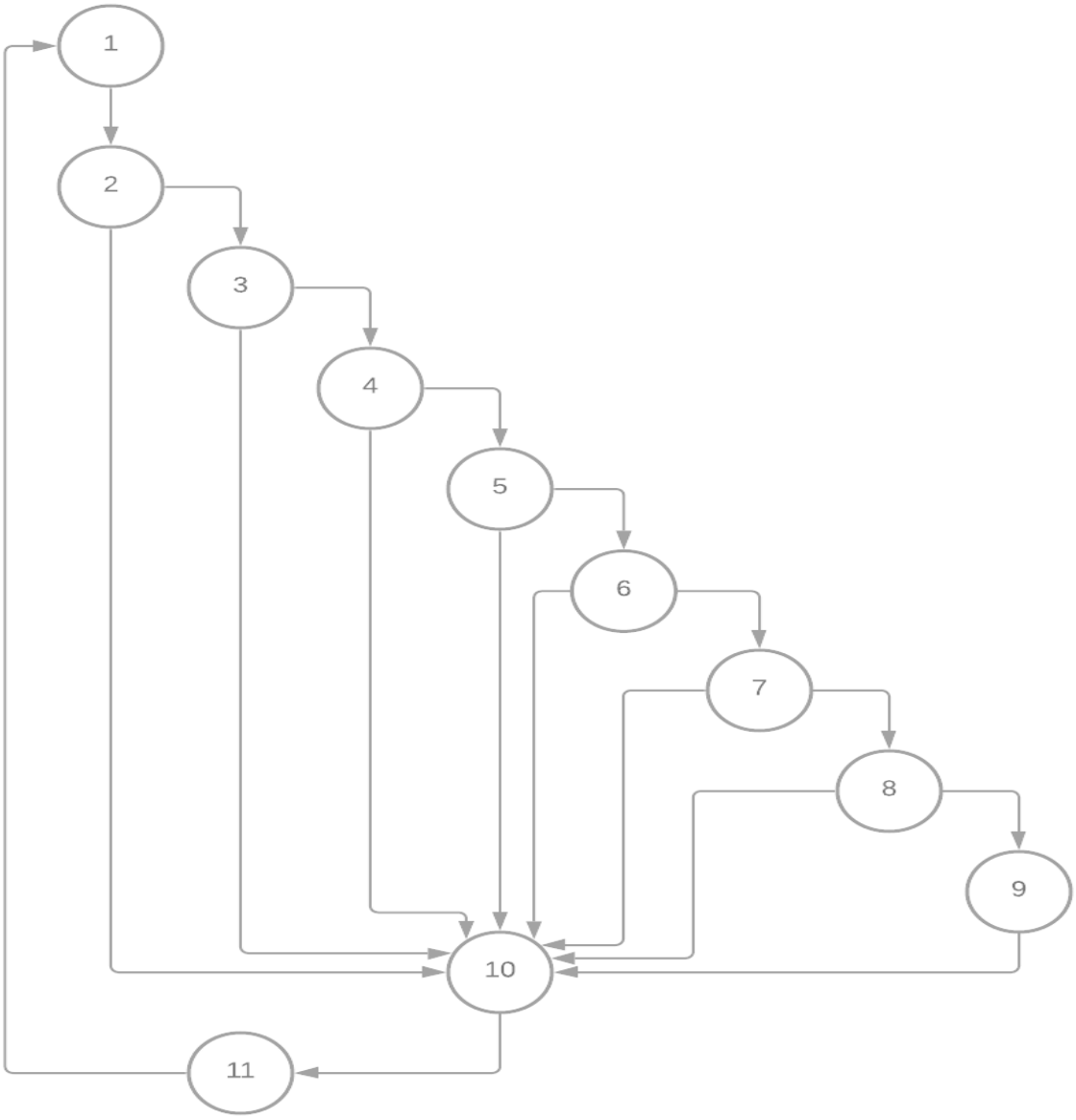
V(G) =k – d + 2p

k=Diyagramdaki kenar çizgi sayısı

d=diyagramdaki düğüm sayısı

p= programdaki bileşen sayısı (ana program ve ilgili alt program sayısını göstermektedir, alt program kullanılmadı ise p=1, n tane alt program kullanıldı ise p=(n+1) olur)

PROJEMİZDEKİ MCCABE KARMAŞIKLIK ÖLÇÜTÜ HESABI:



Şekil 5.3 McCabe algoritması

k=18 kenar sayısı

d=11 düğüm sayısı

p=3 bileşen sayısı

V(G)=18-11+6=13

5.5 Olağan Dışı Durum Çözümleme

Olağan dışı durumlar gerek kod yazım sürecinde gerekse testler sırasında gerçekleşebilir. Verilen hata kodlarının kısa sürede gözden geçirilmesi gerekir. Modüller olarak kodlanan programda bir modülde gerçekleşen hata diğerini etkilese bile çözümü kolay olacaktır. Sistemin çalışmasının; geçersiz, yanlış veri oluşumu veya başka nedenlerle istenmeyen bir biçimde sonlanmasına karşı önlemler alınmıştır. Hata yakalamak için sitemimizde try-catch kod blokları kullanılacaktır.

5.5.1 Olağandışı Durum Tanımları

Olağandışı durumlar, programlama dili tarafından tanımlı durumlar olabileceği gibi kullanıcı tarafından da tanımlanabilmektedir

5.5.2 Farklı Olağan Dışı Durum Çözümleme Yaklaşımları

Şekil 5.4 Çözümleme Yaklaşımları

- **Anında Durdurma:** Hata bulunduğunda program açıklayıcı bir ileti vererek sona erer. Bu iletinin yanı sıra 16'lık sistemde bir döküm de verir. MS Office paketlerinde alınan "This Program Performed an illegal Operation" iletisi ve ardından programın durması bu tür olağan dışı durum çözümleme yaklaşımına örnek olarak verilebilir. Bu yaklaşımda, verilen hata iletisinin anlamlı olması ve programcıya, düzeltilebilmek amacıyla yol gösterici olması önemlidir.
- **Hata Kodu Verme:** İlgili program modülünde başarısız bir şekilde çıktığında, programın bir hata kodu vermesi biçiminde bir yaklaşımdır. Söz konusu kod "başarılı" veya "başarısız" durumu belirten mantıksal bir kod olabileceği gibi bazı uygulamalarda sayısal ya da metin biçiminde olabilir. Bu yaklaşımda ilgili programda da hata oluştuğu anlaşılabilmekte ancak hatanın hangi deyimde oluştuğu anlaşılamamaktadır.
- **Tanımlı Olağandışı Yordam Çalıştırma:** Programlama dili platformunun tanımladığı olağan dışı durum çözümleme olanakları kullanmak biçimindedir. COBOL programlama dilindeki "ON READ EROR" deyimi gibi.
- **Hata Yordamı Çalıştırma (Olmayacak Değer Döndürme) :** Yordamın beklenmeyen bir değer geri döndürmesi gibi bir hesaplamadır. Örneğin yaş hesaplama yapıp, yaş bilgisini geri döndüren bir programın eksi değer döndürmesi gibi.

5.6 Kod Gözden Geçirme

Hiç kimse, önceki sürümlerini gözden geçirmeden ve incelemeden okunabilir bir program yazamaz. Hiçbir yazı, editörün onayını almadan basılmayacağı gibi hiçbir program da incelenmeden, gözden geçirilmeden işletime alınmamalıdır. Kod gözden geçirme ile program sınama işlemlerini birbirinden ayırmak gerekir.

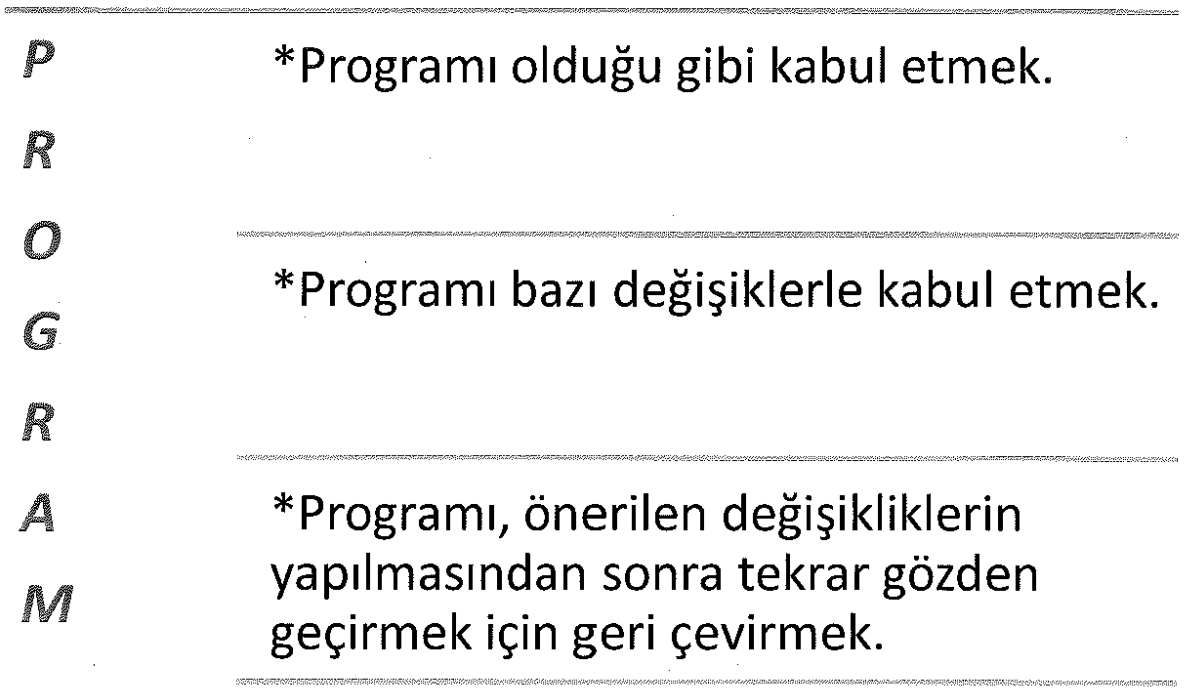
Program sınama, programın işletimi sırasında ortaya çıkabilecek yanlış ya da hataları yakalamak amacıyla yapılır. Kod gözden geçirme işlemi ise programın kaynak kodu üzerinde yapılan bir incelemedir. Kod gözden geçirmelerinde program hatalarının %3-5 oranındaki kesimi yakalanabilmektedir. Eğer programı yazan kişi, yazdığı programın hemen sonra bir "kod inceleme" sürecine gireceğini bilerek program yazdığında daha etkin, az hatalı ve okunabilir programlar elde edilebilmektedir.

5.6.1 Gözden Geçirme Sürecinin Düzenlemesi

Gözden geçirme sürecinin temel özellikleri;

- Hataların bulunması, ancak düzeltilmemesi hedeflenir
- Olabildiğince küçük bir grup tarafından yapılmalıdır. En iyi durum deneyimli bir inceleyici kullanılmasıdır. Birden fazla kişi gerektiğinde bu kişilerin, ileride program bakımı yapacak ekipten seçilmesinde yarar vardır
- Kalite çalışmalarının bir parçası olarak ele alınmalı ve sonuçlar düzenli ve belirlenen bir biçimde saklanmalıdır

Biçiminde özetlenebilir. Burada yanıtı aranan temel soru, programın yazıldığı gibi çalışıp çalışmayacağının belirlenmesidir. Gözden geçirme çalışmasının olası çıktıları;



Şekil 5.5 Gözden geçirme adımları

5.6.2 Gözden Geçirme Sırasında Kullanılacak Sorular

5.6.2.1 Öbek Ara Yüzü

- Her öbek tek bir işlevsel amacı yerine getiriyor mu?
- Öbek adı, işlecini açıklayacak biçimde anlamlı olarak verilmiş mi?
- Öbek tek giriş ve tek çıkışlı mı?
- Öbek eğer bir işlev ise parametrelerinin değerini değiştiriyor mu?

5.6.2.2 Giriş Açıklamaları

- Öbek, doğru biçimde giriş açıklama satırları içeriyor mu?
- Giriş açıklama satırları, öbeğin amacını açıklıyor mu?
- Giriş açıklama satırları, çıktıları (parametre, kütük, vb.) ve hata iletilerini tanımlıyor mu?
- Giriş açıklama satırları, öbeğin algoritma tanımını içeriyor mu?
- Giriş açıklama satırları, öbekte yapılan değişikliklere ilişkin tanımlamaları içeriyor mu?
- Giriş açıklama satırları, öbekteki olağan dışı durumları tamamlıyor mu?
- Giriş açıklama satırları, öbeği yazan kişi ve yazıldığı tarih ile ilgili bilgileri içeriyor mu?
- Her paragrafı açıklayan kısa açıklamalar var mı?

5.6.2.3 Veri Kullanımı

- İşlevsel olarak ilişkili bulunan veri elemanları uygun bir mantıksal veri yapısı içinde gruplanmış mı?
- Değişken adları, işlevlerini yansıtacak bicinde anlamlı mı?
- Her değişken tek bir amaçla mı kullanılıyor?
- Dizin değişkenleri kullanıldıkları dizinin sınırları içerisinde mi tanımlanmış?
- Tanımlanan her gösterge değişkeni için bellek ataması yapılmış mı?

5.6.2.4 Öbeğin Düzenlenişi

- Algoritmalar istenen işlevleri karşılıyor mu?
- Ara yüzler genel tasarımla uyumlu mu?
- Mantıksal karmaşıklık anlamlı mı?
- Veri yapısı çözümleme çalışması sırasında elde edilen veri modeli ile uyumlu mu?
- Belirlenen tasarım standartlarına uyulmuş mu?
- Hata çözümleme tanımlanmış mı?
- Tasarım, kullanılacak programlama diline uygun mu?
- İşletim sistemi ve programlama diline yönelik kısıtlar ya da özellikler kullanılmış mı?
- Bakım dikkate alınmış mı?

5.6.2.5 Sunuş

- Her satır, en fazla bir deyim içeriyor mu?
- Bir deyimin birden fazla satıra taşması durumunda bölünme anlaşıla bilirliği kolaylaştıracak biçimde anlamlı mı?
- Bütün deyimlerde, karmaşıklığı azaltacak şekilde parantezler kullanılmış mı?
- Bütün deyimler, belirlenen program stiline uygun olarak yazılmış mı?
- Öbek yapısı içerisinde akıllı "programlama hileleri" kullanılmış mı?

6. Doğrulama Ve Geçerleme

6.1 Giriş

Yapılan projelerde hatalar kaçınılmazdır. Bu projede de diğer yapılan projelerde olduğu gibi, yapılma aşamasında hatalar vardır. Bu hataları tespit etmeden önce, sorulması gereken iki soru vardır;

1. Doğru ürün mü üretiyoruz?
2. Ürünü doğru olarak mı üretiyoruz?

Müşterinin isterlerini dikkate alarak ve müşteri ile sürekli iletişim halinde olarak, doğru ürün üretildiği ve projenin çıktıları müşterinin isterlerini karşıladığı için doğru ürün üretildiği anlaşılmıştır.

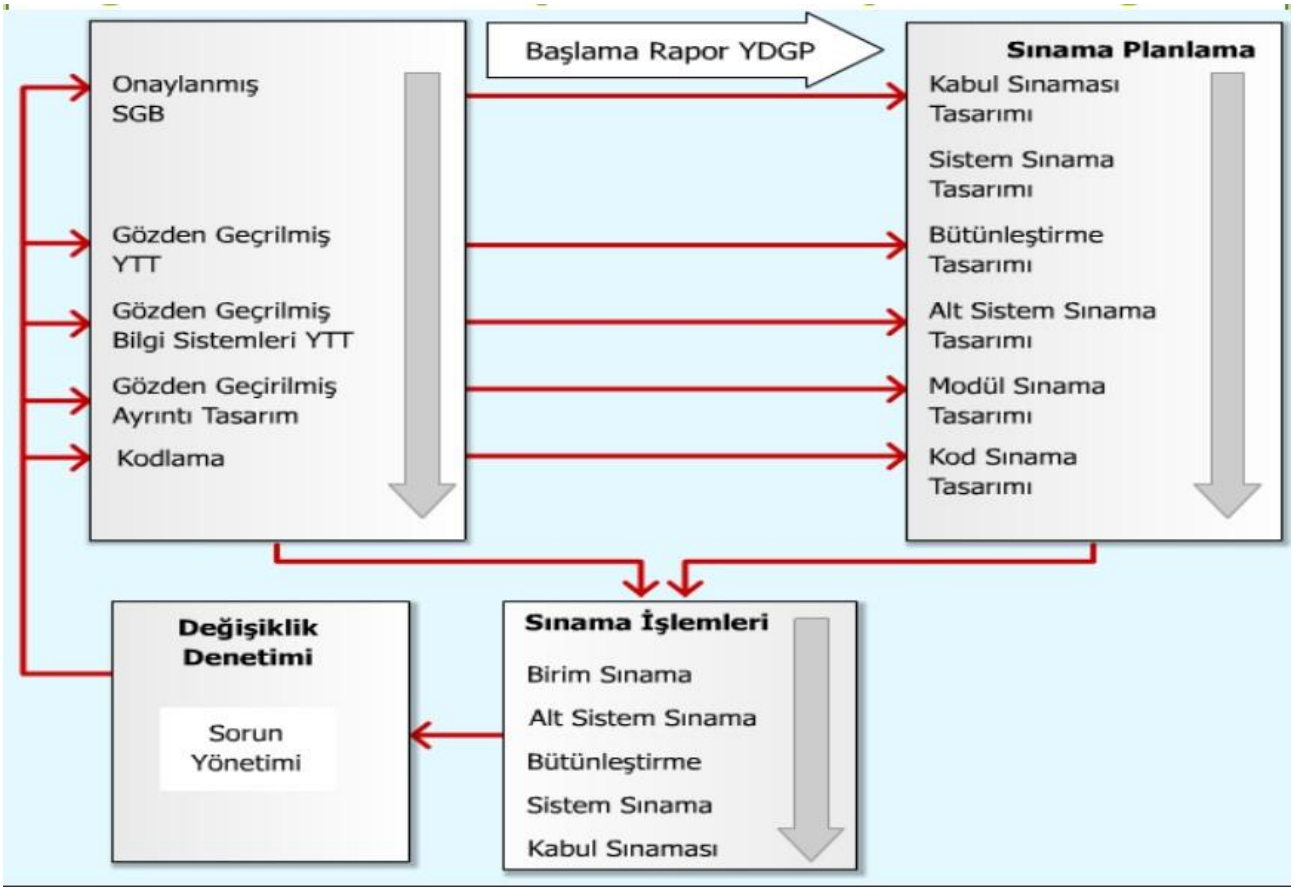
6.2 Sınama Kavramları

Projenin başarılı ve eksiksiz bir şekilde çalışabilmesi için bazı sınamalardan geçmiş olması gerekmektedir.

Bu sınamaların yapıla bilmesi için projeye en uygun sınama yöntemleri kullanılmıştır. Bunlar;

- **Birim Sınama:** Her bir metriğin istenilen sonucun verilip, verilmemesine bakılmıştır.
- **Alt Sistem Sınama:** Arayüz tasarlanırken kullanıcının kullanımına kolaylık sağlamak için, basit ve kullanılabilir bir arayüz tasarlamak istenmiştir. Bu aşamada birçok hata ile karşılaşılmıştır. Bu hatalar giderilerek sisteme zarar vermeden ve kullanıcının kullanımına kolaylık sağlayacak en iyi arayüz tasarlanmıştır.
- **Sistem Sınama:** Temel sınamalardan sonra birçok sistemde proje test edilmiştir ve çıkan hatalar düzeltilmiştir.
- **Kabul Sınama:** Projenin amacına göre farklı PDF'ler denenmiş çıkan hatalar giderilmişrtir.

6.3 Doğrulama ve Geçerleme Yaşam Döngüsü



Şekil 6.1 Doğrulama Yaşam Döngüsü

6.4 Sınama Yöntemleri

6.4.1 Beyaz Kutu Sınaması

Proje yapılırken her bir metriği teker teker incelenmiş, istenilen sonucun verilip verilmediğine bakılmış çıkan hatalar düzeltilmiştir.

6.5 Sınama Ve Bütünleştirme Stratejileri

Sınamalar Kademeli olarak gerçekleştirilmiştir. Bunlar;

1. **Yukarıdan Aşağı Sınama ve Bütünleştirme:**
   Sistem bir bütün haline getirildikten sonra müşterilerin isterlerini doğru karşılayıp karşılamadığına bakılır. Eğer karşılanmıyorsa alt sistemlere inerek hatalar düzeltildi.
2. **Aşağıdan Yukarıya Sınama ve Bütünleştirm**e:
   Sistem bir bütün haline getirilirken yazılan fonksiyonların her biri belli aşamalarda test edilerek bir bütün haline getirildi.

6.6 Sınama Planlaması

Test planı kimliği: Dokümantasyon kontrol testi.

Giriş: Projenin amacına göre herhangi bir Tez incelenirken, içerisindeki ana başlıklara göre sayfa numaralarına ayrılmıştır ve ona göre müşterinin isterleri incelenip test edilmiştir.

Test edilecek sistem: Proje başlangıç seviyesi olarak tek sürüm bulunduğundan, başka sürümlerin testine gerek duyulmamaktadır. Ayrıca ana sistemde bütün fonksiyonlar test edilmiştir.

Test edilecek ana fonksiyonlar:

1.Başlıktaki Her Kelimenin Baş Harfi Büyük Değil

2.Sekiller Listesindeki Şekil Tanımlaması Numarası İle Uyuşmuyor

3.Tablolar Listesindeki Tablo Tanımlaması Numarası İle Uyuşmuyor

4.Denklemler Listesindeki Denklem Tanımlaması Numarası İle Uyuşmuyor

5.Çizelgeler Listesindeki Çizelge Tanımlaması Numarası İle Uyuşmuyor

6.Referanslarda Tanımlanan Referans Tez İçerisinde Kullanılmamış

7.Giriş Sayfasının İlk Paragrafında Teşekkür İbaresi Yer Alıyor

8.İki Adet Çift Tırnak Arasında 50 Kelimeden Fazla Kelime Kullanılmış

9.Paragraf İki Satır ve/veya İki Satırdan Daha Az Satırdan Oluşuyor

10.İçindekiler Kısmındaki Sayfa Numaraları Doğru Değil

11.Tez İçerisinde Herhangi Bir Sorunla Karşılaşılmamıştır.

Her biri teker teker test edilmiştir.

Test edilmeyecek ana fonksiyonlar: Test edilmeyecek fonksiyon şuan için mevcut değildir.

Geçti/Kaldı Kriterleri: Projemiz başarı ile testlerden geçmiş ve istenen sonuca ulaşmıştır.

Test dokümanı: Her bir fonksiyon bütün testlerden başarı ile geçtikten sonra proje başarılı bir şekilde çalışmaktadır.

Sorumluluklar: Test takım lideri Akın Burak Yazlık.

Proje gerçekleştirilirken grup çalışması yapılmış ve sorunlar üzerine herkes dahil olmuştur.

Riskler ve Önlemler: Girdi olarak girilen sayfa sayılarının doğru olarak girilmesi gerekmektedir. Aksi takdirde sonuçları yanlış çıkacaktır.

6.7 Sınama Belirtimleri

Sınanan program modül isimleri:

Icındekiler, Sekiller, Tablolar, Denklemler, Referanslar, Cizelgeler, Giris, Icerik, Tez, PDFIslemleri, HataKontrolleri,

Sınama Türü;

Beyaz Kutu, Birim testi,

Sınama Verileri;

Proje bütün sınamaları başarı ile geçmiştir.

Sınama senaryoları;

Projede çoğunlukla sınıflar ve metotlar kullanıldığı için farklı sınama tekniklerine uygundur.

Sınamayı Yapan;

Proje Grup halinde yapıldığı için grup üyelerince sınamalar gerçekleştirilmiştir.

Sınama Tarihi;

Sınama 22/01/2021 tarihinde gerçekleştirilmiştir.

6.8 Yaşam Döngüsü Boyunca Sınama Etkinlikleri;

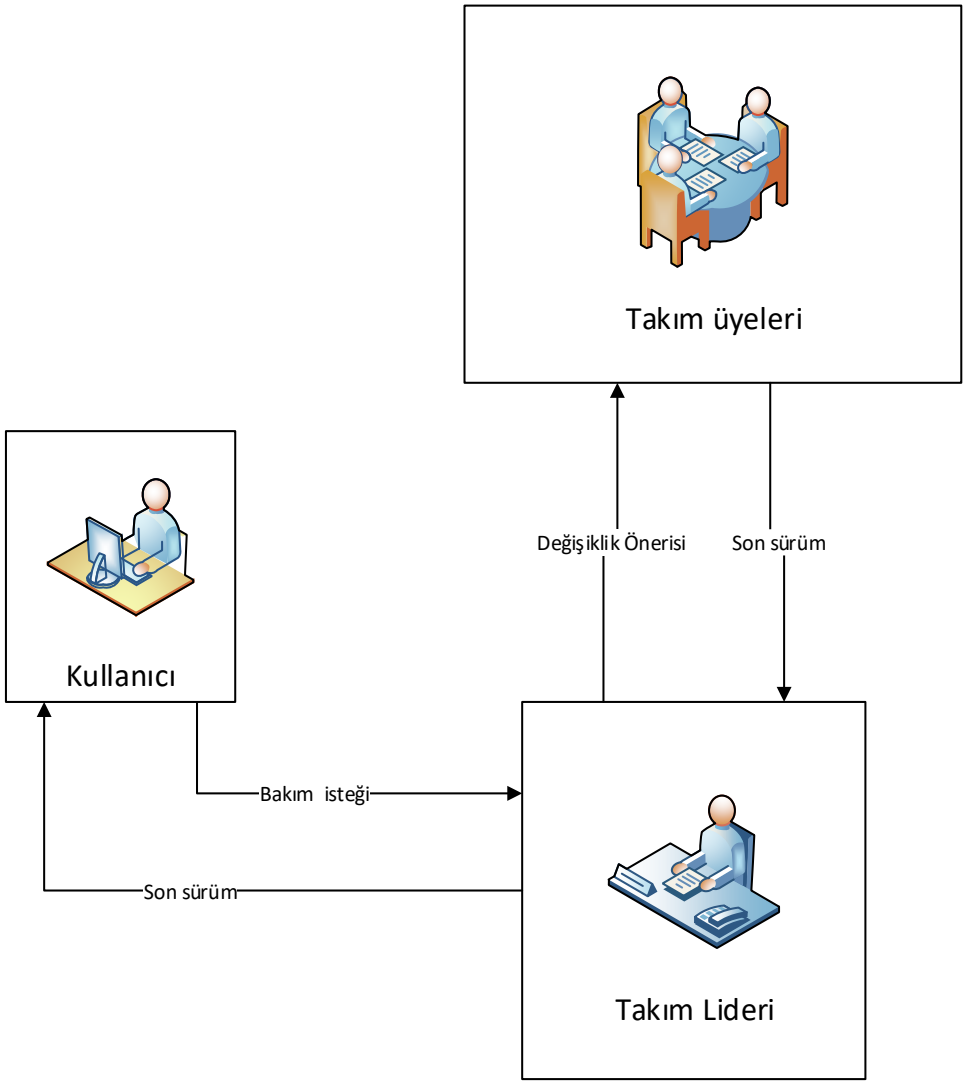
Proje sınama sırasında genellikle birim sınama uygulandığı için her aşamada sınanmıştır. Proje bir bütün halindeyken isterleri doğru karşılayıp karşılamadığı ayrı olarak sınanmıştır. Bulunan hatalar düzeltilmiştir.

7. Bakım

7.1 Giriş

Müşterinin isteği doğrultusunda projede güncellemeler veya bakım yapılabilmektedir.

Şekil 7.1 Bakım aşaması gösterilmiştir.



Şekil 7.1 Yazılım Bakım döngüsü

7.2 Kurulum

Proje internet bağlantısı ve veri tabanı gerektirmediği için özel bir kuruluma ihtiyaç duymamaktadır. Buna ek olarak projenin kullanılacak olan donanıma kurulumu videoda anlatılmaktadır.

7.3 Yerinde Destek Organizasyonu

Proje şuan kişiye özel olduğu için sadece video ve dokümantasyon sunulmaktadır. Eğer halka açık bir şekilde kullanıma açılmış ise belirli yollar izlenerek, yeni bir web sitesi, blog sayfası veya uygulama içerisinde kurulum adımlarının gösterildiği bir doküman ile sunulacaktır.

7.4 Yazılım Bakımı

7.4.1 Tanım

Proje teslim edildikten sonra çıkan hatalar Şekil 7.1 de görüldüğü üzere, kullanıcı takım lideri ile iletişime geçer ve isteklerini beyan eder. Takım lideri bu istekleri takım üyeleri ile paylaşır ve takım lideri takım üyeleri ile birlikte hataların giderilmesi üzerine çalışılır. Ardında hataların giderilmiş olan sürümü kullanıcıya ulaştırılır.

7.4.2 Bakım Süreç Modeli

Bakım süreç modelini adım adım incelersek;

1) Proje tesliminden sonra kullanıcı bir hata ile karşılaşırsa, çıkan hatayı bir rapor halinde takım liderine sunar.
2) Proje lideri raporu proje ekibine sunar.
3) Proje lideri ve ekibi raporu inceler ve hatalar üzerinde araştırma yapar.
4) Düzeltilen hataların testi gerçekleştirilir.
5) Test sonucunda hatalar giderilmiş ise proje güncellenir.
6) Güncellenen proje, proje lideri tarafından kullanıcıya sunulur.
7) Kullanıcı kurulumu gerçekleştirdikten sonra kullanılmaya hazırdır.

Bu süreç Şekil 7.1 de anlatılmıştır.

8. Sonuç

Ortaya çıkan ürün sayesinde gözden kaçan hatalar bulunacak, tez veya doküman incelemesi kolaylaşacaktır.

9. Kaynaklar